



Amazon EMR pada Panduan Pengembangan EKS

Amazon EMR



Amazon EMR: Amazon EMR pada Panduan Pengembangan EKS

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 antara para pelanggan, atau dengan cara apa pun yang menghina 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 EMR di EKS?	1
Arsitektur untuk Amazon EMR di EKS	2
Memahami Amazon EMR pada konsep dan terminologi EKS	3
Namespace Kubernetes	3
Klaster virtual	4
Tugas berjalan	4
Kontainer Amazon EMR	4
Apa yang terjadi ketika Anda mengirimkan pekerjaan ke EMR Amazon di klaster virtual EKS	5
Memulai dengan Amazon EMR di EKS	7
Jalankan aplikasi Spark	8
Praktik terbaik	14
Keamanan	14
Pengajuan pekerjaan Pyspark	14
Penyimpanan	14
Integrasi metastore	15
Debugging	15
Memecahkan masalah Amazon EMR pada masalah EKS	15
Penempatan simpul	15
Performa	15
Optimalisasi biaya	16
Menggunakan AWS Outposts	16
Menyesuaikan gambar Docker	17
Cara menyesuaikan gambar Docker	17
Prasyarat	18
Langkah 1: Ambil gambar dasar dari Amazon Elastic Container Registry (Amazon ECR)	18
Langkah 2: Sesuaikan gambar dasar	19
Langkah 3: (Opsional tapi disarankan) Validasi gambar kustom	20
Langkah 4: Publikasikan gambar khusus	21
Langkah 5: Kirim beban kerja Spark di Amazon EMR menggunakan gambar khusus	22
Sesuaikan gambar Docker untuk titik akhir interaktif	25
Bekerja dengan gambar multi-arsitektur	26
Detail untuk memilih URI gambar dasar	28
Akun registri Amazon ECR	29
Pertimbangan untuk menyesuaikan gambar	31

Lowongan kerja Running Flink	33
Operator Flink Kubernetes	33
Menyiapkan	34
Menginstal operator Flink Kubernetes	35
Jalankan aplikasi Flink	37
Izin peran keamanan untuk menjalankan aplikasi Flink	41
Menghapus instalasi operator	43
Kubernetes Asli Flink	44
Menyiapkan	44
Memulai	45
Persyaratan keamanan	47
Menyesuaikan gambar Docker untuk Flink dan FluentD	48
Prasyarat	48
Ambil gambar dasar dari Amazon Elastic Container Registry	49
Sesuaikan gambar dasar	49
Publikasikan gambar kustom Anda	50
Kirim beban kerja Flink	51
Memantau	52
Menggunakan Amazon Managed Service untuk Prometheus	52
Menggunakan UI Flink	54
Menggunakan konfigurasi pemantauan	55
Bagaimana Flink mendukung ketersediaan tinggi dan ketahanan kerja	60
Menggunakan ketersediaan tinggi	60
Mengoptimalkan waktu restart	66
Penonaktifan anggun	73
Menggunakan Autoscaler	76
Autotuning parameter Autoscaler	78
Pemeliharaan dan pemecahan masalah untuk pekerjaan Flink di Amazon EMR di EKS	87
Mempertahankan aplikasi Flink	87
Pemecahan masalah	89
Rilis yang didukung	93
Lowongan kerja Running Spark	94
StartJobRun	94
Penyiapan	95
Kirim pekerjaan yang dijalankan dengan StartJobRun	125
Menggunakan klasifikasi pengirim pekerjaan	127

Menggunakan klasifikasi default penampung EMR Amazon	133
Operator percikan	136
Menyiapkan	137
Memulai	138
Penskalaan otomatis vertikal	142
Hapus instalasi	147
Menggunakan konfigurasi pemantauan untuk memantau Spark	147
Keamanan	155
spark-submit	166
Menyiapkan	166
Memulai	167
Keamanan	168
Apache Livy	174
Menyiapkan	174
Memulai	175
Menjalankan aplikasi Spark	180
Menghapus instalasi	182
Keamanan	183
Properti instalasi	193
Memecahkan masalah kesalahan format variabel lingkungan yang umum	198
Mengelola pekerjaan berjalan	199
Kelola dengan CLI	199
Jalankan skrip Spark SQL	206
Status tugas berjalan	208
Lihat pekerjaan di konsol	209
Kesalahan menjalankan pekerjaan umum	210
Menggunakan template pekerjaan	216
Membuat dan menggunakan template pekerjaan untuk memulai pekerjaan	217
Mendefinisikan parameter template pekerjaan	219
Mengontrol akses ke templat pekerjaan	221
Menggunakan templat pod	222
Skenario umum	223
Mengaktifkan templat pod dengan Amazon EMR di EKS	225
Bidang templat pod	227
Pertimbangan kontainer sidecar	230
Menggunakan kebijakan coba lagi	232

Menetapkan kebijakan coba lagi	232
Mengambil status kebijakan	234
Memantau tugas	235
Temukan log driver	236
Menggunakan rotasi log peristiwa Spark	236
Menggunakan rotasi log kontainer Spark	237
Menggunakan penskalaan otomatis vertikal	239
Menyiapkan	240
Memulai	243
Konfigurasi	245
Memantau rekomendasi	251
Menghapus instalasi	252
Menjalankan beban kerja interaktif	253
Ikhtisar titik akhir interaktif	253
Prasyarat titik akhir interaktif	255
AWS CLI	255
eksctl	256
Kluster Amazon EKS	256
Akses Grant Cluster	256
Aktifkan peran IAM untuk Akun Layanan	257
Buat peran eksekusi pekerjaan IAM	257
Berikan akses kepada pengguna	257
Daftarkan cluster Amazon EKS dengan Amazon EMR	258
Pengontrol Load Balancer	258
Membuat Endpoint Interaktif	258
Buat titik akhir interaktif	258
Tentukan parameter khusus	259
.....	260
Parameter titik akhir interaktif	260
Mengkonfigurasi pengaturan untuk titik akhir interaktif	262
Lowongan kerja Monitoring Spark	262
Templat pod kustom	264
Menerapkan pod JEG ke grup node	264
Opsi konfigurasi JEG	268
Memodifikasi parameter PySpark	269
Gambar kernel kustom	269

Memantau titik akhir interaktif	271
Contoh	274
Menggunakan notebook Jupyter yang dihosting sendiri	274
Membuat grup keamanan	275
Buat titik akhir interaktif	275
Dapatkan URL server gateway	276
Dapatkan token autentikasi	276
Menyebarkan notebook	277
Bersihkan	282
Mendapatkan informasi tentang titik akhir interaktif dengan perintah CLI	283
.....	283
Daftar titik akhir interaktif	284
Hapus titik akhir interaktif	286
Mengunggah data	287
Prasyarat	287
Memulai	287
Pemantauan tugas	290
Pantau pekerjaan dengan Amazon CloudWatch Events	290
Otomatiskan Amazon EMR di EKS dengan Acara CloudWatch	291
Contoh: Mengatur aturan yang memanggil Lambda	292
Pantau pod driver job dengan kebijakan coba lagi menggunakan Amazon CloudWatch Events	293
Mengelola kluster virtual	294
Membuat kluster virtual	294
Daftar kluster virtual	296
Gambarkan kluster virtual	296
Menghapus kluster virtual	296
Status kluster virtual	296
Tutorial	297
Menggunakan Danau Delta	297
Menggunakan Iceberg	298
Konfigurasi sesi percikan untuk integrasi katalog	299
Menggunakan PyFlink	300
Menggunakan AWS Glue dengan Flink	301
Menggunakan Apache Hudi	304
Kirim pekerjaan Apache Hudi	304

Menggunakan Spark RAPIDS	308
Menggunakan Spark pada Redshift	312
Luncurkan aplikasi Spark	313
Otentikasi ke Amazon Redshift	314
Baca dan tulis ke Amazon Redshift	316
Pertimbangan-pertimbangan	318
Menggunakan Volcano	319
Ikhtisar	319
Penginstalan	319
Kirim: Operator Spark	321
Kirim: spark-submit	323
Menggunakan YuniKorn	324
Ikhtisar	324
Buat klaster Anda	324
Instal YuniKorn	326
Kirim: Operator Spark	327
Kirim: spark-submit	330
Keamanan	14
Praktik terbaik	333
Terapkan prinsip hak istimewa paling rendah	333
Daftar kontrol akses untuk titik akhir	333
Dapatkan pembaruan keamanan terbaru untuk gambar kustom	334
Batasi akses kredensial pod	334
Mengisolasi kode aplikasi yang tidak tepercaya	334
Izin kontrol akses berbasis peran (RBAC)	334
Membatasi akses ke nodegroup IAM role atau kredensial profil instans	335
Perlindungan data	335
Enkripsi saat istirahat	336
Enkripsi saat bergerak	339
Manajemen Identitas dan Akses	340
Audiens	340
Mengautentikasi dengan identitas	341
Mengelola akses menggunakan kebijakan	342
Bagaimana Amazon EMR di EKS bekerja dengan IAM	344
Menggunakan Peran Terkait Layanan	349
Kebijakan terkelola untuk Amazon EMR di EKS	353

Menggunakan peran eksekusi tugas dengan Amazon EMR di EKS	355
Contoh kebijakan berbasis identitas	357
Kebijakan untuk kendali akses berbasis tanda	360
Pemecahan Masalah	364
Menggunakan Amazon EMR di EKS dengan Lake Formation AWS	366
Bagaimana Amazon EMR di EKS bekerja dengan Lake Formation AWS	366
Aktifkan Lake Formation dengan Amazon EMR di EKS	368
Pertimbangan dan batasan	377
Pemecahan masalah	379
Pencatatan dan pemantauan	381
Mengkripsi log	382
CloudTrail log	385
Pemberian Akses S3	388
Ikhtisar	388
Luncurkan cluster	389
Pertimbangan-pertimbangan	390
Validasi Kepatuhan	390
Ketahanan	390
Keamanan Infrastruktur	391
Konfigurasi dan analisis kerentanan	391
Titik akhir VPC antarmuka	392
Buat Kebijakan VPC Endpoint untuk Amazon EMR di EKS	393
Akses lintas akun	396
Prasyarat	396
Cara mengakses bucket Amazon S3 lintas akun atau tabel DynamoDB	396
Penandaan Sumber Daya	401
Dasar-dasar tag	401
Beri tanda pada sumber daya Anda	402
Pembatasan tag	403
Bekerja dengan tag menggunakan AWS CLI dan Amazon EMR di EKS API	404
Pemecahan masalah	15
Kegagalan pekerjaan PVC	405
Verifikasi	405
Patch	406
Patch manual	410
Kegagalan penskalaan otomatis vertikal	412

403 Kesalahan terlarang	412
Namespace tidak ditemukan	412
Kesalahan kredensial Docker	413
Kegagalan operator percikan	413
Pemasangan bagan helm gagal	413
Pengecualian sistem file yang tidak didukung	414
Titik akhir dan kuota layanan	415
Titik akhir layanan	415
Kuota layanan	417
Lihat kuota dan permintaan kuota meningkat	419
Versi rilis	420
rilis emr-spark-8.0.0	421
Rilis	422
Catatan rilis	422
Perubahan dan fitur	423
7.13.0 rilis	424
Rilis	424
Catatan rilis	425
Perubahan dan fitur	427
emr-7.13.0-terbaru	427
emr-7.13.0-20260410	427
emr-7.13.0-flink-terbaru	428
emr-7.13.0-batu pipi-20260410	428
7.12.0 rilis	428
Rilis	428
Catatan rilis	430
Perubahan dan fitur	431
emr-7.12.0-terbaru	432
emr-7.12.0-20251111	432
emr-7.12.0-flink-terbaru	432
emr-7.12.0-batu pipi-20251111	432
7.11.0 rilis	433
Rilis	433
Catatan rilis	435
Perubahan dan fitur	436
emr-7.11.0-terbaru	437

emr-7.11.0-20251020	437
emr-7.11.0-flink-terbaru	437
emr-7.11.0-batu pipi-20251020	437
7.10.0 rilis	437
Rilis	438
Catatan rilis	439
Perubahan dan fitur	441
emr-7.10.0-terbaru	441
emr-7.10.0-20250801	441
emr-7.10.0-flink-terbaru	441
emr-7.10.0-batu pipi-20250801	442
7.9.0 rilis	442
Rilis	442
Catatan rilis	444
Perubahan	445
emr-7.9.0-terbaru	445
emr-7.9.0-20250425	446
emr-7.9.0-flink-terbaru	446
emr-7.9.0-batu pipi-20250425	446
7.8.0 rilis	446
Rilis	446
Catatan rilis	448
Perubahan	450
emr-7.8.0-terbaru	450
emr-7.8.0-20250228	450
emr-7.8.0-flink-terbaru	450
emr-7.8.0-batu pipi-20250228	451
7.7.0 rilis	451
Rilis	451
Catatan rilis	453
Perubahan	454
emr-7.7.0-terbaru	454
emr-7.7.0-20250131	455
emr-7.7.0-flink-terbaru	455
emr-7.7.0-batu pipi-20250131	455
7.6.0 rilis	455

Rilis	456
Catatan rilis	457
Fitur	459
Perubahan	459
emr-7.6.0-terbaru	459
emr-7.6.0-20241213	459
emr-7.6.0-flink-terbaru	460
emr-7.6.0-batu pipi-20241213	460
7.5.0 rilis	460
Rilis	460
Catatan rilis	460
7.4.0 rilis	461
Rilis	461
Catatan rilis	461
7.3.0 rilis	461
Rilis	462
Catatan rilis	463
Fitur	465
Perubahan	465
emr-7.3.0-terbaru	465
emr-7.3.0-20240920	466
emr-7.3.0-flink-terbaru	466
emr-7.3.0-batu pipi-29240920	466
7.2.0 rilis	466
Rilis	466
Catatan rilis	468
Fitur	470
emr-7.2.0-terbaru	470
emr-7.2.0-20240610	471
emr-7.2.0-flink-terbaru	471
emr-7.2.0-batu pipi-20240610	471
7.1.0 rilis	471
Rilis	472
Catatan rilis	473
Fitur	475
emr-7.1.0-terbaru	475

emr-7.1.0-20240321	475
emr-7.1.0-flink-terbaru	475
emr-7.1.0-batu pipi-20240321	476
7.0.0 rilis	476
Rilis	476
Catatan rilis	478
Fitur	479
Perubahan	479
emr-7.0.0-terbaru	480
emr-7.0.0-2024321	480
emr-7.0.0-20231211	480
emr-7.0.0-flink-terbaru	480
emr-7.0.0-batu pipi-2024321	480
emr-7.0.0-batu pipi-20231211	481
6.15.0 rilis	481
Rilis	481
Catatan rilis	483
Fitur	484
emr-6.15.0-terbaru	485
emr-6.15.0-20240105	485
emr-6.15.0-20231109	485
emr-6.15.0-flink-terbaru	485
emr-6.15.0-batu pipi-20240105	486
emr-6.15.0-batu pipi-20231109	486
6.14.0 rilis	486
Rilis	486
Catatan rilis	488
Fitur	489
emr-6.14.0-terbaru	489
emr-6.14.0-20231005	489
6.13.0 rilis	490
Rilis	490
Catatan rilis	491
Fitur	493
emr-6.13.0-terbaru	493
emr-6.13.0-20230814	493

6.12.0 rilis	494
Rilis	494
Catatan rilis	495
Fitur	496
emr-6.12.0-terbaru	496
emr-6.12.0-20240321	496
emr-6.12.0-20230701	497
6.11.0 rilis	497
Rilis	497
Catatan rilis	498
Fitur	499
emr-6.11.0-terbaru	500
emr-6.11.0-20230905	500
emr-6.11.0-20230509	500
6.10.0 rilis	500
emr-6.10.0-terbaru	503
emr-6.10.0-20230905	503
emr-6.10.0-20230624	503
emr-6.10.0-20230421	504
emr-6.10.0-20230403	504
emr-6.10.0-20230220	504
6.9.0 rilis	504
emr-6.9.0-terbaru	507
emr-6.9.0-20230905	507
emr-6.9.0-20230624	507
emr-6.9.0-20221108	508
6.8.0 rilis	508
emr-6.8.0-terbaru	512
emr-6.8.0-20230905	512
emr-6.8.0-20230624	512
emr-6.8.0-20221219	513
emr-6.8.0-20220802	513
6.7.0 rilis	513
emr-6.7.0-terbaru	515
emr-6.7.0-20240321	515
emr-6.7.0-20230624	515

emr-6.7.0-20221219	516
emr-6.7.0-20220630	516
6.6.0 rilis	516
emr-6.6.0-terbaru	517
emr-6.6.0-20240321	518
emr-6.6.0-20230624	518
emr-6.6.0-20221219	518
emr-6.6.0-20220411	518
6.5.0 rilis	519
emr-6.5.0-terbaru	520
emr-6.5.0-20240321	520
emr-6.5.0-20221219	521
emr-6.5.0-20220802	521
emr-6.5.0-20211119	521
6.4.0 rilis	521
emr-6.4.0-terbaru	522
emr-6.4.0-20240321	523
emr-6.4.0-20221219	523
emr-6.4.0-20210830	523
6.3.0 rilis	523
emr-6.3.0-terbaru	525
emr-6.3.0-20240321	525
emr-6.3.0-20220802	525
emr-6.3.0-20211008	525
emr-6.3.0-20210802	526
emr-6.3.0-20210429	526
6.2.0 rilis	526
emr-6.2.0-latest	527
emr-6.2.0-20240321	528
emr-6.2.0-20220802	528
emr-6.2.0-20211008	528
emr-6.2.0-20210802	528
emr-6.2.0-20210615	529
emr-6.2.0-20210129	529
emr-6.2.0-20201218	529
emr-6.2.0-20201201	529

5.36.0 rilis	530
emr-5.36.0-terbaru	531
emr-5.36.0-20240321	531
emr-5.36.0-20221219	531
emr-5.36.0-20220620	531
emr-5.36.0-20220525	532
5.35.0 rilis	532
emr-5.35.0-terbaru	533
emr-5.35.0-20240321	533
emr-5.35.0-20221219	534
emr-5.35.0-20220802	534
emr-5.35.0-20220307	534
5.34 rilis	534
emr-5.34.0-terbaru	535
emr-5.34.0-20240321	536
emr-5.34.0-20220802	536
emr-5.34.0-20211208	536
5.33.0 rilis	536
emr-5.33.0-latest	538
emr-5.33.0-20240321	538
emr-5.33.0-20221219	538
emr-5.33.0-20220802	538
emr-5.33.0-20211008	539
emr-5.33.0-20210802	539
emr-5.33.0-20210615	539
emr-5.33.0-20210323	539
5.32.0 rilis	540
emr-5.32.0-latest	541
emr-5.32.0-20240321	541
emr-5.32.0-20220802	541
emr-5.32.0-20211008	542
emr-5.32.0-20210802	542
emr-5.32.0-20210615	542
emr-5.32.0-20210129	542
emr-5.32.0-20201218	543
emr-5.32.0-20201201	543

Riwayat dokumen	544
.....	dxlvii

Apa itu Amazon EMR di EKS?

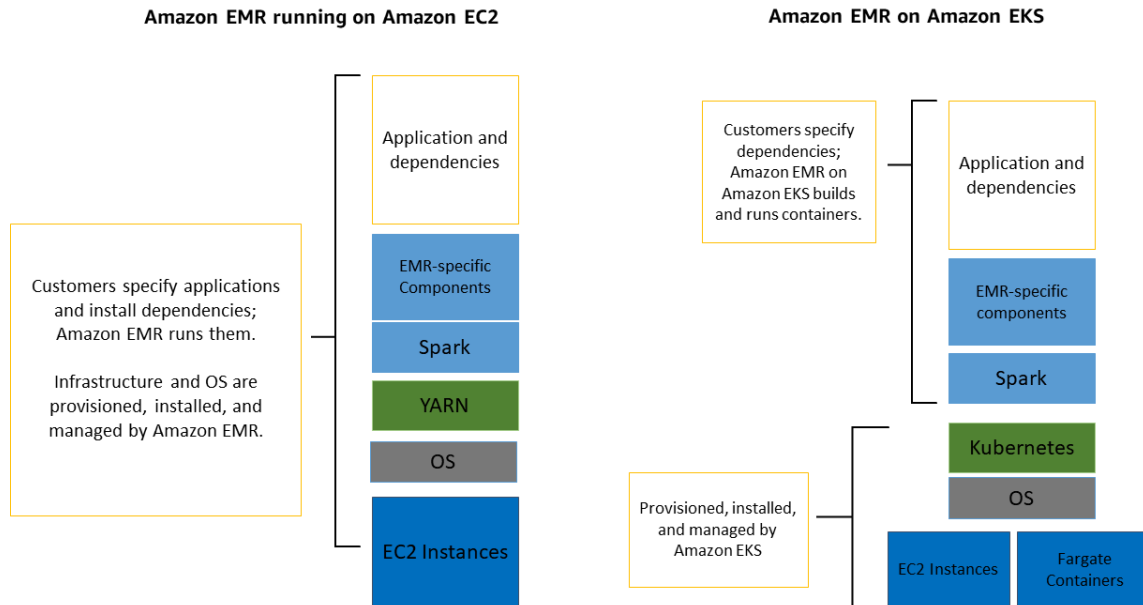
Amazon EMR di EKS menyediakan opsi penyebaran untuk Amazon EMR yang memungkinkan Anda untuk menjalankan kerangka kerja big data sumber terbuka di Amazon Elastic Kubernetes Service (Amazon EKS). Dengan opsi penyebaran ini, Anda dapat fokus pada menjalankan beban kerja analitik sementara Amazon EMR di EKS membangun, mengonfigurasi, dan mengelola kontainer untuk aplikasi sumber terbuka.

Jika Anda sudah menggunakan Amazon EMR, Anda sekarang dapat menjalankan aplikasi berbasis Amazon EMR dengan jenis aplikasi lain pada kluster Amazon EKS yang sama. Opsi penyebaran ini juga meningkatkan pemanfaatan sumber daya dan menyederhanakan manajemen infrastruktur di beberapa Availability Zones. Jika Anda sudah menjalankan kerangka kerja big data di Amazon EKS, Anda sekarang dapat menggunakan Amazon EMR untuk mengotomatiskan penyediaan dan manajemen, dan menjalankan Apache Spark lebih cepat.

Amazon EMR di EKS memungkinkan tim Anda untuk berkolaborasi lebih efisien dan memproses sejumlah besar data dengan lebih mudah dan hemat biaya:

- Anda dapat menjalankan aplikasi pada kolam umum sumber daya tanpa harus menyediakan infrastruktur. Anda dapat menggunakan [Amazon EMR Studio](#) dan AWS SDK atau AWS CLI untuk mengembangkan, mengirimkan, dan mendiagnosis aplikasi analitik yang berjalan di kluster EKS. Anda dapat menjalankan tugas terjadwal di Amazon EMR di EKS menggunakan Apache Airflow terkelola mandiri atau Amazon Managed Workflows for Apache Airflow (MWAA).
- Tim infrastruktur dapat mengelola platform komputasi umum secara terpusat untuk mengkonsolidasikan beban kerja Amazon EMR dengan aplikasi berbasis kontainer lainnya. Anda dapat menyederhanakan manajemen infrastruktur dengan alat umum Amazon EKS dan memanfaatkan kluster bersama untuk beban kerja yang membutuhkan versi kerangka kerja sumber terbuka yang berbeda. Anda juga dapat mengurangi overhead operasional dengan manajemen kluster Kubernetes otomatis dan patching OS. Dengan Amazon EC2 dan AWS Fargate, Anda dapat mengaktifkan beberapa sumber daya komputasi untuk memenuhi persyaratan kinerja, operasional, atau keuangan.

Diagram berikut menunjukkan dua model penyebaran yang berbeda untuk Amazon EMR.



Topik

- [Arsitektur untuk Amazon EMR di EKS](#)
- [Memahami Amazon EMR pada konsep dan terminologi EKS](#)
- [Apa yang terjadi ketika Anda mengirimkan pekerjaan ke EMR Amazon di kluster virtual EKS](#)

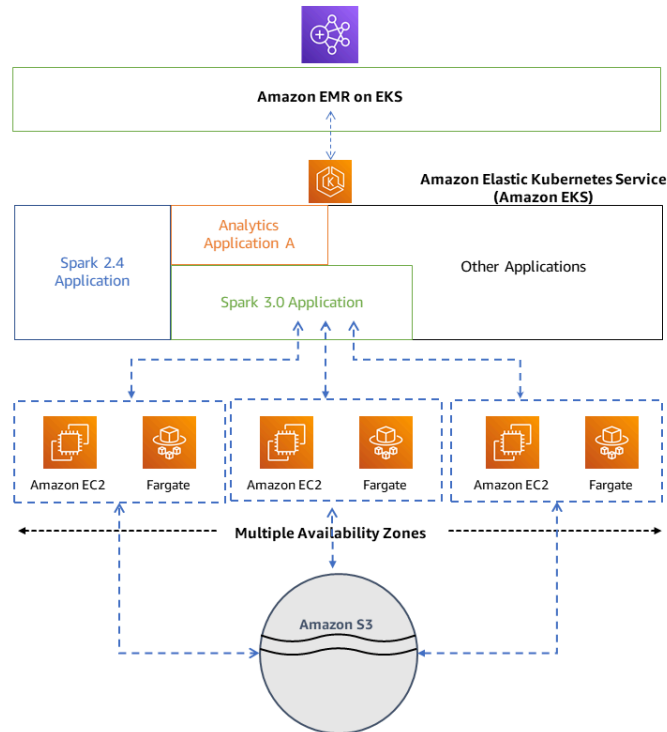
Arsitektur untuk Amazon EMR di EKS

Amazon EMR di EKS menggabungkan dengan longgar aplikasi untuk infrastruktur tempat aplikasi berjalan. Setiap lapisan infrastruktur menyediakan orkestrasi untuk lapisan berikutnya. Ketika Anda mengirimkan tugas ke Amazon EMR, definisi tugas Anda berisi semua parameter khusus aplikasi. Amazon EMR menggunakan parameter ini untuk menginstruksikan Amazon EKS tentang yang pod dan kontainer mana yang disebar. Amazon EKS kemudian menghadirkan sumber daya komputasi dari Amazon EC2 secara online dan AWS Fargate diperlukan untuk menjalankan pekerjaan itu.

Dengan penggabungan longgar layanan, Anda dapat menjalankan beberapa tugas yang terisolasi dengan aman secara bersamaan. Anda juga dapat mengukur tugas yang sama dengan backend

komputasi yang berbeda atau menyebarkan tugas Anda di beberapa Availability Zones yang sama untuk meningkatkan ketersediaan.

Diagram berikut menggambarkan bagaimana Amazon EMR di EKS bekerja dengan AWS layanan lain.



Memahami Amazon EMR pada konsep dan terminologi EKS

Amazon EMR di EKS menyediakan opsi penyebaran untuk Amazon EMR yang memungkinkan Anda untuk menjalankan kerangka kerja big data sumber terbuka di Amazon Elastic Kubernetes Service (Amazon EKS). Topik ini memberi Anda konteks tentang beberapa terminologi umum untuk itu, termasuk ruang nama, cluster virtual, dan menjalankan pekerjaan, yang merupakan unit pekerjaan yang Anda kirimkan untuk diproses.

Namespace Kubernetes

Amazon EKS menggunakan namespace Kubernetes untuk membagi sumber daya kluster antara beberapa pengguna dan aplikasi. Namespace ini adalah dasar untuk lingkungan multi-penyewa. Namespace Kubernetes dapat memiliki Amazon EC2 atau sebagai penyedia komputasi. AWS

Fargate Fleksibilitas ini memberi Anda pilihan performa dan biaya yang berbeda untuk dijalankan pada tugas Anda.

Klaster virtual

Sebuah klaster virtual adalah namespace Kubernetes tempat Amazon EMR terdaftar. Amazon EMR menggunakan klaster virtual untuk menjalankan tugas dan meng-host titik akhir. Beberapa klaster virtual dapat didukung oleh klaster fisik yang sama. Namun, setiap klaster virtual memetakan ke satu namespace pada klaster EKS. Klaster virtual tidak membuat sumber daya aktif apa pun yang berkontribusi pada tagihan Anda atau yang memerlukan manajemen siklus hidup di luar layanan.

Tugas berjalan

Job run adalah unit kerja, seperti jar Spark, PySpark skrip, atau kueri SparkSQL, yang Anda kirimkan ke Amazon EMR di EKS. Satu tugas dapat memiliki beberapa tugas berjalan. Ketika Anda mengirimkan tugas berjalan, Anda menyertakan informasi berikut:

- Sebuah klaster virtual di mana tugas harus berjalan.
- Sebuah nama tugas untuk mengidentifikasi tugas.
- Peran eksekusi — IAM role tercakup yang menjalankan tugas dan memungkinkan Anda untuk menentukan sumber daya mana yang dapat diakses oleh tugas.
- Label rilis Amazon EMR yang menentukan versi aplikasi sumber terbuka untuk digunakan.
- Artefak yang digunakan saat mengirimkan tugas Anda, seperti parameter spark-submit.

Secara default, log diunggah ke server Riwayat Spark dan dapat diakses dari Konsol Manajemen AWS. Anda juga dapat mendorong log peristiwa, log eksekusi, dan metrik ke Amazon S3 dan Amazon CloudWatch

Kontainer Amazon EMR

Kontainer Amazon EMR adalah [Nama API untuk Amazon EMR di EKS](#). Prefiks `emr-containers` digunakan dalam skenario berikut:

- Ini adalah prefiks dalam perintah CLI untuk Amazon EMR di EKS. Sebagai contoh, `aws emr-containers start-job-run`.
- Ini adalah prefiks sebelum tindakan kebijakan IAM untuk Amazon EMR di EKS. Misalnya, `"Action": ["emr-containers:StartJobRun"]`. Untuk informasi selengkapnya, lihat [Tindakan kebijakan untuk Amazon EMR di EKS](#).

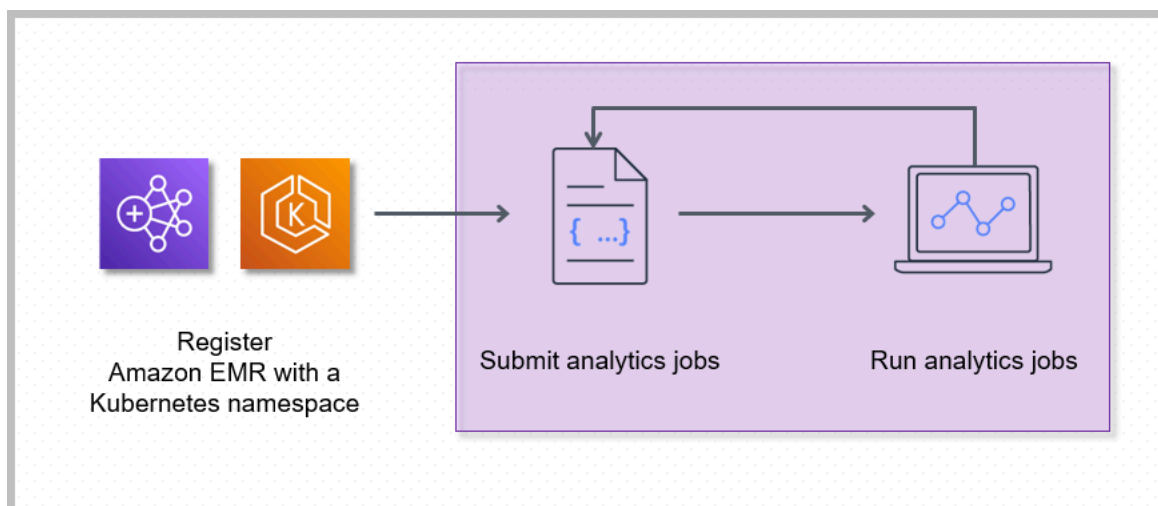
- Ini adalah prefiks yang digunakan di Amazon EMR pada titik akhir layanan EKS. Misalnya, `emr-containers.us-east-1.amazonaws.com`. Untuk informasi selengkapnya, lihat [Amazon EMR pada Titik Akhir Layanan EKS](#).

Apa yang terjadi ketika Anda mengirimkan pekerjaan ke EMR Amazon di kluster virtual EKS

Mendaftarkan Amazon EMR dengan namespace Kubernetes pada Amazon EKS membuat kluster virtual. Amazon EMR kemudian dapat menjalankan beban kerja analitik pada namespace tersebut. Saat Anda menggunakan Amazon EMR di EKS untuk mengirimkan tugas Spark ke kluster virtual, Amazon EMR di EKS meminta penjadwal Kubernetes di Amazon EKS untuk menjadwalkan pod.

Langkah-langkah dan diagram berikut menggambarkan alur kerja Amazon EMR di EKS:

- Menggunakan kluster Amazon EKS yang ada atau membuat kluster dengan menggunakan utilitas baris perintah `eksctl` atau konsol Amazon EKS.
- Buat kluster virtual dengan mendaftarkan Amazon EMR dengan namespace pada kluster EKS.
- Kirimkan pekerjaan Anda ke kluster virtual menggunakan AWS CLI atau SDK.



Untuk setiap tugas yang Anda jalankan, Amazon EMR di EKS menciptakan sebuah kontainer dengan gambar dasar Amazon Linux 2, Apache Spark, dan dependensi terkait. Setiap tugas berjalan dalam pod yang mengunduh kontainer dan mulai menjalankannya. Pod berakhir setelah tugas berakhir. Jika gambar kontainer sebelumnya telah disebar ke simpul, maka gambar yang di-cache digunakan dan unduhan dilewati. Kontainer sidecar, seperti penerus log atau metrik, dapat disebar ke pod.

Setelah tugas berakhir, Anda masih dapat melakukan debug menggunakan UI aplikasi Spark di konsol Amazon EMR.

Memulai dengan Amazon EMR di EKS

Topik ini membantu Anda mulai menggunakan Amazon EMR di EKS dengan menerapkan aplikasi Spark pada cluster virtual. Ini termasuk langkah-langkah untuk mengatur izin yang benar dan untuk memulai pekerjaan. Sebelum memulai, pastikan bahwa Anda telah menyelesaikan langkah-langkah dalam [Menyiapkan Amazon EMR di EKS](#). Ini membantu Anda mendapatkan alat seperti AWS CLI pengaturan sebelum membuat cluster virtual Anda. Untuk templat lain yang dapat membantu Anda memulai, lihat [Panduan Praktik Terbaik Kontainer EMR](#) kami di GitHub

Anda memerlukan informasi berikut dari langkah-langkah penyiapan:

- ID klaster virtual untuk klaster Amazon EKS dan namespace Kubernetes terdaftar dengan Amazon EMR

Important

Saat membuat cluster EKS, pastikan untuk menggunakan m5.xlarge sebagai tipe instance, atau jenis instance lainnya dengan CPU dan memori yang lebih tinggi. Menggunakan tipe instans dengan CPU atau memori yang lebih rendah dari m5.xlarge dapat menyebabkan kegagalan pekerjaan karena sumber daya yang tidak mencukupi yang tersedia di cluster.

- Nama IAM role yang digunakan untuk eksekusi tugas
- Label rilis untuk rilis Amazon EMR (misalnya, emr-6.4.0-latest)
- Target tujuan untuk pencatatan dan pemantauan:
 - Nama grup CloudWatch log Amazon dan awalan aliran log
 - Lokasi Amazon S3 untuk menyimpan peristiwa dan log kontainer

Important

Amazon EMR pada pekerjaan EKS menggunakan Amazon CloudWatch dan Amazon S3 sebagai target tujuan untuk pemantauan dan pencatatan. Anda dapat memantau kemajuan pekerjaan dan memecahkan masalah kegagalan dengan melihat log pekerjaan yang dikirim ke tujuan ini. Untuk mengaktifkan logging, kebijakan IAM yang terkait dengan peran IAM untuk pelaksanaan pekerjaan harus memiliki izin yang diperlukan untuk mengakses sumber daya target. Jika kebijakan IAM tidak memiliki izin yang diperlukan, Anda harus mengikuti langkah-langkah yang diuraikan, [Mengonfigurasi proses Perbarui kebijakan kepercayaan dari](#)

[peran eksekusi tugas pekerjaan untuk menggunakan log Amazon S3](#), dan [Mengonfigurasi pekerjaan yang dijalankan untuk menggunakan CloudWatch Log sebelum menjalankan tugas sampel ini](#).

Jalankan aplikasi Spark

Ambil langkah-langkah berikut untuk menjalankan aplikasi Spark sederhana di Amazon EMR di EKS. `entryPointFile` aplikasi untuk aplikasi Spark Python terletak di.

`s3://REGION.elasticmapreduce/emr-containers/samples/wordcount/scripts/wordcount.py` *REGION* ini adalah Wilayah di mana EMR Amazon Anda di kluster virtual EKS berada, seperti *us-east-1*

1. Perbarui kebijakan IAM untuk peran pelaksanaan pekerjaan dengan izin yang diperlukan, seperti yang ditunjukkan oleh pernyataan kebijakan berikut.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadFromLoggingAndInputScriptBuckets",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::*elasticmapreduce",
        "arn:aws:s3:::*elasticmapreduce/*",
        "arn:aws:s3:::amzn-s3-demo-bucket",
        "arn:aws:s3:::amzn-s3-demo-bucket/*",
        "arn:aws:s3:::amzn-s3-demo-bucket-b",
        "arn:aws:s3:::amzn-s3-demo-bucket-b/*"
      ]
    },
    {
      "Sid": "WriteToLoggingAndOutputDataBuckets",
      "Effect": "Allow",
```

```

    "Action": [
      "s3:PutObject",
      "s3>DeleteObject"
    ],
    "Resource": [
      "arn:aws:s3:::amzn-s3-demo-bucket/*",
      "arn:aws:s3:::amzn-s3-demo-bucket-b/*"
    ]
  },
  {
    "Sid": "DescribeAndCreateCloudwatchLogStream",
    "Effect": "Allow",
    "Action": [
      "logs:CreateLogStream",
      "logs:DescribeLogGroups",
      "logs:DescribeLogStreams"
    ],
    "Resource": [
      "arn:aws:logs:*:*:*"
    ]
  },
  {
    "Sid": "WriteToCloudwatchLogs",
    "Effect": "Allow",
    "Action": [
      "logs:PutLogEvents"
    ],
    "Resource": [
      "arn:aws:logs:*:*:log-group:my_log_group_name:log-
stream:my_log_stream_prefix/*"
    ]
  }
]
}

```

- Pernyataan pertama `ReadFromLoggingAndInputScriptBuckets` dalam kebijakan ini memberikan `ListBucket` dan `GetObject`s akses ke bucket Amazon S3 berikut:
 - `REGION.elasticmapreduce-` ember tempat `entryPoint` file aplikasi berada.
 - `amzn-s3-demo-destination-bucket-` ember yang Anda tentukan untuk data keluaran Anda.
 - `amzn-s3-demo-logging-bucket-` ember yang Anda tentukan untuk data logging Anda.

- Pernyataan kedua `WriteToLoggingAndOutputDataBuckets` dalam kebijakan ini memberikan izin pekerjaan untuk menulis data ke bucket keluaran dan logging Anda masing-masing.
 - Pernyataan ketiga `DescribeAndCreateCloudwatchLogStream` memberikan pekerjaan dengan izin untuk mendeskripsikan dan membuat Amazon CloudWatch Logs.
 - Pernyataan keempat `WriteToCloudwatchLogs` memberikan izin untuk menulis log ke grup CloudWatch log Amazon bernama *my_log_group_name* di bawah aliran log bernama *my_log_stream_prefix*
2. Untuk menjalankan aplikasi Spark Python, gunakan perintah berikut. Ganti semua nilai yang dapat diganti dengan *red italicized* nilai yang sesuai. *REGION* ini adalah Wilayah di mana EMR Amazon Anda di kluster virtual EKS berada, seperti *us-east-1*

```
aws emr-containers start-job-run \
--virtual-cluster-id cluster_id \
--name sample-job-name \
--execution-role-arn execution-role-arn \
--release-label emr-6.4.0-latest \
--job-driver '{
  "sparkSubmitJobDriver": {
    "entryPoint": "s3://REGION.elasticmapreduce/emr-containers/samples/wordcount/
scripts/wordcount.py",
    "entryPointArguments": ["s3://amzn-s3-demo-destination-bucket/
wordcount_output"],
    "sparkSubmitParameters": "--conf spark.executor.instances=2 --
conf spark.executor.memory=2G --conf spark.executor.cores=2 --conf
spark.driver.cores=1"
  }
}' \
--configuration-overrides '{
  "monitoringConfiguration": {
    "cloudWatchMonitoringConfiguration": {
      "logGroupName": "my_log_group_name",
      "logStreamNamePrefix": "my_log_stream_prefix"
    },
    "s3MonitoringConfiguration": {
      "logUri": "s3://amzn-s3-demo-logging-bucket"
    }
  }
}'
```

Data output dari pekerjaan ini akan tersedia di `s3://amzn-s3-demo-destination-bucket/wordcount_output`.

Anda juga dapat membuat file JSON dengan parameter tertentu untuk menjalankan tugas Anda. Kemudian jalankan perintah `start-job-run` dengan jalur ke file JSON. Untuk informasi selengkapnya, lihat [Kirim pekerjaan yang dijalankan dengan StartJobRun](#). Untuk detail lebih lanjut tentang mengonfigurasi parameter untuk menjalankan tugas, lihat [Pilihan untuk mengonfigurasi tugas berjalan](#).

3. Untuk menjalankan aplikasi Spark SQL, gunakan perintah berikut. Ganti semua *red italicized* nilai dengan nilai yang sesuai. *REGION* ini adalah Wilayah di mana EMR Amazon Anda di kluster virtual EKS berada, seperti *us-east-1*

```
aws emr-containers start-job-run \
--virtual-cluster-id cluster_id \
--name sample-job-name \
--execution-role-arn execution-role-arn \
--release-label emr-6.7.0-latest \
--job-driver '{
  "sparkSqlJobDriver": {
    "entryPoint": "s3://query-file.sql",
    "sparkSqlParameters": "--conf spark.executor.instances=2 --
conf spark.executor.memory=2G --conf spark.executor.cores=2 --conf
spark.driver.cores=1"
  }
}' \
--configuration-overrides '{
  "monitoringConfiguration": {
    "cloudWatchMonitoringConfiguration": {
      "logGroupName": "my_log_group_name",
      "logStreamNamePrefix": "my_log_stream_prefix"
    },
    "s3MonitoringConfiguration": {
      "logUri": "s3://amzn-s3-demo-logging-bucket"
    }
  }
}'
```

Contoh file query SQL ditunjukkan di bawah ini. Anda harus memiliki penyimpanan file eksternal, seperti S3, tempat data untuk tabel disimpan.

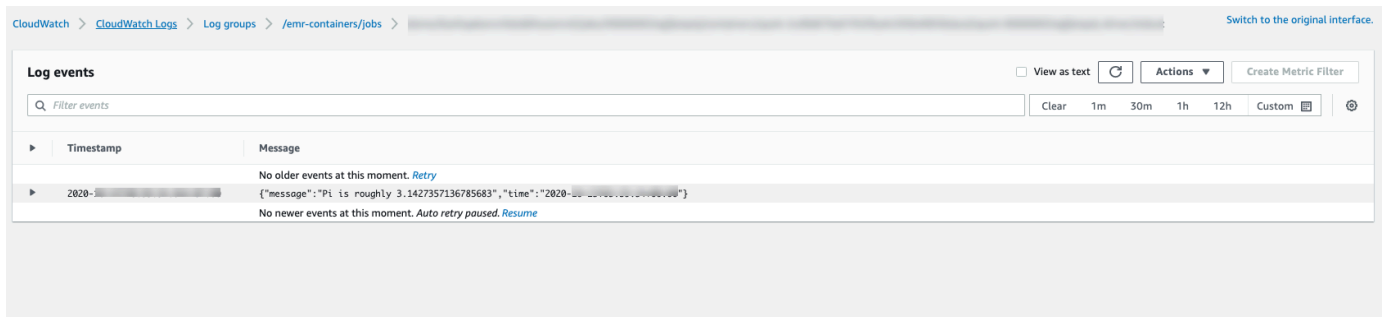
```
CREATE DATABASE demo;
CREATE EXTERNAL TABLE IF NOT EXISTS demo.amazonreview( marketplace string,
  customer_id string, review_id string, product_id string, product_parent string,
  product_title string, star_rating integer, helpful_votes integer, total_votes
  integer, vine string, verified_purchase string, review_headline string,
  review_body string, review_date date, year integer) STORED AS PARQUET LOCATION
  's3://URI to parquet files';
SELECT count(*) FROM demo.amazonreview;
SELECT count(*) FROM demo.amazonreview WHERE star_rating = 3;
```

Output untuk pekerjaan ini akan tersedia di log stdout driver di S3 atau CloudWatch, tergantung pada `monitoringConfiguration` yang dikonfigurasi.

- Anda juga dapat membuat file JSON dengan parameter tertentu untuk menjalankan tugas Anda. Kemudian jalankan perintah `start-job-run` dengan jalur ke file JSON. Untuk informasi selengkapnya, lihat [Mengirimkan pekerjaan](#). Untuk detail selengkapnya tentang mengonfigurasi parameter job run, lihat [Opsi untuk mengonfigurasi job run](#).

Untuk memantau kemajuan pekerjaan atau kegagalan debug, Anda dapat memeriksa log yang diunggah ke Amazon S3, Log, CloudWatch atau keduanya. Lihat jalur log di Amazon S3 di [Konfigurasi pekerjaan yang dijalankan untuk menggunakan log S3 dan untuk log](#) Cloudwatch di [Konfigurasi pekerjaan](#) yang dijalankan untuk menggunakan Log. CloudWatch Untuk melihat log di CloudWatch Log, ikuti petunjuk di bawah ini.

- Buka CloudWatch konsol di <https://console.aws.amazon.com/cloudwatch/>.
- Di panel Navigasi, pilih Log. Lalu pilih Grup log.
- Pilih grup log untuk Amazon EMR di EKS kemudian lihat log acara yang diunggah.



⚠ Important

Pekerjaan memiliki [kebijakan coba ulang yang dikonfigurasi secara default](#). Untuk informasi tentang cara mengubah atau menonaktifkan konfigurasi, lihat [Menggunakan kebijakan coba ulang pekerjaan](#).

Tautan ke Amazon EMR pada panduan praktik terbaik EKS GitHub

Kami telah membangun [EMR Amazon di Panduan Praktik Terbaik EKS](#) menggunakan kolaborasi komunitas open source sehingga kami dapat mengulangi dengan cepat dan memberikan rekomendasi untuk aspek pembuatan dan menjalankan kluster virtual. Kami menyarankan Anda menggunakan [Amazon EMR pada panduan praktik terbaik EKS](#) untuk bagian tersebut. Pilih tautan di setiap bagian untuk pergi ke GitHub situs.

Keamanan

Note

Untuk informasi lebih lanjut tentang keamanan dengan Amazon EMR di EKS, lihat. [Amazon EMR pada praktik terbaik keamanan EKS](#)

[Praktik terbaik enkripsi](#): cara menggunakan enkripsi untuk data saat istirahat dan dalam perjalanan.

[Mengelola keamanan jaringan](#) menjelaskan cara mengonfigurasi grup keamanan untuk pod untuk Amazon EMR di EKS saat Anda terhubung ke sumber data yang di-host seperti Amazon RDS dan Layanan AWS Amazon Redshift.

[Mengggunakan manajer AWS rahasia untuk menyimpan rahasia](#).

Pengajuan pekerjaan Pyspark

[Pengajuan pekerjaan Pyspark](#): menentukan berbagai jenis kemasan untuk aplikasi PySpark menggunakan format kemasan seperti zip, egg, wheel, dan pex.

Penyimpanan

[Mengggunakan volume EBS](#):: cara menggunakan penyediaan statis dan dinamis untuk pekerjaan yang membutuhkan volume EBS.

[Mengggunakan Amazon FSx untuk volume Lustre](#): cara menggunakan penyediaan statis dan dinamis untuk pekerjaan yang membutuhkan Amazon FSx untuk volume Luster.

[Menggunakan volume penyimpanan Instance](#): cara menggunakan volume penyimpanan instance untuk pemrosesan pekerjaan.

Integrasi metastore

[Menggunakan Hive metastore](#): menawarkan berbagai cara untuk menggunakan metastore Hive.

[Menggunakan AWS Glue](#): menawarkan berbagai cara untuk mengkonfigurasi katalog AWS Glue.

Debugging

[Menggunakan debugging Spark](#): cara mengubah level log.

[Menghubungkan ke Spark UI pada pod driver](#).

[Cara menggunakan server riwayat Spark yang dihosting sendiri dengan Amazon EMR](#) di EKS.

Memecahkan masalah Amazon EMR pada masalah EKS

[Pemecahan masalah](#).

Penempatan simpul

[Menggunakan pemilih node Kubernetes](#) untuk single-az dan kasus penggunaan lainnya.

[Menggunakan penempatan simpul Fargate](#).

Performa

[Menggunakan Dynamic Resource Allocation \(DRA\)](#).

Secara default, `spark.dynamicAllocation.preallocateExecutors` diaktifkan di Amazon EMR Spark. Kapan `spark.dynamicAllocation.initialExecutors` dan tidak `spark.dynamicAllocation.minExecutors` disetel, Spark dapat meminta sejumlah besar pelaksana saat startup berdasarkan perkiraan jumlah tugas, bahkan untuk beban kerja kecil. Untuk menghindari churn kontainer yang berlebihan, gunakan salah satu pendekatan berikut:

- Tetapkan `spark.dynamicAllocation.initialExecutors` atau `spark.dynamicAllocation.minExecutors` ke nilai yang sesuai untuk ukuran beban kerja Anda.
- Setel `spark.dynamicAllocation.preallocateExecutors.maxEstimatedTasks` ke nilai yang lebih rendah untuk membatasi jumlah pelaksana yang diminta saat startup.
- Setel `spark.dynamicAllocation.preallocateExecutors` `false` untuk menonaktifkan prealokasi eksekutor sepenuhnya.

[Praktik terbaik EKS](#) untuk plugin Amazon VPC Container Network Interface (CNI), Cluster Autoscaler, dan Core DNS.

Optimalisasi biaya

[Menggunakan instans spot](#): Praktik terbaik instans spot Amazon EC2 dan cara menggunakan fitur dekomisi simpul Spark.

Menggunakan AWS Outposts

[Menjalankan Amazon EMR di EKS menggunakan AWS Outposts](#)

Menyesuaikan gambar Docker untuk Amazon EMR di EKS

Anda dapat menggunakan gambar Docker yang disesuaikan dengan Amazon EMR di EKS.

Menyesuaikan Amazon EMR pada gambar waktu aktif EKS memberikan manfaat sebagai berikut:

- Menggabungkan ketergantungan aplikasi dan lingkungan waktu aktif menjadi kontainer tetap tunggal yang mempromosikan portabilitas dan menyederhanakan manajemen ketergantungan untuk setiap beban kerja.
- Menginstal dan mengkonfigurasi paket yang dioptimalkan untuk beban kerja Anda. Paket ini mungkin tidak tersedia secara luas dalam distribusi publik dari waktu aktif Amazon EMR.
- Integrasikan Amazon EMR pada EKS dengan proses membangun, menguji, dan penyebaran yang ada saat ini dalam organisasi Anda, termasuk pengembangan dan pengujian lokal.
- Terapkan proses keamanan yang telah ditetapkan, seperti pemindaian gambar, yang memenuhi persyaratan kepatuhan dan tata kelola dalam organisasi Anda.

Topik

- [Cara menyesuaikan gambar Docker](#)
- [Detail untuk memilih URI gambar dasar](#)
- [Pertimbangan untuk menyesuaikan gambar](#)

Cara menyesuaikan gambar Docker

Ikuti langkah-langkah ini untuk menyesuaikan gambar Docker untuk Amazon EMR di EKS. Langkah-langkah menunjukkan cara mendapatkan gambar dasar, menyesuaikan dan mempublikasikannya, dan mengirimkan beban kerja menggunakan gambar.

- [Prasyarat](#)
- [Langkah 1: Ambil gambar dasar dari Amazon Elastic Container Registry \(Amazon ECR\)](#)
- [Langkah 2: Sesuaikan gambar dasar](#)
- [Langkah 3: \(Opsional tapi disarankan\) Validasi gambar kustom](#)
- [Langkah 4: Publikasikan gambar khusus](#)
- [Langkah 5: Kirim beban kerja Spark di Amazon EMR menggunakan gambar khusus](#)

Note

Opsi lain yang mungkin ingin Anda pertimbangkan saat menyesuaikan gambar Docker adalah menyesuaikan untuk titik akhir interaktif, yang Anda lakukan untuk memastikan Anda memiliki dependensi yang diperlukan, atau menggunakan gambar wadah multi-arsitektur:

- [Sesuaikan gambar Docker untuk titik akhir interaktif](#)
- [Bekerja dengan gambar multi-arsitektur](#)

Prasyarat

- Selesaikan [Menyiapkan Amazon EMR di EKS](#) langkah-langkah untuk Amazon EMR di EKS.
- Instal Docker di lingkungan Anda. Untuk informasi lebih lanjut, lihat [Get Docker](#).

Langkah 1: Ambil gambar dasar dari Amazon Elastic Container Registry (Amazon ECR)

Gambar dasar berisi runtime Amazon EMR dan konektor yang digunakan untuk mengakses layanan lain. AWS Untuk Amazon EMR 6.9.0 dan yang lebih tinggi, Anda bisa mendapatkan gambar dasar dari Galeri Publik Amazon ECR. Jelajahi galeri untuk menemukan tautan gambar dan tarik gambar ke ruang kerja lokal Anda. Misalnya, untuk rilis Amazon EMR 7.13.0, `docker pull` perintah berikut memberi Anda gambar dasar standar terbaru. Anda dapat mengganti `emr-7.13.0:latest` dengan `emr-7.13.0-spark-rapids:latest` untuk mengambil gambar yang memiliki akselerator Nvidia RAPIDS. Anda juga dapat mengganti `emr-7.13.0:latest` dengan `emr-7.13.0-java11:latest` untuk mengambil gambar dengan runtime Java 11.

```
docker pull public.ecr.aws/emr-on-eks/spark/emr-7.13.0:latest
```

Jika Anda ingin mengambil gambar dasar untuk Amazon EMR 6.9.0 atau rilis ealier, atau jika Anda lebih suka mengambil dari akun registri Amazon ECR di setiap Wilayah, gunakan langkah-langkah berikut:

1. Pilih URI gambar dasar. URI gambar mengikuti format ini, *ECR-registry-account.dkr.ecr.Region.amazonaws.com/spark/container-image-tag*, seperti yang ditunjukkan contoh berikut.

```
895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.6.0:latest
```

Untuk memilih gambar dasar di Wilayah Anda, lihat [Detail untuk memilih URI gambar dasar](#).

2. Masuk ke repositori Amazon ECR di mana gambar dasar disimpan. Ganti `895885662937` dan `us-west-2` dengan akun registri Amazon ECR dan AWS Wilayah yang telah Anda pilih.

```
aws ecr get-login-password --region us-west-2 | docker login --username AWS --password-stdin 895885662937.dkr.ecr.us-west-2.amazonaws.com
```

3. Tarik gambar dasar ke Workspace lokal Anda. Ganti `emr-6.6.0:latest` dengan tag gambar kontainer yang telah Anda pilih.

```
docker pull 895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.6.0:latest
```

Langkah 2: Sesuaikan gambar dasar

Ikuti langkah-langkah ini untuk menyesuaikan gambar dasar yang telah Anda tarik dari Amazon ECR.

1. Buat Dockerfile baru di Workspace lokal Anda.
2. Edit Dockerfile yang baru saja Anda buat dan tambahkan konten berikut. Dockerfile ini menggunakan gambar kontainer yang telah Anda tarik dari `895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.6.0:latest`.

```
FROM 895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.6.0:latest
USER root
### Add customization commands here ###
USER hadoop:hadoop
```

3. Tambahkan perintah di Dockerfile untuk menyesuaikan gambar dasar. Sebagai contoh, tambahkan perintah untuk menginstal pustaka Python, seperti yang ditunjukkan Dockerfile berikut.

```
FROM 895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.6.0:latest
USER root
RUN pip3 install --upgrade boto3 pandas numpy // For python 3
USER hadoop:hadoop
```

4. Dari direktori yang sama di mana Dockerfile dibuat, jalankan perintah berikut untuk membangun gambar Docker. Berikan nama untuk image Docker, misalnya, *emr6.6_custom*.

```
docker build -t emr6.6_custom .
```

Langkah 3: (Opsional tapi disarankan) Validasi gambar kustom

Kami menyarankan Anda menguji kompatibilitas gambar kustom Anda sebelum menerbitkannya. Anda dapat menggunakan [EMR Amazon pada CLI gambar khusus EKS](#) untuk memeriksa apakah gambar Anda memiliki struktur file yang diperlukan dan konfigurasi yang benar untuk berjalan di Amazon EMR di EKS.

Note

Amazon EMR pada CLI gambar khusus EKS tidak dapat mengonfirmasi bahwa gambar Anda bebas dari kesalahan. Berhati-hatilah saat menghapus dependensi dari gambar dasar.

Ambil langkah-langkah berikut untuk memvalidasi gambar kustom Anda.

1. Unduh dan instal Amazon EMR pada CLI gambar khusus EKS. Untuk informasi selengkapnya, lihat [Amazon EMR di Panduan Instalasi CLI gambar khusus EKS](#).
2. Jalankan perintah berikut untuk menguji instalasi.

```
emr-on-eks-custom-image --version
```

Berikut ini menunjukkan contoh output.

```
Amazon EMR on EKS Custom Image CLI  
Version: x.xx
```

3. Jalankan perintah berikut untuk memvalidasi gambar kustom Anda.

```
emr-on-eks-custom-image validate-image -i image_name -r release_version [-  
t image_type]
```

- -i menentukan URI gambar lokal yang perlu divalidasi. Ini bisa berupa URI gambar, nama atau tag apa pun yang Anda tentukan untuk gambar Anda.

- -rmenentukan versi rilis yang tepat untuk gambar dasar, misalnya, `emr-6.6.0-latest`.
- -tmenentukan jenis gambar. Jika ini adalah gambar Spark, masukanspark. Nilai default-nya adalah spark. EMR Amazon saat ini pada versi CLI gambar khusus EKS hanya mendukung gambar runtime Spark.

Jika Anda menjalankan perintah dengan sukses dan gambar kustom memenuhi semua konfigurasi dan struktur file yang diperlukan, output yang dikembalikan menampilkan hasil dari semua pengujian, seperti contoh berikut menunjukkan.

```
Amazon EMR on EKS Custom Image Test
Version: x.xx
... Checking if docker cli is installed
... Checking Image Manifest
[INFO] Image ID: xxx
[INFO] Created On: 2021-05-17T20:50:07.986662904Z
[INFO] Default User Set to hadoop:hadoop : PASS
[INFO] Working Directory Set to /home/hadoop : PASS
[INFO] Entrypoint Set to /usr/bin/entrypoint.sh : PASS
[INFO] SPARK_HOME is set with value: /usr/lib/spark : PASS
[INFO] JAVA_HOME is set with value: /etc/alternatives/jre : PASS
[INFO] File Structure Test for spark-jars in /usr/lib/spark/jars: PASS
[INFO] File Structure Test for hadoop-files in /usr/lib/hadoop: PASS
[INFO] File Structure Test for hadoop-jars in /usr/lib/hadoop/lib: PASS
[INFO] File Structure Test for bin-files in /usr/bin: PASS
... Start Running Sample Spark Job
[INFO] Sample Spark Job Test with local:///usr/lib/spark/examples/jars/spark-
examples.jar : PASS
-----
Overall Custom Image Validation Succeeded.
-----
```

Jika gambar kustom tidak memenuhi konfigurasi atau struktur file yang diperlukan, pesan kesalahan akan terjadi. Output yang dikembalikan memberikan informasi tentang konfigurasi atau struktur file yang salah.

Langkah 4: Publikasikan gambar khusus

Publikasikan gambar Docker baru ke registri Amazon ECR Anda.

1. Jalankan perintah berikut untuk membuat repositori Amazon ECR untuk menyimpan gambar Docker Anda. Berikan nama untuk repositori Anda, misalnya, `emr6.6_custom_repo` Ganti `us-west-2` dengan Wilayah Anda.

```
aws ecr create-repository \  
  --repository-name emr6.6_custom_repo \  
  --image-scanning-configuration scanOnPush=true \  
  --region us-west-2
```

Untuk informasi lebih lanjut, lihat [Membuat repositori](#) dalam Panduan Pengguna Amazon ECR.

2. Jalankan perintah berikut untuk mengautentikasi ke registri default Anda.

```
aws ecr get-login-password --region us-west-2 | docker login --username AWS --  
password-stdin aws_account_id.dkr.ecr.us-west-2.amazonaws.com
```

Untuk informasi selengkapnya, lihat [Autentikasi ke registri default Anda](#) di Panduan Pengguna Amazon ECR.

3. Tandai dan publikasikan gambar ke repositori Amazon ECR yang Anda buat.

Tandai gambar.

```
docker tag emr6.6_custom aws_account_id.dkr.ecr.us-  
west-2.amazonaws.com/emr6.6_custom_repo
```

Tekan gambar.

```
docker push aws_account_id.dkr.ecr.us-west-2.amazonaws.com/emr6.6_custom_repo
```

Untuk informasi selengkapnya, lihat [Menekan gambar ke Amazon ECR](#) di Panduan Pengguna Amazon ECR.

Langkah 5: Kirim beban kerja Spark di Amazon EMR menggunakan gambar khusus

Setelah gambar kustom dibangun dan diterbitkan, Anda dapat mengirimkan tugas Amazon EMR di EKS menggunakan gambar kustom.

Pertama, buat start-job-run-request file.json dan tentukan `spark.kubernetes.container.image` parameter untuk referensi gambar kustom, seperti contoh file JSON berikut menunjukkan.

Note

Anda dapat menggunakan `local://` skema untuk merujuk ke file yang tersedia dalam gambar kustom seperti yang ditunjukkan dengan `entryPoint` argumen dalam cuplikan JSON di bawah ini. Anda juga dapat menggunakan `local://` skema untuk merujuk ke dependensi aplikasi. Semua file dan dependensi yang dirujuk menggunakan `local://` skema harus sudah ada di jalur yang ditentukan dalam gambar kustom.

```
{
  "name": "spark-custom-image",
  "virtualClusterId": "virtual-cluster-id",
  "executionRoleArn": "execution-role-arn",
  "releaseLabel": "emr-6.6.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "local:///usr/lib/spark/examples/jars/spark-examples.jar",
      "entryPointArguments": [
        "10"
      ],
      "sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi --conf spark.kubernetes.container.image=123456789012.dkr.ecr.us-west-2.amazonaws.com/emr6.6_custom_repo"
    }
  }
}
```

Anda juga dapat mereferensikan gambar kustom dengan `applicationConfiguration` properti seperti contoh berikut menunjukkan.

```
{
  "name": "spark-custom-image",
  "virtualClusterId": "virtual-cluster-id",
  "executionRoleArn": "execution-role-arn",
  "releaseLabel": "emr-6.6.0-latest",
```

```
"jobDriver": {
  "sparkSubmitJobDriver": {
    "entryPoint": "local:///usr/lib/spark/examples/jars/spark-examples.jar",
    "entryPointArguments": [
      "10"
    ],
    "sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi"
  }
},
"configurationOverrides": {
  "applicationConfiguration": [
    {
      "classification": "spark-defaults",
      "properties": {
        "spark.kubernetes.container.image": "123456789012.dkr.ecr.us-west-2.amazonaws.com/emr6.6_custom_repo"
      }
    }
  ]
}
}
```

Kemudian jalankan perintah `start-job-run` untuk mengirimkan tugas.

```
aws emr-containers start-job-run --cli-input-json file:///./start-job-run-request.json
```

Dalam contoh JSON di atas, ganti *emr-6.6.0-latest* dengan versi rilis Amazon EMR Anda. Kami sangat menyarankan Anda menggunakan versi `-latest` rilis untuk memastikan bahwa versi yang dipilih berisi pembaruan keamanan terbaru. Untuk informasi selengkapnya tentang versi rilis Amazon EMR dan tag gambarnya, lihat [Detail untuk memilih URI gambar dasar](#)

Note

Anda dapat menggunakan `spark.kubernetes.driver.container.image` dan `spark.kubernetes.executor.container.image` untuk menentukan gambar yang berbeda untuk driver dan pod eksekutor.

Sesuaikan gambar Docker untuk titik akhir interaktif

Anda juga dapat menyesuaikan gambar Docker untuk titik akhir interaktif sehingga Anda dapat menjalankan gambar kernel dasar yang disesuaikan. Ini membantu Anda memastikan bahwa Anda memiliki dependensi yang Anda butuhkan saat menjalankan beban kerja interaktif dari EMR Studio.

1. Ikuti [Langkah 1-4](#) yang diuraikan di atas untuk menyesuaikan gambar Docker. Untuk rilis Amazon EMR 6.9.0 dan yang lebih baru, Anda bisa mendapatkan URI gambar dasar dari Galeri Publik Amazon ECR. Untuk rilis sebelum Amazon EMR 6.9.0, Anda bisa mendapatkan gambar di akun Amazon ECR Registry di masing-masing akun Wilayah AWS, dan satu-satunya perbedaan adalah URI gambar dasar di Dockerfile Anda. URI gambar dasar mengikuti format:

```
ECR-registry-account.dkr.ecr.Region.amazonaws.com/notebook-spark/container-image-tag
```

Anda perlu menggunakan `notebook-spark` URI gambar dasar, bukan `spark`. Gambar dasar berisi runtime Spark dan kernel notebook yang berjalan dengannya. Untuk informasi selengkapnya tentang memilih tag gambar Wilayah dan kontainer, lihat [Detail untuk memilih URI gambar dasar](#).

Note

Saat ini hanya penggantian gambar dasar yang didukung dan memperkenalkan kernel yang sama sekali baru dari jenis lain selain yang AWS disediakan gambar dasar tidak didukung.

2. Buat endpoint interaktif yang dapat digunakan dengan gambar kustom.

Pertama, buat file JSON yang disebut `custom-image-managed-endpoint.json` dengan konten berikut.

```
{
  "name": "endpoint-name",
  "virtualClusterId": "virtual-cluster-id",
  "type": "JUPYTER_ENTERPRISE_GATEWAY",
  "releaseLabel": "emr-6.6.0-latest",
  "executionRoleArn": "execution-role-arn",
  "certificateArn": "certificate-arn",
  "configurationOverrides": {
```

```
"applicationConfiguration": [
  {
    "classification": "jupyter-kernel-overrides",
    "configurations": [
      {
        "classification": "python3",
        "properties": {
          "container-image": "123456789012.dkr.ecr.us-
west-2.amazonaws.com/custom-notebook-python:latest"
        }
      },
      {
        "classification": "spark-python-kubernetes",
        "properties": {
          "container-image": "123456789012.dkr.ecr.us-
west-2.amazonaws.com/custom-notebook-spark:latest"
        }
      }
    ]
  }
]
```

Selanjutnya, buat endpoint interaktif menggunakan konfigurasi yang ditentukan dalam file JSON, seperti contoh berikut menunjukkan.

```
aws emr-containers create-managed-endpoint --cli-input-json custom-image-managed-
endpoint.json
```

Untuk informasi selengkapnya, lihat [Membuat titik akhir interaktif untuk kluster virtual Anda](#).

3. Connect ke endpoint interaktif melalui EMR Studio. Untuk informasi selengkapnya, lihat [Menghubungkan dari Studio](#).

Bekerja dengan gambar multi-arsitektur

Amazon EMR di EKS mendukung gambar kontainer multi-arsitektur untuk Amazon Elastic Container Registry (Amazon ECR) Registry ECR). Untuk informasi selengkapnya, lihat [Memperkenalkan gambar kontainer multi-arsitektur untuk Amazon ECR](#).

Amazon EMR pada gambar khusus EKS mendukung instans EC2 AWS berbasis Graviton dan instans EC2. non-Graviton-based Gambar berbasis Graviton disimpan dalam repositori gambar yang sama di Amazon ECR sebagai gambar. non-Graviton-based

Misalnya, untuk memeriksa daftar manifes Docker untuk gambar 6.6.0, jalankan perintah berikut.

```
docker manifest inspect 895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.6.0:latest
```

Berikut adalah outputnya. `arm64` arsitekturnya untuk contoh Graviton. `amd64` ini untuk contoh non-Graviton.

```
{
  "schemaVersion": 2,
  "mediaType": "application/vnd.docker.distribution.manifest.list.v2+json",
  "manifests": [
    {
      "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
      "size": 1805,
      "digest":
"xxx123:6b971cb47d11011ab3d45fff925e9442914b4977ae0f9fbcdf5cfa99a7593f0",
      "platform": {
        "architecture": "arm64",
        "os": "linux"
      }
    },
    {
      "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
      "size": 1805,
      "digest":
"xxx123:6f2375582c9c57fa9838c1d3a626f1b4fc281e287d2963a72dfe0bd81117e52f",
      "platform": {
        "architecture": "amd64",
        "os": "linux"
      }
    }
  ]
}
```

Ambil langkah-langkah berikut untuk membuat gambar multi-arsitektur:

1. Buat `Dockerfile` dengan konten berikut sehingga Anda dapat menarik `arm64` gambar.

```
FROM --platform=arm64 895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/
emr-6.6.0:latest
USER root

RUN pip3 install boto3 // install customizations here
USER hadoop:hadoop
```

- Ikuti petunjuk di [Memperkenalkan gambar wadah multi-arsitektur untuk Amazon ECR](#) untuk membuat gambar multi-arsitektur.

Note

Anda harus membuat arm64 gambar pada arm64 instance. Demikian pula, Anda harus membangun amd64 gambar pada amd64 instance.

Anda juga dapat membuat gambar multi-arsitektur tanpa membangun setiap jenis instance tertentu dengan perintah `Dockerbuildx`. Untuk informasi selengkapnya, lihat [Memanfaatkan dukungan arsitektur multi-CPU](#).

- Setelah Anda membangun gambar multi-arsitektur, Anda dapat mengirimkan pekerjaan dengan `spark.kubernetes.container.image` parameter yang sama dan mengarahkannya ke gambar. Dalam cluster heterogen dengan instance AWS berbasis Graviton dan non-Graviton-based EC2, instance menentukan gambar arsitektur yang benar berdasarkan arsitektur instance yang menarik gambar.

Detail untuk memilih URI gambar dasar

Note

Untuk rilis Amazon EMR 6.9.0 dan yang lebih baru, Anda dapat mengambil gambar dasar dari Galeri Publik Amazon ECR, sehingga Anda tidak perlu membuat URI gambar dasar sebagai petunjuk langsung pada halaman ini. Untuk menemukan tag gambar kontainer untuk gambar dasar Anda, lihat [halaman catatan rilis](#) untuk rilis Amazon EMR yang sesuai di EKS.

Gambar Docker dasar yang dapat Anda pilih disimpan di Amazon Elastic Container Registry (Amazon ECR). URI gambar mengikuti format ini: *ECR-registry-*

`account.dkr.ecr.Region.amazonaws.com/spark/container-image-tag`, seperti contoh berikut menunjukkan.

```
895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-7.13.0:latest
```

URI gambar untuk endpoint interaktif mengikuti format ini: `ECR-registry-account.dkr.ecr.Region.amazonaws.com/notebook-spark/container-image-tag`, seperti contoh berikut menunjukkan. Anda perlu menggunakan `notebook-spark` URI gambar dasar, bukan `spark`.

```
895885662937.dkr.ecr.us-west-2.amazonaws.com/notebook-spark/emr-7.13.0:latest
```

Demikian pula, untuk `python3` gambar non-Spark untuk titik akhir interaktif, URI gambar adalah `ECR-registry-account.dkr.ecr.Region.amazonaws.com/notebook-python/container-image-tag` Contoh URI berikut diformat dengan benar:

```
895885662937.dkr.ecr.us-west-2.amazonaws.com/notebook-python/emr-7.13.0:latest
```

Untuk menemukan tag gambar kontainer untuk gambar dasar Anda, lihat [halaman catatan rilis](#) untuk rilis Amazon EMR yang sesuai di EKS.

Akun registri Amazon ECR berdasarkan Wilayah

Untuk menghindari latensi jaringan yang tinggi, tarik gambar dasar dari yang terdekat Wilayah AWS. Pilih akun registri Amazon ECR yang sesuai dengan Wilayah tempat Anda menarik gambar berdasarkan tabel berikut.

Wilayah	Akun registri Amazon ECR
ap-east-1	736135916053
ap-northeast-1	059004520145
ap-northeast-2	996579266876
ap-northeast-3	705689932349
ap-southeast-3	946962994502

Wilayah	Akun registri Amazon ECR	
ap-south-1	235914868574	
ap-south-2	691480105545	
ap-southeast-1	671219180197	
ap-southeast-2	038297999601	
ca-central-1	351826393999	
eu-central-1	107292555468	
eu-central-2	314408114945	
eu-north-1	830386416364	
eu-west-1	483788554619	
eu-west-2	118780647275	
eu-west-3	307523725174	
eu-south-1	238014973495	
eu-south-2	350796622945	
il-central-1	395734710648	
me-south-1	008085056818	
me-central-1	818935616732	
sa-east-1	052806832358	
us-gov-west-1	299385240661	
us-gov-east-1	299393998622	
us-east-1	755674844232	

Wilayah	Akun registri Amazon ECR
us-east-2	711395599931
us-west-1	608033475327
us-west-2	895885662937
af-south-1	358491847878
cn-north-1	068337069695
cn-northwest-1	068420816659

Pertimbangan untuk menyesuaikan gambar

Saat Anda menyesuaikan gambar Docker, Anda dapat memilih runtime yang tepat untuk pekerjaan Anda pada tingkat granular. Pertimbangkan praktik terbaik ini saat Anda menggunakan fitur ini. Ini termasuk pertimbangan untuk keamanan, konfigurasi, dan pemasangan gambar:

- Keamanan adalah tanggung jawab bersama antara Anda AWS dan Anda. Anda bertanggung jawab atas keamanan menambal binari yang Anda tambahkan ke gambar. Ikuti [Amazon EMR pada praktik terbaik keamanan EKS](#), terutama [Dapatkan pembaruan keamanan terbaru untuk gambar kustom](#) dan [Terapkan prinsip hak istimewa paling rendah](#).
- Saat Anda menyesuaikan gambar dasar, Anda harus mengubah pengguna Docker `hadoop:hadoop` agar pekerjaan tidak berjalan dengan pengguna `root`.
- Amazon EMR di EKS memasang file di atas konfigurasi untuk gambar, seperti, saat dijalankan `spark-defaults.conf`. Untuk mengganti file konfigurasi ini, kami sarankan Anda menggunakan `applicationOverrides` parameter selama pengiriman pekerjaan dan tidak secara langsung memodifikasi file dalam gambar kustom.
- Amazon EMR di EKS memasang folder tertentu saat dijalankan. Modifikasi apa pun yang Anda buat pada folder ini tidak tersedia di wadah. Jika Anda ingin menambahkan aplikasi atau dependensinya untuk gambar kustom, kami sarankan Anda memilih direktori yang bukan bagian dari jalur yang telah ditentukan berikut:
 - `/var/log/fluentd`
 - `/var/log/spark/user`

- `/var/log/spark/apps`
 - `/mnt`
 - `/tmp`
 - `/home/hadoop`
- Anda dapat mengunggah gambar yang disesuaikan ke repositori yang kompatibel dengan Docker, seperti Amazon ECR, Docker Hub, atau repositori perusahaan swasta. Untuk informasi selengkapnya tentang cara mengonfigurasi otentikasi kluster Amazon EKS dengan repositori Docker yang dipilih, lihat [Tarik Gambar dari Registri](#) Pribadi.

Menjalankan pekerjaan Flink dengan Amazon EMR di EKS

Amazon EMR merilis 6.13.0 dan lebih tinggi mendukung Amazon EMR di EKS dengan Apache Flink, atau operator Flink Kubernetes, sebagai model pengiriman pekerjaan untuk Amazon EMR di EKS. Dengan Amazon EMR di EKS dengan Apache Flink, Anda dapat menerapkan dan mengelola aplikasi Flink dengan runtime rilis Amazon EMR di kluster Amazon EKS Anda sendiri. Setelah Anda menerapkan operator Flink Kubernetes di cluster Amazon EKS Anda, Anda dapat langsung mengirimkan aplikasi Flink dengan operator. Operator mengelola siklus hidup aplikasi Flink.

Topik

- [Menyiapkan dan menggunakan operator Flink Kubernetes](#)
- [Menggunakan Flink Native Kubernetes](#)
- [Menyesuaikan gambar Docker untuk Flink dan FluentD](#)
- [Memantau pekerjaan operator Flink Kubernetes dan Flink](#)
- [Bagaimana Flink mendukung ketersediaan tinggi dan ketahanan kerja](#)
- [Menggunakan Autoscaler untuk aplikasi Flink](#)
- [Pemeliharaan dan pemecahan masalah untuk pekerjaan Flink di Amazon EMR di EKS](#)
- [Rilis yang didukung untuk Amazon EMR di EKS dengan Apache Flink](#)

Menyiapkan dan menggunakan operator Flink Kubernetes

Halaman-halaman berikut menjelaskan cara mengatur dan menggunakan operator Flink Kubernetes untuk menjalankan pekerjaan Flink dengan Amazon EMR di EKS. Topik yang tersedia mencakup prasyarat yang diperlukan, cara mengatur lingkungan Anda, dan menjalankan aplikasi Flink di Amazon EMR di EKS.

Topik

- [Menyiapkan operator Flink Kubernetes untuk Amazon EMR di EKS](#)
- [Menginstal operator Flink Kubernetes untuk Amazon EMR di EKS](#)
- [Jalankan aplikasi Flink](#)
- [Izin peran keamanan untuk menjalankan aplikasi Flink](#)
- [Menghapus instalasi operator Flink Kubernetes untuk Amazon EMR di EKS](#)

Menyiapkan operator Flink Kubernetes untuk Amazon EMR di EKS

Selesaikan tugas-tugas berikut untuk menyiapkan sebelum Anda menginstal operator Flink Kubernetes di Amazon EKS. Jika Anda sudah mendaftar untuk Amazon Web Services (AWS) dan telah menggunakan Amazon EKS, Anda hampir siap untuk menggunakan Amazon EMR di EKS. Selesaikan tugas-tugas berikut untuk menyiapkan operator Flink di Amazon EKS. Jika Anda telah menyelesaikan salah satu prasyarat, Anda dapat melewatinya dan melanjutkan ke yang berikutnya.

- [Instal atau perbarui ke versi terbaru AWS CLI](#) - Jika Anda sudah menginstal AWS CLI, konfirmasi bahwa Anda memiliki versi terbaru.
- [Siapkan kubectl dan eksctl - eksctl](#) adalah alat baris perintah yang Anda gunakan untuk berkomunikasi dengan Amazon EKS.
- [Instal Helm](#) — Manajer paket Helm untuk Kubernetes membantu Anda menginstal dan mengelola aplikasi di kluster Kubernetes Anda.
- [Memulai Amazon EKS — eksctl](#) — Ikuti langkah-langkah untuk membuat cluster Kubernetes baru dengan node di Amazon EKS.
- [Pilih label rilis EMR Amazon \(rilis 6.13.0 atau lebih tinggi\)](#) — operator Flink Kubernetes didukung dengan Amazon EMR rilis 6.13.0 dan yang lebih tinggi.
- [Aktifkan Peran IAM untuk Akun Layanan \(IRSA\) di kluster Amazon EKS.](#)
- [Buat peran eksekusi pekerjaan.](#)
- [Perbarui kebijakan kepercayaan dari peran eksekusi pekerjaan.](#)
- Buat peran eksekusi operator. Langkah ini bersifat opsional. Anda dapat menggunakan peran yang sama untuk pekerjaan dan operator Flink. Jika Anda ingin memiliki peran IAM yang berbeda untuk operator Anda, Anda dapat membuat peran terpisah.
- Perbarui kebijakan kepercayaan dari peran eksekusi operator. Anda harus secara eksplisit menambahkan satu entri kebijakan kepercayaan untuk peran yang ingin Anda gunakan untuk akun layanan operator Amazon EMR Flink Kubernetes. Anda dapat mengikuti format contoh ini:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::AWS_ACCOUNT_ID:oidc-provider/OIDC_PROVIDER"
      }
    }
  ]
}
```

```

    },
    "Action": "sts:AssumeRoleWithWebIdentity",
    "Condition": {
      "StringEquals": {
        "OIDC_PROVIDER:aud": "sts.amazonaws.com",
        "OIDC_PROVIDER:sub": "system:serviceaccount:emr:emr-containers-sa-
flink-operator"
      }
    }
  ]
}

```

Menginstal operator Flink Kubernetes untuk Amazon EMR di EKS

Topik ini membantu Anda mulai menggunakan operator Flink Kubernetes di Amazon EKS dengan menyiapkan penerapan Flink.

Instal operator Kubernetes

Gunakan langkah-langkah berikut untuk menginstal operator Kubernetes untuk Apache Flink.

1. Jika Anda belum melakukannya, selesaikan langkah-langkahnya [the section called “Menyiapkan”](#).
2. Instal *cert-manager* (sekali per kluster Amazon EKS) untuk mengaktifkan penambahan komponen webhook.

```
kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.12.0/cert-manager.yaml
```

3. Instal bagan Helm.

```

export VERSION=7.13.0 # The Amazon EMR release version
export NAMESPACE=The Kubernetes namespace to deploy the operator

helm install flink-kubernetes-operator \
oci://public.ecr.aws/emr-on-eks/flink-kubernetes-operator \
--version $VERSION \
--namespace $NAMESPACE

```

Contoh output:

```
NAME: flink-kubernetes-operator
LAST DEPLOYED: Tue May 31 17:38:56 2022
NAMESPACE: $NAMESPACE
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

4. Tunggu hingga penerapan selesai dan verifikasi instalasi bagan.

```
kubectl wait deployment flink-kubernetes-operator --namespace $NAMESPACE --for
condition=Available=True --timeout=30s
```

5. Anda akan melihat pesan berikut saat penerapan selesai.

```
deployment.apps/flink-kubernetes-operator condition met
```

6. Gunakan perintah berikut untuk melihat operator yang dikerahkan.

```
helm list --namespace $NAMESPACE
```

Berikut ini menunjukkan contoh keluaran, di mana versi aplikasi `x.y.z-amzn-n` akan sesuai dengan versi operator Flink untuk EMR Amazon Anda pada rilis EKS. Untuk informasi selengkapnya, lihat [Rilis yang didukung untuk Amazon EMR di EKS dengan Apache Flink](#).

NAME	STATUS	CHART	NAMESPACE	REVISION	UPDATED	APP VERSION
flink-kubernetes-operator-0500	EST	deployed	\$NAMESPACE	1	2023-02-22 16:43:45	24148
			flink-kubernetes-operator-emr-7.13.0			x.y.z-amzn-n

Tingkatkan operator Kubernetes

Untuk memutakhirkan versi operator Flink, ikuti langkah-langkah ini:

1. Copot pemasangan yang lama `flink-kubernetes-operator:helm uninstall flink-kubernetes-operator -n <NAMESPACE>`.
2. Hapus CRD (karena helm tidak akan secara otomatis menghapus CRD lama): `kubectl delete crd flinkdeployments.flink.apache.org flinksessionjobs.flink.apache.org`

3. Instal ulang `flink-kubernetes-operator` dengan versi yang lebih baru.

Jalankan aplikasi Flink

Dengan Amazon EMR 6.13.0 dan yang lebih tinggi, Anda dapat menjalankan aplikasi Flink dengan operator Flink Kubernetes dalam mode Aplikasi di Amazon EMR di EKS. Dengan Amazon EMR 6.15.0 dan yang lebih tinggi, Anda juga dapat menjalankan aplikasi Flink dalam mode Sesi. Halaman ini menjelaskan kedua metode yang dapat Anda gunakan untuk menjalankan aplikasi Flink dengan Amazon EMR di EKS.

Note

Anda harus memiliki bucket Amazon S3 yang dibuat untuk menyimpan metadata ketersediaan tinggi saat mengirimkan pekerjaan Flink Anda. Jika Anda tidak ingin menggunakan fitur ini, Anda dapat menonaktifkannya. Ini diaktifkan secara default.

Prasyarat — Sebelum Anda dapat menjalankan aplikasi Flink dengan operator Flink Kubernetes, selesaikan langkah-langkah di dan [the section called “Menyiapkan”](#) [the section called “Instal operator Kubernetes”](#)

Application mode

Dengan Amazon EMR 6.13.0 dan yang lebih tinggi, Anda dapat menjalankan aplikasi Flink dengan operator Flink Kubernetes dalam mode Aplikasi di Amazon EMR di EKS.

1. Buat file FlinkDeployment definisi `basic-example-app-cluster.yaml` seperti pada contoh berikut. Jika Anda mengaktifkan dan menggunakan salah satu [opt-in Wilayah AWS](#), pastikan Anda membatalkan komentar dan mengonfigurasi konfigurasi `fs.s3a.endpoint.region`

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: basic-example-app-cluster
spec:
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
    #fs.s3a.endpoint.region: OPT_IN_AWS_REGION_NAME
```

```
state.checkpoints.dir: CHECKPOINT_S3_STORAGE_PATH
state.savepoints.dir: SAVEPOINT_S3_STORAGE_PATH
flinkVersion: v1_17
executionRoleArn: JOB_EXECUTION_ROLE_ARN
emrReleaseLabel: "emr-6.13.0-flink-latest" # 6.13 or higher
jobManager:
  storageDir: HIGH_AVAILABILITY_STORAGE_PATH
  resource:
    memory: "2048m"
    cpu: 1
taskManager:
  resource:
    memory: "2048m"
    cpu: 1
job:
  # if you have your job jar in S3 bucket you can use that path as well
  jarURI: local:///opt/flink/examples/streaming/StateMachineExample.jar
  parallelism: 2
  upgradeMode: savepoint
  savepointTriggerNonce: 0
monitoringConfiguration:
  cloudWatchMonitoringConfiguration:
    logGroupName: LOG_GROUP_NAME
```

2. Kirim penyebaran Flink dengan perintah berikut. Ini juga akan membuat FlinkDeployment objek bernamabasic-example-app-cluster.

```
kubectl create -f basic-example-app-cluster.yaml -n <NAMESPACE>
```

3. Akses UI Flink.

```
kubectl port-forward deployments/basic-example-app-cluster 8081 -n NAMESPACE
```

4. Buka localhost:8081 untuk melihat pekerjaan Flink Anda secara lokal.
5. Bersihkan pekerjaan. Ingatlah untuk membersihkan artefak S3 yang dibuat untuk pekerjaan ini, seperti checkpointing, ketersediaan tinggi, metadata savepointing, dan log. CloudWatch

Untuk informasi selengkapnya tentang mengirimkan aplikasi ke Flink melalui operator Flink Kubernetes, lihat contoh operator Flink Kubernetes di [folder](#) di [apache/flink-kubernetes-operator](#) GitHub

Session mode

Dengan Amazon EMR 6.15.0 dan yang lebih tinggi, Anda dapat menjalankan aplikasi Flink dengan operator Flink Kubernetes dalam mode Session di Amazon EMR di EKS.

1. Buat file FlinkDeployment definisi bernama `basic-example-app-cluster.yaml` seperti pada contoh berikut. Jika Anda mengaktifkan dan menggunakan salah satu [opt-in Wilayah AWS](#), pastikan Anda membatalkan komentar dan mengonfigurasi konfigurasi `fs.s3a.endpoint.region`

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: basic-example-session-cluster
spec:
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
    #fs.s3a.endpoint.region: OPT_IN_AWS_REGION_NAME
    state.checkpoints.dir: CHECKPOINT_S3_STORAGE_PATH
    state.savepoints.dir: SAVEPOINT_S3_STORAGE_PATH
  flinkVersion: v1_17
  executionRoleArn: JOB_EXECUTION_ROLE_ARN
  emrReleaseLabel: "emr-6.15.0-flink-latest"
  jobManager:
    storageDir: HIGH_AVAILABILITY_S3_STORAGE_PATH
  resource:
    memory: "2048m"
    cpu: 1
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
  monitoringConfiguration:
    s3MonitoringConfiguration:
      logUri:
    cloudWatchMonitoringConfiguration:
      logGroupName: LOG_GROUP_NAME
```

2. Kirim penyebaran Flink dengan perintah berikut. Ini juga akan membuat FlinkDeployment objek bernama `basic-example-session-cluster`.

```
kubectl create -f basic-example-app-cluster.yaml -n NAMESPACE
```

- Gunakan perintah berikut untuk mengonfirmasi bahwa cluster sesi LIFECYCLE adalah STABLE:

```
kubectl get flinkdeployments.flink.apache.org basic-example-session-cluster -n NAMESPACE
```

Outputnya harus mirip dengan contoh berikut:

NAME	JOB STATUS	LIFECYCLE STATE
basic-example-session-cluster		STABLE

- Buat file sumber daya definisi FlinkSessionJob khusus basic-session-job.yaml dengan contoh konten berikut:

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkSessionJob
metadata:
  name: basic-session-job
spec:
  deploymentName: basic-session-deployment
  job:
    # If you have your job jar in an S3 bucket you can use that path.
    # To use jar in S3 bucket, set
    # OPERATOR_EXECUTION_ROLE_ARN (--set emrContainers.operatorExecutionRoleArn=OPERATOR_EXECUTION_ROLE_ARN)
    # when you install Spark operator
    jarURI: https://repo1.maven.org/maven2/org/apache/flink/flink-examples-streaming_2.12/1.16.1/flink-examples-streaming_2.12-1.16.1-TopSpeedWindowing.jar
    parallelism: 2
    upgradeMode: stateless
```

- Kirim pekerjaan sesi Flink dengan perintah berikut. Ini akan membuat FlinkSessionJob objek basic-session-job.

```
kubectl apply -f basic-session-job.yaml -n NAMESPACE
```

- Gunakan perintah berikut untuk mengonfirmasi bahwa cluster sesi LIFECYCLE adalah STABLE, dan JOB STATUS adalah RUNNING:

```
kubectl get flinkdeployments.flink.apache.org basic-example-session-cluster -n NAMESPACE
```

Outputnya harus mirip dengan contoh berikut:

NAME	JOB STATUS	LIFECYCLE STATE
basic-example-session-cluster	RUNNING	STABLE

7. Akses UI Flink.

```
kubectl port-forward deployments/basic-example-session-cluster 8081 -n NAMESPACE
```

- Buka `localhost:8081` untuk melihat pekerjaan Flink Anda secara lokal.
- Bersihkan pekerjaan. Ingatlah untuk membersihkan artefak S3 yang dibuat untuk pekerjaan ini, seperti checkpointing, ketersediaan tinggi, metadata savepointing, dan log. CloudWatch

Izin peran keamanan untuk menjalankan aplikasi Flink

Topik ini menjelaskan peran keamanan untuk menerapkan dan menjalankan aplikasi Flink. Ada dua peran yang diperlukan untuk mengelola penyebaran dan untuk membuat dan mengelola pekerjaan, peran operator dan peran pekerjaan. Topik ini memperkenalkan mereka dan mencantumkan izin mereka.

Kontrol akses berbasis peran

Untuk menerapkan operator dan menjalankan pekerjaan Flink, kita harus membuat dua peran Kubernetes: satu operator dan satu peran pekerjaan. Amazon EMR membuat dua peran secara default saat Anda menginstal operator.

Peran operator

Kami menggunakan peran operator untuk mengelola `flinkdeployments` untuk membuat dan mengelola JobManager untuk setiap pekerjaan Flink dan sumber daya lainnya, seperti layanan.

Nama default peran operator adalah `emr-containers-sa-flink-operator` dan memerlukan izin berikut.

```
rules:
- apiGroups:
  - ""
  resources:
  - pods
  - services
```

```
- events
- configmaps
- secrets
- serviceaccounts
verbs:
- '*'
- apiGroups:
  - rbac.authorization.k8s.io
resources:
- roles
- rolebindings
verbs:
- '*'
- apiGroups:
  - apps
resources:
- deployments
- deployments/finalizers
- replicaset
verbs:
- '*'
- apiGroups:
  - extensions
resources:
- deployments
- ingresses
verbs:
- '*'
- apiGroups:
  - flink.apache.org
resources:
- flinkdeployments
- flinkdeployments/status
- flinksessionjobs
- flinksessionjobs/status
verbs:
- '*'
- apiGroups:
  - networking.k8s.io
resources:
- ingresses
verbs:
- '*'
- apiGroups:
```

```
- coordination.k8s.io
resources:
- leases
verbs:
- '*'
```

Peran Job

JobManager Menggunakan peran pekerjaan untuk membuat dan mengelola TaskManagers dan ConfigMaps untuk setiap pekerjaan.

```
rules:
- apiGroups:
  - ""
  resources:
  - pods
  - configmaps
  verbs:
  - '*'
- apiGroups:
  - apps
  resources:
  - deployments
  - deployments/finalizers
  verbs:
  - '*'
```

Menghapus instalasi operator Flink Kubernetes untuk Amazon EMR di EKS

Ikuti langkah-langkah berikut untuk menghapus instalasi operator Flink Kubernetes.

1. Hapus operator.

```
helm uninstall flink-kubernetes-operator -n <NAMESPACE>
```

2. Hapus sumber daya Kubernetes yang tidak dihapus Helm.

```
kubectl delete serviceaccounts, roles, rolebindings -l emr-
containers.amazonaws.com/component=flink.operator --namespace <namespace>
kubectl delete crd flinkdeployments.flink.apache.org
flinksessionjobs.flink.apache.org
```

3. (Opsional) Hapus cert-manager.

```
kubectl delete -f https://github.com/jetstack/cert-manager/releases/download/v1.12.0/cert-manager.yaml
```

Menggunakan Flink Native Kubernetes

Amazon EMR merilis 6.13.0 dan dukungan yang lebih tinggi Flink Native Kubernetes sebagai alat baris perintah yang dapat Anda gunakan untuk mengirimkan dan menjalankan aplikasi Flink ke EMR Amazon di kluster EKS.

Topik

- [Menyiapkan Flink Native Kubernetes untuk Amazon EMR di EKS](#)
- [Memulai dengan Kubernetes asli Flink untuk Amazon EMR di EKS](#)
- [Persyaratan keamanan akun JobManager layanan Flink untuk Native Kubernetes](#)

Menyiapkan Flink Native Kubernetes untuk Amazon EMR di EKS

Selesaikan tugas-tugas berikut untuk menyiapkan sebelum Anda dapat menjalankan aplikasi dengan Flink CLI di Amazon EMR di EKS. Jika Anda sudah mendaftar untuk Amazon Web Services (AWS) dan telah menggunakan Amazon EKS, Anda hampir siap untuk menggunakan Amazon EMR di EKS. Jika Anda telah menyelesaikan salah satu prasyarat, Anda dapat melewatinya dan melanjutkan ke yang berikutnya.

- [Instal atau perbarui ke versi terbaru AWS CLI](#) - Jika Anda sudah menginstal AWS CLI, konfirmasi bahwa Anda memiliki versi terbaru.
- [Memulai Amazon EKS — eksctl](#) — Ikuti langkah-langkah untuk membuat cluster Kubernetes baru dengan node di Amazon EKS.
- [Pilih URI image dasar EMR Amazon](#) (rilis 6.13.0 atau lebih tinggi) — perintah Flink Kubernetes didukung dengan Amazon EMR rilis 6.13.0 dan yang lebih tinggi.
- Konfirmasi bahwa akun JobManager layanan memiliki izin yang sesuai untuk membuat dan menonton TaskManager pod. Untuk informasi selengkapnya, lihat [persyaratan keamanan akun JobManager layanan Flink untuk Kubernetes Asli](#).
- Siapkan profil [AWS kredensial](#) lokal Anda.

- [Buat atau perbarui file kubeconfig untuk cluster Amazon EKS](#) tempat Anda ingin menjalankan aplikasi Flink.

Memulai dengan Kubernetes asli Flink untuk Amazon EMR di EKS

Langkah-langkah ini menunjukkan cara mengonfigurasi, menyiapkan akun layanan, dan menjalankan aplikasi Flink. Flink Native Kubernetes digunakan untuk menyebarkan Flink pada kluster Kubernetes yang sedang berjalan.

Konfigurasi dan jalankan aplikasi Flink

Amazon EMR 6.13.0 dan yang lebih tinggi mendukung Flink Native Kubernetes untuk menjalankan aplikasi Flink di cluster Amazon EKS. Untuk menjalankan aplikasi Flink, ikuti langkah-langkah berikut:

1. Sebelum Anda dapat menjalankan aplikasi Flink dengan perintah Flink Native Kubernetes, selesaikan langkah-langkahnya. [the section called “Menyiapkan”](#)
2. [Unduh dan instal Flink.](#)
3. Tetapkan nilai untuk variabel lingkungan berikut.

```
#Export the FLINK_HOME environment variable to your local installation of Flink
export FLINK_HOME=/usr/local/bin/flink #Will vary depending on your installation
export NAMESPACE=flink
export CLUSTER_ID=flink-application-cluster
export IMAGE=<123456789012.dkr.ecr.sample-wilayah AWS-.amazonaws.com/flink/
emr-6.13.0-flink:latest>
export FLINK_SERVICE_ACCOUNT=emr-containers-sa-flink
export FLINK_CLUSTER_ROLE_BINDING=emr-containers-crb-flink
```

4. Buat akun layanan untuk mengelola sumber daya Kubernetes.

```
kubectl create serviceaccount $FLINK_SERVICE_ACCOUNT -n $NAMESPACE
kubectl create clusterrolebinding $FLINK_CLUSTER_ROLE_BINDING --clusterrole=edit --
serviceaccount=$NAMESPACE:$FLINK_SERVICE_ACCOUNT
```

5. Jalankan run-application perintah CLI.

```
$FLINK_HOME/bin/flink run-application \  
  --target kubernetes-application \  
  -Dkubernetes.namespace=$NAMESPACE \  
  -Dkubernetes.cluster-id=$CLUSTER_ID \  
  \
```

```

-Dkubernetes.container.image.ref=$IMAGE \
-Dkubernetes.service-account=$FLINK_SERVICE_ACCOUNT \
local:///opt/flink/examples/streaming/Iteration.jar
2022-12-29 21:13:06,947 INFO org.apache.flink.kubernetes.utils.KubernetesUtils
    [] - Kubernetes deployment requires a fixed port. Configuration
blob.server.port will be set to 6124
2022-12-29 21:13:06,948 INFO org.apache.flink.kubernetes.utils.KubernetesUtils
    [] - Kubernetes deployment requires a fixed port. Configuration
taskmanager.rpc.port will be set to 6122
2022-12-29 21:13:07,861 WARN
org.apache.flink.kubernetes.KubernetesClusterDescriptor    [] - Please note that
Flink client operations(e.g. cancel, list, stop, savepoint, etc.) won't work from
outside the Kubernetes cluster since 'kubernetes.rest-service.exposed.type' has
been set to ClusterIP.
2022-12-29 21:13:07,868 INFO
org.apache.flink.kubernetes.KubernetesClusterDescriptor    [] - Create flink
application cluster flink-application-cluster successfully, JobManager Web
Interface: http://flink-application-cluster-rest.flink:8081

```

6. Periksa sumber daya Kubernetes yang dibuat.

```

kubectl get all -n <namespace>
NAME READY STATUS RESTARTS AGE
pod/flink-application-cluster-546687cb47-w2p2z 1/1 Running 0 3m37s
pod/flink-application-cluster-taskmanager-1-1 1/1 Running 0 3m24s

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
service/flink-application-cluster ClusterIP None <none> 6123/TCP,6124/TCP 3m38s
service/flink-application-cluster-rest ClusterIP 10.100.132.158 <none> 8081/TCP
3m38s

NAME READY UP-TO-DATE AVAILABLE AGE
deployment.apps/flink-application-cluster 1/1 1 1 3m38s

NAME DESIRED CURRENT READY AGE
replicaset.apps/flink-application-cluster-546687cb47 1 1 1 3m38s

```

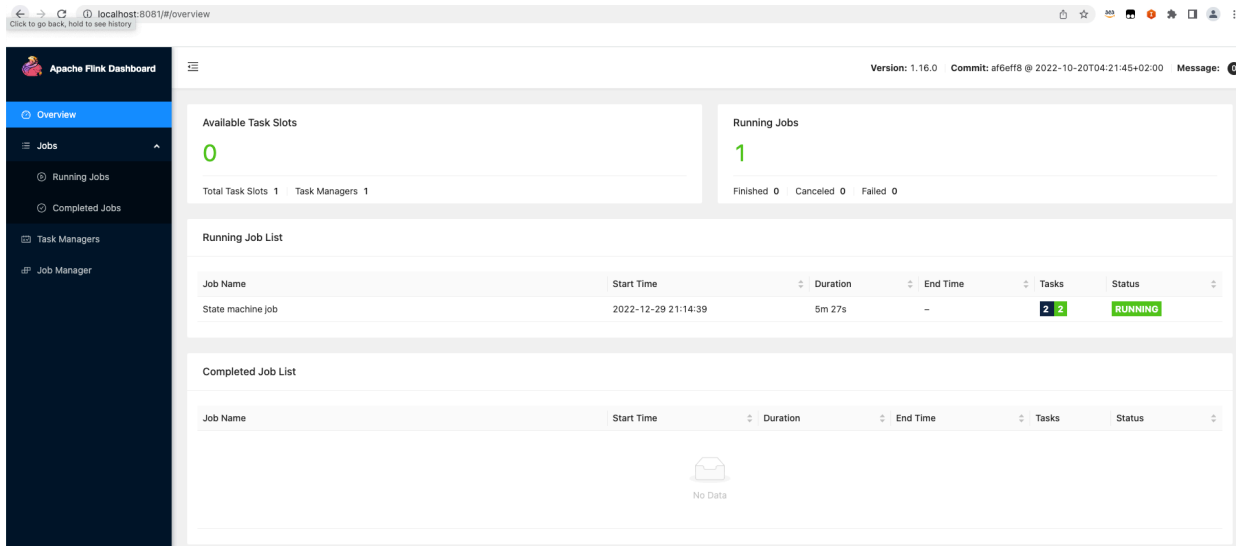
7. Port maju ke 8081.

```

kubectl port-forward service/flink-application-cluster-rest 8081 -n <namespace>
Forwarding from 127.0.0.1:8081 -> 8081

```

8. Akses UI Flink secara lokal.



9. Hapus aplikasi Flink.

```
kubectl delete deployment.apps/flink-application-cluster -n <namespace>
deployment.apps "flink-application-cluster" deleted
```

Untuk informasi selengkapnya tentang mengirimkan aplikasi ke Flink, lihat [Native Kubernetes](#) di dokumentasi Apache Flink.

Persyaratan keamanan akun JobManager layanan Flink untuk Native Kubernetes

JobManager Pod Flink menggunakan akun layanan Kubernetes untuk mengakses server API Kubernetes untuk membuat dan menonton pod. TaskManager Akun JobManager layanan harus memiliki izin yang sesuai untuk create/delete TaskManager pod dan memungkinkan pemimpin TaskManager to watch ConfigMaps untuk mengambil alamat JobManager dan ResourceManager di kluster Anda.

Aturan berikut berlaku untuk akun layanan ini.

```
rules:
- apiGroups:
  - ""
  resources:
  - pods
```

```
verbs:
- "*"
- apiGroups:
- ""
resources:
- services
verbs:
- "*"
- apiGroups:
- ""
resources:
- configmaps
verbs:
- "*"
- apiGroups:
- "apps"
resources:
- deployments
verbs:
- "*"

```

Menyesuaikan gambar Docker untuk Flink dan FluentD

Ambil langkah-langkah berikut untuk menyesuaikan gambar Docker untuk Amazon EMR di EKS dengan gambar Apache Flink atau FluentD. Ini termasuk panduan teknis untuk mendapatkan gambar dasar, menyesuaikannya, menerbitkannya, dan mengirimkan beban kerja.

Topik

- [Prasyarat](#)
- [Langkah 1: Ambil gambar dasar dari Amazon Elastic Container Registry](#)
- [Langkah 2: Sesuaikan gambar dasar](#)
- [Langkah 3: Publikasikan gambar kustom Anda](#)
- [Langkah 4: Kirim beban kerja Flink di Amazon EMR menggunakan gambar khusus](#)

Prasyarat

Sebelum Anda menyesuaikan image Docker Anda, pastikan bahwa Anda telah menyelesaikan prasyarat berikut:

- Menyelesaikan [Menyiapkan operator Flink Kubernetes untuk Amazon EMR](#) pada langkah-langkah EKS.
- Menginstal Docker di lingkungan Anda. Untuk informasi lebih lanjut, lihat [Get Docker](#).

Langkah 1: Ambil gambar dasar dari Amazon Elastic Container Registry

Gambar dasar berisi runtime Amazon EMR dan konektor yang perlu Anda akses lainnya. Layanan AWS Jika Anda menggunakan Amazon EMR di EKS dengan Flink versi 6.14.0 atau lebih tinggi, Anda bisa mendapatkan gambar dasar dari Galeri Publik Amazon ECR. Jelajahi galeri untuk menemukan tautan gambar dan tarik gambar ke ruang kerja lokal Anda. Misalnya, untuk rilis Amazon EMR 6.14.0, `docker pull` perintah berikut mengembalikan gambar dasar standar terbaru. Ganti `emr-6.14.0:latest` dengan versi rilis yang Anda inginkan.

```
docker pull public.ecr.aws/emr-on-eks/flink/emr-6.14.0-flink:latest
```

Berikut ini adalah tautan ke gambar galeri Flink dan gambar galeri Fluentd:

- [emr-on-eks/flink/emr-6.14.0-flink](#)
- [emr-on-eks/fasih/emr-6.14.0 \(](#)

Langkah 2: Sesuaikan gambar dasar

Langkah-langkah berikut menjelaskan cara menyesuaikan gambar dasar yang Anda tarik dari Amazon ECR.

1. Buat Dockerfile baru di Workspace lokal Anda.
2. Edit Dockerfile dan tambahkan konten berikut. Ini Dockerfile menggunakan gambar kontainer yang Anda tarik `public.ecr.aws/emr-on-eks/flink/emr-7.13.0-flink:latest`.

```
FROM public.ecr.aws/emr-on-eks/flink/emr-7.13.0-flink:latest
USER root
### Add customization commands here ####
USER hadoop:hadoop
```

Gunakan konfigurasi berikut jika Anda menggunakan Fluentd.

```
FROM public.ecr.aws/emr-on-eks/fluentd/emr-7.13.0:latest
USER root
### Add customization commands here ###
USER hadoop:hadoop
```

3. Tambahkan perintah di Dockerfile untuk menyesuaikan gambar dasar. Perintah berikut menunjukkan cara menginstal pustaka Python.

```
FROM public.ecr.aws/emr-on-eks/flink/emr-7.13.0-flink:latest
USER root
RUN pip3 install --upgrade boto3 pandas numpy // For python 3
USER hadoop:hadoop
```

4. Di direktori yang sama dengan tempat Anda membuat Dockerfile, jalankan perintah berikut untuk membangun image Docker. Bidang yang Anda berikan mengikuti -t bendera adalah nama kustom Anda untuk gambar.

```
docker build -t <YOUR_ACCOUNT_ID>.dkr.ecr.<YOUR_ECR_REGION>.amazonaws.com/
<ECR_REPO>:<ECR_TAG>
```

Langkah 3: Publikasikan gambar kustom Anda

Anda sekarang dapat mempublikasikan gambar Docker baru ke registri Amazon ECR Anda.

1. Jalankan perintah berikut untuk membuat repositori Amazon ECR untuk menyimpan image Docker Anda. Berikan nama untuk repositori Anda, seperti `emr_custom_repo`. Untuk informasi selengkapnya, lihat [Membuat repositori di Panduan Pengguna](#) Amazon Elastic Container Registry.

```
aws ecr create-repository \
  --repository-name emr_custom_repo \
  --image-scanning-configuration scanOnPush=true \
  --region <AWS_REGION>
```

2. Jalankan perintah berikut untuk mengautentikasi ke registri default Anda. Untuk informasi selengkapnya, lihat [Mengautentikasi ke registri default Anda](#) di Panduan Pengguna Amazon Elastic Container Registry.

```
aws ecr get-login-password --region <AWS_REGION> | docker login --username AWS --password-stdin <AWS_ACCOUNT_ID>.dkr.ecr.<YOUR_ECR_REGION>.amazonaws.com
```

3. Tekan gambar. Untuk informasi selengkapnya, lihat [Mendorong gambar ke Amazon ECR](#) di Panduan Pengguna Amazon Elastic Container Registry.

```
docker push <YOUR_ACCOUNT_ID>.dkr.ecr.<YOUR_ECR_REGION>.amazonaws.com/  
<ECR_REPO>:<ECR_TAG>
```

Langkah 4: Kirim beban kerja Flink di Amazon EMR menggunakan gambar khusus

Buat perubahan berikut pada FlinkDeployment spesifikasi Anda untuk menggunakan gambar kustom. Untuk melakukannya, masukkan gambar Anda sendiri di `spec.image` baris spesifikasi penerapan Anda.

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: basic-example
spec:
  flinkVersion: v1_18
  image: <YOUR_ACCOUNT_ID>.dkr.ecr.<YOUR_ECR_REGION>.amazonaws.com/  
<ECR_REPO>:<ECR_TAG>
  imagePullPolicy: Always
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "1"
```

Untuk menggunakan gambar kustom untuk pekerjaan Fluentd Anda, masukkan gambar Anda sendiri di `monitoringConfiguration.image` baris spesifikasi penerapan Anda.

```
monitoringConfiguration:
  image: <YOUR_ACCOUNT_ID>.dkr.ecr.<YOUR_ECR_REGION>.amazonaws.com/  
<ECR_REPO>:<ECR_TAG>
  cloudWatchMonitoringConfiguration:
    logGroupName: flink-log-group
    logStreamNamePrefix: custom-fluentd
```

Memantau pekerjaan operator Flink Kubernetes dan Flink

Bagian ini menjelaskan beberapa cara agar Anda dapat memantau pekerjaan Flink Anda dengan Amazon EMR di EKS. Ini termasuk mengintegrasikan Flink dengan Amazon Managed Service untuk Prometheus, menggunakan Dasbor Web Flink, yang menyediakan status pekerjaan dan metrik, atau menggunakan konfigurasi pemantauan untuk mengirim data log ke Amazon S3 dan Amazon CloudWatch

Topik

- [Menggunakan Amazon Managed Service untuk Prometheus untuk memantau pekerjaan Flink](#)
- [Gunakan UI Flink untuk memantau pekerjaan Flink](#)
- [Gunakan konfigurasi pemantauan untuk memantau pekerjaan operator Flink Kubernetes dan Flink](#)

Menggunakan Amazon Managed Service untuk Prometheus untuk memantau pekerjaan Flink

Anda dapat mengintegrasikan Apache Flink dengan Amazon Managed Service untuk Prometheus (portal manajemen). Layanan Terkelola Amazon untuk Prometheus mendukung pengambilan metrik dari Amazon Managed Service untuk server Prometheus dalam cluster yang berjalan di Amazon EKS. Layanan Terkelola Amazon untuk Prometheus bekerja sama dengan server Prometheus yang sudah berjalan di cluster Amazon EKS Anda. Menjalankan Amazon Managed Service untuk integrasi Prometheus dengan operator Amazon EMR Flink akan secara otomatis menerapkan dan mengonfigurasi server Prometheus untuk diintegrasikan dengan Amazon Managed Service untuk Prometheus.

1. [Buat Layanan Terkelola Amazon untuk Prometheus](#) Workspace. Ruang kerja ini berfungsi sebagai titik akhir konsumsi. Anda akan memerlukan URL tulis jarak jauh nanti.
2. Siapkan peran IAM untuk akun layanan.

Untuk metode orientasi ini, gunakan peran IAM untuk akun layanan di kluster Amazon EKS tempat server Prometheus berjalan. Peran ini juga disebut peran layanan.

Jika Anda belum memiliki peran, [siapkan peran layanan untuk menelan metrik dari kluster Amazon EKS](#).

Sebelum Anda melanjutkan, buat peran IAM yang disebut `amp-iamproxy-ingest-role`.

3. Instal Operator Flink EMR Amazon dengan Amazon Managed Service untuk Prometheus.

Sekarang setelah Anda memiliki Layanan Terkelola Amazon untuk ruang kerja Prometheus, peran IAM khusus untuk Layanan Terkelola Amazon untuk Prometheus, dan izin yang diperlukan, Anda dapat menginstal operator Amazon EMR Flink.

Buat file `enable-amp.yaml`. File ini memungkinkan Anda menggunakan konfigurasi khusus untuk mengganti Layanan Terkelola Amazon untuk pengaturan Prometheus. Pastikan untuk menggunakan peran Anda sendiri.

```
kube-prometheus-stack:
  prometheus:
    serviceAccount:
      create: true
      name: "amp-iamproxy-ingest-service-account"
      annotations:
        eks.amazonaws.com/role-arn: "arn:aws:iam::<AWS_ACCOUNT_ID>:role/amp-iamproxy-ingest-role"
    remoteWrite:
      - url: <AMAZON_MANAGED_PROMETHEUS_REMOTE_WRITE_URL>
    sigv4:
      region: <AWS_REGION>
    queueConfig:
      maxSamplesPerSend: 1000
      maxShards: 200
      capacity: 2500
```

Gunakan [Helm Install --set](#) perintah untuk meneruskan penggantian ke bagan. `flink-kubernetes-operator`

```
helm upgrade -n <namespace> flink-kubernetes-operator \
  oci://public.ecr.aws/emr-on-eks/flink-kubernetes-operator \
  --set prometheus.enabled=true
-f enable-amp.yaml
```

Perintah ini secara otomatis menginstal reporter Prometheus di operator pada port 9999. Any future `FlinkDeployment` juga mengekspos `metrics` port di 9249.

- Metrik operator Flink muncul di Prometheus di bawah label. `flink_k8soperator_`
- Metrik Flink Task Manager muncul di Prometheus di bawah label. `flink_taskmanager_`
- Metrik Manajer Job Flink muncul di Prometheus di bawah label. `flink_jobmanager_`

Gunakan UI Flink untuk memantau pekerjaan Flink

Untuk memantau kesehatan dan kinerja aplikasi Flink yang sedang berjalan, gunakan Flink Web Dashboard. Dasbor ini memberikan informasi tentang status pekerjaan, jumlah TaskManagers, dan metrik serta log untuk pekerjaan itu. Ini juga memungkinkan Anda melihat dan memodifikasi konfigurasi pekerjaan Flink, dan berinteraksi dengan cluster Flink untuk mengirimkan atau membatalkan pekerjaan.

Untuk mengakses Flink Web Dashboard untuk aplikasi Flink yang sedang berjalan di Kubernetes:

1. Gunakan `kubectl port-forward` perintah untuk meneruskan port lokal ke port tempat Flink Web Dashboard berjalan di pod aplikasi TaskManager Flink. Secara default, port ini adalah 8081. Ganti *deployment-name* dengan nama penerapan aplikasi Flink dari atas.

```
kubectl get deployments -n namespace
```

Contoh output:

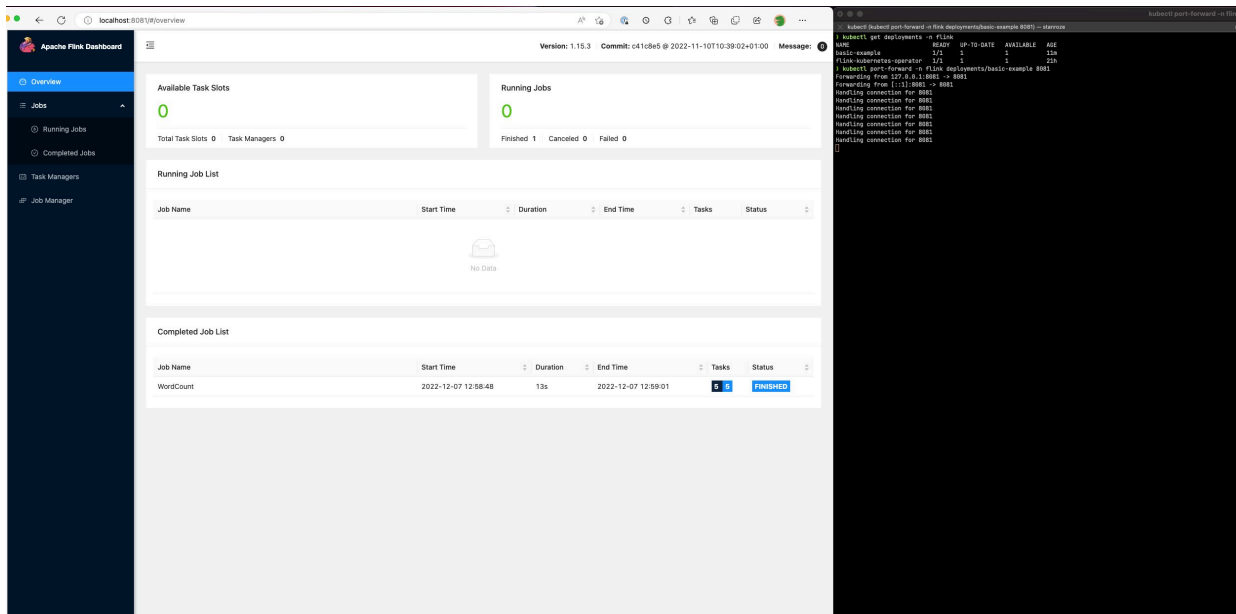
```
kubectl get deployments -n flink-namespace
NAME                                READY    UP-TO-DATE    AVAILABLE    AGE
basic-example                       1/1      1              1            11m
flink-kubernetes-operator           1/1      1              1            21h
```

```
kubectl port-forward deployments/deployment-name 8081 -n namespace
```

2. Jika Anda ingin menggunakan port yang berbeda secara lokal, gunakan *local-port* parameter:8081.

```
kubectl port-forward -n flink deployments/basic-example 8080:8081
```

3. Di browser web, navigasikan ke `http://localhost:8081` (atau `http://localhost:local-port` jika Anda menggunakan port lokal khusus) untuk mengakses Dasbor Web Flink. Dasbor ini menampilkan informasi tentang aplikasi Flink yang sedang berjalan, seperti status pekerjaan, jumlah TaskManagers, dan metrik serta log untuk pekerjaan tersebut.



Gunakan konfigurasi pemantauan untuk memantau pekerjaan operator Flink Kubernetes dan Flink

Konfigurasi pemantauan memungkinkan Anda dengan mudah mengatur pengarsipan log aplikasi Flink dan log operator Anda ke S3 and/or CloudWatch (Anda dapat memilih salah satu atau keduanya). Melakukan hal itu menambahkan sidecar FluentD ke JobManager Anda TaskManager dan pod dan selanjutnya meneruskan log komponen ini ke sink yang dikonfigurasi.

Note

Anda harus mengatur Peran IAM untuk akun layanan untuk operator Flink Anda dan pekerjaan Flink Anda (Akun Layanan) untuk dapat menggunakan fitur ini, karena memerlukan interaksi dengan yang lain. Layanan AWS Anda harus mengatur ini menggunakan IRSA di [Menyiapkan operator Flink Kubernetes untuk Amazon EMR di EKS](#).

Log aplikasi Flink

Anda dapat menentukan konfigurasi ini dengan cara berikut.

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
```

```

name: basic-example
spec:
  image: FLINK IMAGE TAG
  imagePullPolicy: Always
  flinkVersion: v1_17
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
  executionRoleArn: JOB EXECUTION ROLE
  jobManager:
    resource:
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
  job:
    jarURI: local:///opt/flink/examples/streaming/StateMachineExample.jar
  monitoringConfiguration:
    s3MonitoringConfiguration:
      logUri: S3 BUCKET
    cloudWatchMonitoringConfiguration:
      logGroupName: LOG GROUP NAME
      logStreamNamePrefix: LOG GROUP STREAM PREFIX
  sidecarResources:
    limits:
      cpuLimit: 500m
      memoryLimit: 250Mi
  containerLogRotationConfiguration:
    rotationSize: 2GB
    maxFilesToKeep: 10

```

Berikut ini adalah opsi konfigurasi.

- `s3MonitoringConfiguration`— Kunci konfigurasi untuk mengatur penerusan ke S3
 - `logUri`(wajib) — jalur bucket S3 tempat Anda ingin menyimpan log Anda.
 - Jalur di S3 setelah log diunggah akan terlihat seperti berikut.
 - Tidak ada rotasi log yang diaktifkan:

```
s3://{logUri}/{POD NAME}/STDOUT or STDERR.gz
```

- Rotasi log diaktifkan. Anda dapat menggunakan file yang diputar dan file saat ini (satu tanpa cap tanggal).

```
s3://${logUri}/${POD_NAME}/STDOUT or STDERR.gz
```

Format berikut adalah angka yang bertambah.

```
s3://${logUri}/${POD_NAME}/stdout_YYYYMMDD_index.gz
```

- Izin IAM berikut diperlukan untuk menggunakan forwarder ini.

```
{
  "Effect": "Allow",
  "Action": [
    "s3:PutObject"
  ],
  "Resource": [
    "${S3_BUCKET_URI}/*",
    "${S3_BUCKET_URI}"
  ]
}
```

- `cloudWatchMonitoringConfiguration`— kunci konfigurasi untuk mengatur penerusan ke CloudWatch
- `logGroupName(wajib)` — nameof grup CloudWatch log yang ingin Anda kirim log (secara otomatis membuat grup jika tidak ada).
- `logStreamNamePrefix(opsional)` — nama aliran log yang ingin Anda kirim log ke. Nilai default adalah string kosong. Formatnya adalah sebagai berikut:

```
${logStreamNamePrefix}/${POD_NAME}/STDOUT or STDERR
```

- Izin IAM berikut diperlukan untuk menggunakan forwarder ini.

```
{
  "Effect": "Allow",
  "Action": [
    "logs:CreateLogStream",
    "logs:CreateLogGroup",
    "logs:PutLogEvents"
  ],
}
```

```

"Resource": [
  "arn:aws:logs:REGION:ACCOUNT-ID:log-group:{YOUR_LOG_GROUP_NAME}:*",
  "arn:aws:logs:REGION:ACCOUNT-ID:log-group:{YOUR_LOG_GROUP_NAME}"
]
}

```

- `sideCarResources`(opsional) — kunci konfigurasi untuk menetapkan batas sumber daya pada wadah sidecar Fluentbit yang diluncurkan.
- `memoryLimit`(opsional) - nilai default adalah 512Mi. Sesuaikan sesuai dengan kebutuhan Anda.
- `cpuLimit`(opsional) — opsi ini tidak memiliki default. Sesuaikan sesuai dengan kebutuhan Anda.
- `containerLogRotationConfiguration`(opsional) — mengontrol perilaku rotasi log kontainer. Agen tidak diaktifkan secara default.
 - `rotationSize`(wajib) - menentukan ukuran file untuk rotasi log. Kisaran nilai yang mungkin adalah dari 2KB hingga 2GB. Bagian unit numerik dari parameter `RotationSize` dilewatkan sebagai bilangan bulat. Karena nilai desimal tidak didukung, Anda dapat menentukan ukuran rotasi 1,5GB, misalnya, dengan nilai 1500MB. Defaultnya adalah 2GB.
 - `maxFilesToKeep`(wajib) — menentukan jumlah maksimum file untuk disimpan dalam wadah setelah rotasi telah terjadi. Nilai minimum adalah 1, dan nilai maksimum adalah 50. Default-nya adalah 10.

Log operator Flink

Kami juga dapat mengaktifkan pengarsipan log untuk operator dengan menggunakan opsi berikut dalam `values.yaml` file di instalasi bagan helm Anda. Anda dapat mengaktifkan S3, CloudWatch, atau keduanya.

```

monitoringConfiguration:
  s3MonitoringConfiguration:
    logUri: "S3-BUCKET"
    totalFileSize: "1G"
    uploadTimeout: "1m"
  cloudWatchMonitoringConfiguration:
    logGroupName: "flink-log-group"
    logStreamNamePrefix: "example-job-prefix-test-2"
  sideCarResources:
    limits:

```

```
cpuLimit: 1
memoryLimit: 800Mi
memoryBufferLimit: 700M
```

Berikut ini adalah opsi konfigurasi yang tersedia di `bawahmonitoringConfiguration`.

- `s3MonitoringConfiguration`— atur opsi ini untuk mengarsipkan ke S3.
- `logUri(wajib)` - Jalur bucket S3 tempat Anda ingin menyimpan log Anda.
- Berikut ini adalah format seperti apa jalur bucket S3 setelah log diunggah.
 - Tidak ada rotasi log yang diaktifkan.

```
s3://${logUri}/${POD NAME}/OPERATOR or WEBHOOK/STDOUT or STDERR.gz
```

- Rotasi log diaktifkan. Anda dapat menggunakan file yang diputar dan file saat ini (satu tanpa cap tanggal).

```
s3://${logUri}/${POD NAME}/OPERATOR or WEBHOOK/STDOUT or STDERR.gz
```

Indeks format berikut adalah angka yang bertambah.

```
s3://${logUri}/${POD NAME}/OPERATOR or WEBHOOK/stdout_YYYYMMDD_index.gz
```

- `cloudWatchMonitoringConfiguration`— kunci konfigurasi untuk mengatur penerusan ke CloudWatch
 - `logGroupName(wajib)` — nama grup CloudWatch log yang ingin Anda kirim log. Grup secara otomatis akan dibuat jika tidak ada.
 - `logStreamNamePrefix(opsional)` — nama aliran log yang ingin Anda kirim log ke. Nilai default adalah string kosong. Formatnya CloudWatch adalah sebagai berikut:

```
${logStreamNamePrefix}/${POD NAME}/STDOUT or STDERR
```

- `sideCarResources(opsional)` — kunci konfigurasi untuk menetapkan batas sumber daya pada wadah sidecar Fluentbit yang diluncurkan.
 - `memoryLimit(opsional)` — batas memori. Sesuaikan sesuai dengan kebutuhan Anda. Defaultnya adalah 512Mi.
 - `cpuLimit`— batas CPU. Sesuaikan sesuai dengan kebutuhan Anda. Tidak ada nilai default.

- `containerLogRotationConfiguration`(opsional): — mengontrol perilaku rotasi log kontainer. Agen tidak diaktifkan secara default.
- `rotationSize`(wajib) - menentukan ukuran file untuk rotasi log. Kisaran nilai yang mungkin adalah dari 2KB hingga 2GB. Bagian unit numerik dari parameter `RotationSize` dilewatkan sebagai bilangan bulat. Karena nilai desimal tidak didukung, Anda dapat menentukan ukuran rotasi 1,5GB, misalnya, dengan nilai 1500MB. Defaultnya adalah 2GB.
- `maxFilesToKeep`(wajib) — menentukan jumlah maksimum file untuk disimpan dalam wadah setelah rotasi telah terjadi. Nilai minimum adalah 1, dan nilai maksimum adalah 50. Default-nya adalah 10.

Bagaimana Flink mendukung ketersediaan tinggi dan ketahanan kerja

Bagian berikut menguraikan bagaimana Flink membuat pekerjaan lebih andal dan sangat tersedia. Ia melakukan ini melalui kemampuan bawaan seperti ketersediaan tinggi Flink dan berbagai kemampuan pemulihan jika terjadi kegagalan.

Topik

- [Menggunakan ketersediaan tinggi \(HA\) untuk Operator Flink dan Aplikasi Flink](#)
- [Mengoptimalkan waktu restart pekerjaan Flink untuk pemulihan tugas dan operasi penskalaan dengan Amazon EMR di EKS](#)
- [Penonaktifan Instans Spot yang anggun dengan Flink di Amazon EMR di EKS](#)

Menggunakan ketersediaan tinggi (HA) untuk Operator Flink dan Aplikasi Flink

Topik ini menunjukkan cara mengonfigurasi ketersediaan tinggi dan menjelaskan cara kerjanya untuk beberapa kasus penggunaan yang berbeda. Ini termasuk ketika Anda menggunakan Job manager dan ketika Anda menggunakan kubernetes asli Flink.

Operator Flink ketersediaan tinggi

Kami mengaktifkan ketersediaan tinggi untuk Operator Flink sehingga kami dapat gagal ke Operator Flink siaga untuk meminimalkan waktu henti di loop kontrol operator jika terjadi kegagalan.

Ketersediaan tinggi diaktifkan secara default dan jumlah default replika operator awal adalah 2. Anda dapat mengonfigurasi bidang replika di `values.yaml` file Anda untuk bagan helm.

Bidang berikut dapat disesuaikan:

- `replicas`(opsional, defaultnya adalah 2): Menyetel nomor ini menjadi lebih besar dari 1 membuat Operator siaga lainnya dan memungkinkan pemulihan pekerjaan Anda lebih cepat.
- `highAvailabilityEnabled`(opsional, defaultnya benar): Mengontrol apakah Anda ingin mengaktifkan HA. Menentukan parameter ini sebagai `true` memungkinkan dukungan penyebaran multi AZ, serta menetapkan parameter yang benar `flink-conf.yaml`.

Anda dapat menonaktifkan HA untuk operator Anda dengan mengatur konfigurasi berikut di `values.yaml` file Anda.

```
...
imagePullSecrets: []

replicas: 1

# set this to false if you don't want HA
highAvailabilityEnabled: false
...
```

Penyebaran multi AZ

Kami membuat pod operator di beberapa Availability Zone. Ini adalah kendala lunak, dan pod operator Anda akan dijadwalkan di AZ yang sama jika Anda tidak memiliki cukup sumber daya di AZ yang berbeda.

Menentukan replika pemimpin

Jika HA diaktifkan, replika menggunakan sewa untuk menentukan pemimpin mana dan menggunakan Sewa K8 untuk pemilihan pemimpin. JMs Anda dapat menjelaskan Sewa dan melihat bidang Identitas `.Spec.Holder` untuk menentukan pemimpin saat ini

```
kubectl describe lease <Helm Install Release Name>-<NAMESPACE>-lease -n <NAMESPACE> |
grep "Holder Identity"
```

Interaksi Flink-S3

Mengonfigurasi kredensial akses

Pastikan Anda telah mengonfigurasi IRSA dengan izin IAM yang sesuai untuk mengakses bucket S3.

Mengambil stoples pekerjaan dari mode Aplikasi S3

Operator Flink juga mendukung pengambilan stoples aplikasi dari S3. Anda hanya menyediakan lokasi S3 untuk JarUri dalam spesifikasi Anda FlinkDeployment .

Anda juga dapat menggunakan fitur ini untuk mengunduh artefak lain seperti PyFlink skrip. Skrip Python yang dihasilkan dijatuhkan di bawah jalur. `/opt/flink/usrlib/`

Contoh berikut menunjukkan bagaimana menggunakan fitur ini untuk PyFlink pekerjaan. Perhatikan bidang JarUri dan args.

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: python-example
spec:
  image: <YOUR CUSTOM PYFLINK IMAGE>
  emrReleaseLabel: "emr-6.12.0-flink-latest"
  flinkVersion: v1_16
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "1"
  serviceAccount: flink
  jobManager:
    highAvailabilityEnabled: false
    replicas: 1
    resource:
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
  job:
    jarURI: "s3://<S3-BUCKET>/scripts/pyflink.py" # Note, this will trigger the
    artifact download process
    entryClass: "org.apache.flink.client.python.PythonDriver"
    args: ["-pyclientexec", "/usr/local/bin/python3", "-py", "/opt/flink/usrlib/
    pyflink.py"]
    parallelism: 1
```

```
upgradeMode: stateless
```

Konektor Flink S3

Flink dikemas dengan dua konektor S3 (tercantum di bawah). Bagian berikut membahas kapan harus menggunakan konektor mana.

Checkpointing: Konektor Presto S3

- Setel skema S3 ke `s3p://`
- Konektor yang disarankan untuk digunakan ke pos pemeriksaan ke s3. Untuk informasi selengkapnya, lihat [khusus S3 dalam dokumentasi](#) Apache Flink.

Contoh FlinkDeployment spesifikasi:

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: basic-example
spec:
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
    state.checkpoints.dir: s3p://<BUCKET-NAME>/flink-checkpoint/
```

Membaca dan menulis ke S3: konektor Hadoop S3

- Atur skema S3 ke `s3://` atau `() s3a://`
- Konektor yang direkomendasikan untuk membaca dan menulis file dari S3 (hanya konektor S3 yang mengimplementasikan antarmuka [Flinks](#) Filesystem).
- Secara default, kami mengatur `fs.s3a.aws.credentials.provider` dalam `flink-conf.yaml` file, yaitu `com.amazonaws.auth.WebIdentityTokenCredentialsProvider`. Jika Anda mengganti default `flink-conf` sepenuhnya dan Anda berinteraksi dengan S3, pastikan untuk menggunakan penyedia ini.

Contoh FlinkDeployment spesifikasi

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
```

```
metadata:
  name: basic-example
spec:
  job:
    jarURI: local:///opt/flink/examples/streaming/WordCount.jar
    args: [ "--input", "s3a://<INPUT BUCKET>/PATH", "--output", "s3a://<OUTPUT BUCKET>/PATH" ]
    parallelism: 2
    upgradeMode: stateless
```

Manajer Pekerjaan Flink

Ketersediaan Tinggi (HA) untuk Penerapan Flink memungkinkan pekerjaan terus membuat kemajuan bahkan jika kesalahan sementara ditemukan dan crash Anda. JobManager Pekerjaan akan dimulai ulang tetapi dari pos pemeriksaan terakhir yang berhasil dengan HA diaktifkan. Tanpa HA diaktifkan, Kubernetes akan memulai ulang pekerjaan Anda JobManager, tetapi pekerjaan Anda akan dimulai sebagai pekerjaan baru dan akan kehilangan kemajuannya. Setelah mengonfigurasi HA, kami dapat memberi tahu Kubernetes untuk menyimpan metadata HA dalam penyimpanan persisten untuk referensi jika terjadi kegagalan sementara di JobManager dan kemudian melanjutkan pekerjaan kami dari pos pemeriksaan terakhir yang berhasil.

HA diaktifkan secara default untuk pekerjaan Flink Anda (jumlah replika disetel ke 2, yang mengharuskan Anda menyediakan lokasi penyimpanan S3 agar metadata HA tetap ada).

Konfigurasi HA

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: basic-example
spec:
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
  executionRoleArn: "<JOB EXECUTION ROLE ARN>"
  emrReleaseLabel: "emr-6.13.0-flink-latest"
  jobManager:
    resource:
      memory: "2048m"
      cpu: 1
    replicas: 2
    highAvailabilityEnabled: true
    storageDir: "s3://<S3 PERSISTENT STORAGE DIR>"
```

```
taskManager:
  resource:
    memory: "2048m"
    cpu: 1
```

Berikut ini adalah deskripsi untuk konfigurasi HA di atas di Job Manager (didefinisikan di bawah `spec.jobManager`):

- `highAvailabilityEnabled`(opsional, default adalah `true`): Setel ini ke `false` jika Anda tidak ingin HA diaktifkan dan tidak ingin menggunakan konfigurasi HA yang disediakan. Anda masih dapat memanipulasi bidang “replika” untuk mengonfigurasi HA secara manual.
- `replicas`(opsional, defaultnya adalah 2): Menyetel nomor ini menjadi lebih besar dari 1 membuat siaga lainnya JobManagers dan memungkinkan pemulihan pekerjaan Anda lebih cepat. Jika Anda menonaktifkan HA, Anda harus mengatur jumlah replika ke 1, atau Anda akan terus mendapatkan kesalahan validasi (hanya 1 replika yang didukung jika HA tidak diaktifkan).
- `storageDir`(required): Karena kami menggunakan jumlah replika sebagai 2 secara default, kami harus menyediakan StorageDir persisten. Saat ini bidang ini hanya menerima jalur S3 sebagai lokasi penyimpanan.

Lokalitas pod

Jika Anda mengaktifkan HA, kami juga mencoba mengkolokasi pod di AZ yang sama, yang mengarah pada peningkatan kinerja (mengurangi latensi jaringan dengan memiliki pod yang sama AZs). Ini adalah proses upaya terbaik, artinya jika Anda tidak memiliki cukup sumber daya di AZ di mana sebagian besar Pod Anda dijadwalkan, Pod yang tersisa masih akan dijadwalkan tetapi mungkin berakhir pada node di luar AZ ini.

Menentukan replika pemimpin

Jika HA diaktifkan, replika menggunakan sewa untuk menentukan pemimpin mana dan menggunakan Configmap K8s sebagai datastore untuk menyimpan metadata ini. JMs Jika Anda ingin menentukan pemimpin, Anda dapat melihat konten Configmap dan melihat kunci di `org.apache.flink.k8s.leader.restserver` bawah data untuk menemukan pod K8s dengan alamat IP. Anda juga dapat menggunakan perintah bash berikut.

```
ip=$(kubectl get configmap -n <NAMESPACE> <JOB-NAME>-cluster-config-map -o json | jq -r ".data[\"org.apache.flink.k8s.leader.restserver\"]" | awk -F: '{print $2}' | awk -F '/' '{print $3}')
```

```
kubectl get pods -n NAMESPACE -o json | jq -r ".items[] | select(.status.podIP == \\\$ip\\") | .metadata.name"
```

Pekerjaan Flink - Kubernetes asli

Amazon EMR 6.13.0 dan yang lebih tinggi mendukung Kubernetes asli Flink untuk menjalankan aplikasi Flink dalam mode ketersediaan tinggi pada cluster Amazon EKS.

Note

Anda harus memiliki bucket Amazon S3 yang dibuat untuk menyimpan metadata ketersediaan tinggi saat mengirimkan pekerjaan Flink Anda. Jika Anda tidak ingin menggunakan fitur ini, Anda dapat menonaktifkannya. Ini diaktifkan secara default.

Untuk mengaktifkan fitur ketersediaan tinggi Flink, berikan parameter Flink berikut saat Anda menjalankan perintah [CLI `run-application`](#). Parameter didefinisikan di bawah contoh.

```
-Dhigh-availability.type=kubernetes \
-Dhigh-availability.storageDir=S3://DOC-EXAMPLE-STORAGE-BUCKET \
-
Dfs.s3a.aws.credentials.provider="com.amazonaws.auth.WebIdentityTokenCredentialsProvider" \
-Dkubernetes.jobmanager.replicas=3 \
-Dkubernetes.cluster-id=example-cluster
```

- **Dhigh-availability.storageDir**- Bucket Amazon S3 tempat Anda ingin menyimpan metadata ketersediaan tinggi untuk pekerjaan Anda.

Dkubernetes.jobmanager.replicas— Jumlah pod Job Manager yang akan dibuat sebagai bilangan bulat lebih besar dari 1.

Dkubernetes.cluster-id— ID unik yang mengidentifikasi cluster Flink.

Mengoptimalkan waktu restart pekerjaan Flink untuk pemulihan tugas dan operasi penskalaan dengan Amazon EMR di EKS

Ketika tugas gagal atau ketika operasi penskalaan terjadi, Flink mencoba untuk mengeksekusi kembali tugas dari pos pemeriksaan terakhir selesai. Proses restart bisa memakan waktu satu menit

atau lebih lama untuk dijalankan, tergantung pada ukuran status pos pemeriksaan dan jumlah tugas paralel. Selama periode restart, tugas backlog dapat menumpuk untuk pekerjaan itu. Ada beberapa cara, bahwa Flink mengoptimalkan kecepatan pemulihan dan memulai ulang grafik eksekusi untuk meningkatkan stabilitas pekerjaan.

Halaman ini menjelaskan beberapa cara Amazon EMR Flink dapat meningkatkan waktu restart pekerjaan selama pemulihan tugas atau operasi penskalaan pada instance spot. Instans spot adalah kapasitas komputasi yang tidak terpakai yang tersedia dengan diskon. Ini memiliki perilaku unik, termasuk gangguan sesekali, jadi penting untuk memahami bagaimana Amazon EMR di EKS menangani ini, termasuk bagaimana Amazon EMR di EKS melakukan penonaktifan dan memulai kembali pekerjaan.

Topik

- [Pemulihan tugas-lokal](#)
- [Pemulihan tugas-lokal dengan pemasangan volume Amazon EBS](#)
- [Pos pemeriksaan inkremental berbasis log generik](#)
- [Pemulihan berbutir halus](#)
- [Mekanisme restart gabungan dalam penjadwal adaptif](#)

Pemulihan tugas-lokal

Note

Pemulihan tugas-lokal didukung dengan Flink di Amazon EMR di EKS 6.14.0 dan lebih tinggi.

Dengan pos pemeriksaan Flink, setiap tugas menghasilkan snapshot statusnya yang ditulis Flink ke penyimpanan terdistribusi seperti Amazon S3. Dalam kasus pemulihan, tugas mengembalikan keadaan mereka dari penyimpanan terdistribusi. Penyimpanan terdistribusi memberikan toleransi kesalahan dan dapat mendistribusikan kembali status selama penskalaan ulang karena dapat diakses oleh semua node.

Namun, toko terdistribusi jarak jauh juga memiliki kelemahan: semua tugas harus membaca statusnya dari lokasi terpencil melalui jaringan. Hal ini dapat mengakibatkan waktu pemulihan yang lama untuk negara bagian besar selama pemulihan tugas atau operasi penskalaan.

Masalah waktu pemulihan yang lama ini diselesaikan dengan pemulihan tugas-lokal. Tugas menulis status mereka di pos pemeriksaan ke dalam penyimpanan sekunder yang bersifat lokal untuk tugas, seperti pada disk lokal. Mereka juga menyimpan status mereka di penyimpanan utama, atau Amazon S3 dalam kasus kami. Selama pemulihan, penjadwal menjadwalkan tugas pada Task Manager yang sama di mana tugas berjalan lebih awal sehingga mereka dapat pulih dari penyimpanan status lokal alih-alih membaca dari penyimpanan status jarak jauh. Untuk informasi selengkapnya, lihat [Task-Local Recovery di Dokumentasi Apache Flink](#).

Tes benchmark kami dengan pekerjaan sampel telah menunjukkan bahwa waktu pemulihan telah dikurangi dari menit menjadi beberapa detik dengan pemulihan tugas-lokal diaktifkan.

Untuk mengaktifkan pemulihan tugas-lokal, atur konfigurasi berikut di file Anda. `flink-conf.yaml`. Tentukan nilai interval checkpointing dalam milidetik.

```
state.backend.local-recovery: true
state.backend: hadoop or rocksdb
state.checkpoints.dir: s3://STORAGE-BUCKET-PATH/checkpoint
execution.checkpointing.interval: 15000
```

Pemulihan tugas-lokal dengan pemasangan volume Amazon EBS

Note

Pemulihan tugas-lokal oleh Amazon EBS didukung dengan Flink di Amazon EMR di EKS 6.15.0 dan lebih tinggi.

Dengan Flink di Amazon EMR di EKS, Anda dapat secara otomatis menyediakan volume Amazon EBS ke pod untuk pemulihan TaskManager tugas lokal. Mount overlay default dilengkapi dengan volume 10 GB, yang cukup untuk pekerjaan dengan status lebih rendah. Pekerjaan dengan status besar dapat mengaktifkan opsi pemasangan volume EBS otomatis. TaskManagerPod secara otomatis dibuat dan dipasang selama pembuatan pod dan dihapus selama penghapusan pod.

Gunakan langkah-langkah berikut untuk mengaktifkan pemasangan volume EBS otomatis untuk Flink di Amazon EMR di EKS:

1. Ekspor nilai untuk variabel berikut yang akan Anda gunakan dalam langkah mendatang.

```
export AWS_REGION=aa-example-1
```

```
export FLINK_EKS_CLUSTER_NAME=my-cluster
export AWS_ACCOUNT_ID=111122223333
```

2. Buat atau perbarui file kubeconfig YAMM untuk klaster Anda.

```
aws eks update-kubeconfig --name $FLINK_EKS_CLUSTER_NAME --region $AWS_REGION
```

3. Buat akun layanan IAM untuk driver Amazon EBS Container Storage Interface (CSI) di cluster Amazon EKS Anda.

```
eksctl create iamserviceaccount \
  --name ebs-csi-controller-sa \
  --namespace kube-system \
  --region $AWS_REGION \
  --cluster $FLINK_EKS_CLUSTER_NAME \
  --role-name TLR_${AWS_REGION}_${FLINK_EKS_CLUSTER_NAME} \
  --role-only \
  --attach-policy-arn arn:aws:iam::aws:policy/service-role/
AmazonEBSCSIDriverPolicy \
  --approve
```

4. Buat driver Amazon EBS CSI dengan perintah berikut:

```
eksctl create addon \
  --name aws-ebs-csi-driver \
  --region $AWS_REGION \
  --cluster $FLINK_EKS_CLUSTER_NAME \
  --service-account-role-arn arn:aws:iam::${AWS_ACCOUNT_ID}:role/TLR_
${AWS_REGION}_${FLINK_EKS_CLUSTER_NAME}
```

5. Buat kelas penyimpanan Amazon EBS dengan perintah berikut:

```
cat # EOF # storage-class.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ebs-sc
provisioner: ebs.csi.aws.com
volumeBindingMode: WaitForFirstConsumer
EOF
```

Dan kemudian terapkan kelas:

```
kubectl apply -f storage-class.yaml
```

6. Helm instal operator Amazon EMR Flink Kubernetes dengan opsi untuk membuat akun layanan. Ini menciptakan `emr-containers-sa-flink` untuk digunakan dalam penerapan Flink.

```
helm install flink-kubernetes-operator flink-kubernetes-operator/ \
  --set jobServiceAccount.create=true \
  --set rbac.jobRole.create=true \
  --set rbac.jobRoleBinding.create=true
```

7. Untuk mengirimkan pekerjaan Flink dan mengaktifkan penyediaan otomatis volume EBS untuk pemulihan tugas-lokal, atur konfigurasi berikut di file Anda. `flink-conf.yaml` Sesuaikan batas ukuran untuk ukuran status pekerjaan. Atur `serviceAccount` ke `emr-containers-sa-flink`. Tentukan nilai interval checkpointing dalam milidetik. Dan hilangkan. `executionRoleArn`

```
flinkConfiguration:
  task.local-recovery.ebs.enable: true
  kubernetes.taskmanager.local-recovery.persistentVolumeClaim.sizeLimit: 10Gi
  state.checkpoints.dir: s3://BUCKET-PATH/checkpoint
  state.backend.local-recovery: true
  state.backend: hasmap or rocksdb
  state.backend.incremental: "true"
  execution.checkpointing.interval: 15000
  serviceAccount: emr-containers-sa-flink
```

Saat Anda siap untuk menghapus plugin driver Amazon EBS CSI, gunakan perintah berikut:

```
# Detach Attached Policy
aws iam detach-role-policy --role-name TLR_${AWS_REGION}_${FLINK_EKS_CLUSTER_NAME}
--policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy
# Delete the created Role
aws iam delete-role --role-name TLR_${AWS_REGION}_${FLINK_EKS_CLUSTER_NAME}
# Delete the created service account
eksctl delete iamserviceaccount --name ebs-csi-controller-sa --namespace kube-system
--cluster $FLINK_EKS_CLUSTER_NAME --region $AWS_REGION
# Delete Addon
eksctl delete addon --name aws-ebs-csi-driver --cluster $FLINK_EKS_CLUSTER_NAME --
region $AWS_REGION
# Delete the EBS storage class
```

```
kubectl delete -f storage-class.yaml
```

Pos pemeriksaan inkremental berbasis log generik

Note

Pemeriksaan inkremental berbasis log generik didukung dengan Flink di Amazon EMR di EKS 6.14.0 dan yang lebih tinggi.

Checkpointing inkremental berbasis log generik ditambahkan di Flink 1.16 untuk meningkatkan kecepatan pos pemeriksaan. Interval pos pemeriksaan yang lebih cepat sering mengakibatkan pengurangan pekerjaan pemulihan karena lebih sedikit peristiwa yang perlu diproses ulang setelah pemulihan. Untuk informasi selengkapnya, lihat [Meningkatkan kecepatan dan stabilitas pos pemeriksaan dengan pos pemeriksaan inkremental berbasis log generik di Blog Apache Flink](#).

Dengan pekerjaan sampel, tes benchmark kami telah menunjukkan bahwa waktu pos pemeriksaan berkurang dari menit menjadi beberapa detik dengan pos pemeriksaan inkremental berbasis log generik.

Untuk mengaktifkan pos pemeriksaan inkremental berbasis log generik, atur konfigurasi berikut di file Anda. `flink-conf.yaml` Tentukan nilai interval checkpointing dalam milidetik.

```
state.backend.changelog.enabled: true
state.backend.changelog.storage: filesystem
dstl.dfs.base-path: s3://bucket-path/changelog
state.backend.local-recovery: true
state.backend: rocksdb
state.checkpoints.dir: s3://bucket-path/checkpoint
execution.checkpointing.interval: 15000
```

Pemulihan berbutir halus

Note

Dukungan pemulihan berbutir halus untuk penjadwal default didukung dengan Flink di Amazon EMR di EKS 6.14.0 dan yang lebih tinggi. Dukungan pemulihan berbutir halus dalam penjadwal adaptif tersedia dengan Flink di Amazon EMR di EKS 6.15.0 dan lebih tinggi.

Ketika tugas gagal selama eksekusi, Flink mengatur ulang seluruh grafik eksekusi dan memicu eksekusi ulang lengkap dari pos pemeriksaan terakhir yang diselesaikan. Ini lebih mahal daripada hanya menjalankan kembali tugas yang gagal. Pemulihan berbutir halus hanya memulai kembali komponen yang terhubung dengan pipa dari tugas yang gagal. Dalam contoh berikut, grafik pekerjaan memiliki 5 simpul (AkeE). Semua koneksi antara simpul disalurkan dengan distribusi pointwise, dan `parallelism.default` untuk pekerjaan diatur ke. 2

```
A # B # C # D # E
```

Untuk contoh ini, ada total 10 tugas yang berjalan. Pipeline pertama (a1toe1) berjalan pada a TaskManager (TM1), dan pipeline kedua (a2toe2) berjalan pada yang lain TaskManager (TM2).

```
a1 # b1 # c1 # d1 # e1
a2 # b2 # c2 # d2 # e2
```

Ada dua komponen yang terhubung dengan pipa: a1 # e1, dan a2 # e2. Jika salah satu TM1 atau TM2 gagal, kegagalan hanya berdampak pada 5 tugas dalam pipeline tempat TaskManager sedang berjalan. Strategi restart hanya memulai komponen pipelined yang terpengaruh.

Pemulihan berbutir halus hanya berfungsi dengan pekerjaan Flink paralel sempurna. Ini tidak didukung dengan `keyBy()` atau `redistribute()` operasi. Untuk informasi selengkapnya, lihat [FLIP-1: Pemulihan Berbutir Halus dari Kegagalan Tugas](#) dalam proyek Jira Proposal Peningkatan Flink.

Untuk mengaktifkan pemulihan berbutir halus, atur konfigurasi berikut di file Anda. `flink-conf.yaml`

```
jobmanager.execution.failover-strategy: region
restart-strategy: exponential-delay or fixed-delay
```

Mekanisme restart gabungan dalam penjadwal adaptif

Note

Mekanisme restart gabungan dalam penjadwal adaptif didukung dengan Flink di Amazon EMR di EKS 6.15.0 dan lebih tinggi.

Penjadwal adaptif dapat menyesuaikan paralelisme pekerjaan berdasarkan slot yang tersedia. Ini secara otomatis mengurangi paralelisme jika tidak cukup slot yang tersedia agar sesuai dengan paralelisme pekerjaan yang dikonfigurasi. Jika slot baru tersedia, pekerjaan ditingkatkan lagi ke paralelisme pekerjaan yang dikonfigurasi. Penjadwal adaptif menghindari waktu henti di tempat kerja ketika tidak ada cukup sumber daya yang tersedia. Ini adalah penjadwal yang didukung untuk Flink Autoscaler. Kami merekomendasikan penjadwal adaptif dengan Amazon EMR Flink karena alasan ini. Namun, penjadwal adaptif mungkin melakukan beberapa restart dalam waktu singkat, satu restart untuk setiap sumber daya baru yang ditambahkan. Hal ini dapat menyebabkan penurunan kinerja dalam pekerjaan.

Dengan Amazon EMR 6.15.0 dan yang lebih tinggi, Flink memiliki mekanisme restart gabungan dalam penjadwal adaptif yang membuka jendela restart ketika sumber daya pertama ditambahkan, dan kemudian menunggu hingga interval jendela yang dikonfigurasi dari default 1 menit. Ini melakukan restart tunggal ketika ada sumber daya yang cukup tersedia untuk menjalankan pekerjaan dengan paralelisme yang dikonfigurasi atau ketika interval waktu habis.

Dengan contoh pekerjaan, pengujian benchmark kami menunjukkan bahwa fitur ini memproses 10% rekaman lebih banyak daripada perilaku default saat Anda menggunakan adaptive scheduler dan Flink autoscaler.

Untuk mengaktifkan mekanisme restart gabungan, atur konfigurasi berikut di `flink-conf.yaml` file Anda.

```
jobmanager.adaptive-scheduler.combined-restart.enabled: true
jobmanager.adaptive-scheduler.combined-restart.window-interval: 1m
```

Penonaktifan Instans Spot yang anggun dengan Flink di Amazon EMR di EKS

Flink dengan Amazon EMR di EKS dapat meningkatkan waktu restart pekerjaan selama pemulihan tugas atau operasi penskalaan.

Ikhtisar

Amazon EMR di EKS merilis 6.15.0 dan yang lebih tinggi mendukung penonaktifan Manajer Tugas di Instans Spot di Amazon EMR di EKS dengan Apache Flink. Sebagai bagian dari fitur ini, Amazon EMR di EKS dengan Flink menyediakan kemampuan berikut:

- **Just-in-time checkpointing** — Pekerjaan streaming Flink dapat merespons interupsi Instans Spot, melakukan pos pemeriksaan just-in-time (JIT) dari pekerjaan yang sedang berjalan, dan mencegah penjadwalan tugas tambahan pada Instans Spot ini. Pos pemeriksaan JIT didukung dengan penjadwal default dan adaptif.
- **Mekanisme restart gabungan** — Mekanisme restart gabungan melakukan upaya terbaik untuk memulai kembali pekerjaan setelah mencapai paralelisme sumber daya target atau akhir jendela yang dikonfigurasi saat ini. Ini juga mencegah restart pekerjaan berturut-turut yang mungkin disebabkan oleh beberapa penghentian Instans Spot. Mekanisme restart gabungan hanya tersedia dengan penjadwal adaptif.

Kemampuan ini memberikan manfaat sebagai berikut:

- Anda dapat memanfaatkan Instans Spot untuk menjalankan Manajer Tugas dan mengurangi pengeluaran kluster.
- Peningkatan keaktifan untuk Spot Instance Task Manager menghasilkan ketahanan yang lebih tinggi dan penjadwalan pekerjaan yang lebih efisien.
- Pekerjaan Flink Anda akan memiliki lebih banyak uptime karena akan ada lebih sedikit restart dari penghentian Instans Spot.

Cara kerja dekomisioning yang anggun

Pertimbangkan contoh berikut: Anda menyediakan EMR Amazon di kluster EKS yang menjalankan Apache Flink, dan Anda menentukan node Sesuai Permintaan untuk Job Manager, dan node Instans Spot untuk Task Manager. Dua menit sebelum penghentian, Task Manager menerima pemberitahuan gangguan.

Dalam skenario ini, Job Manager akan menangani sinyal interupsi Instans Spot, memblokir penjadwalan tugas tambahan pada Instans Spot, dan memulai pemeriksaan JIT untuk pekerjaan streaming.

Kemudian, Job Manager akan memulai ulang grafik pekerjaan hanya setelah ada ketersediaan sumber daya baru yang cukup untuk memenuhi paralelisme pekerjaan saat ini di jendela interval restart saat ini. Interval jendela restart ditentukan berdasarkan durasi penggantian Instans Spot, pembuatan pod Task Manager baru, dan pendaftaran dengan Job Manager.

Prasyarat

Untuk menggunakan dekomisioning yang anggun, buat dan jalankan pekerjaan streaming di Amazon EMR di kluster EKS yang menjalankan Apache Flink. Aktifkan Penjadwal Adaptif dan Manajer Tugas yang dijadwalkan pada setidaknya satu Instance Spot, seperti yang ditunjukkan pada contoh berikut. Anda harus menggunakan node On-Demand untuk Job Manager, dan Anda dapat menggunakan node On-Demand untuk Task Manager selama setidaknya ada satu Instance Spot juga.

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: deployment_name
spec:
  flinkVersion: v1_17
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
    cluster.taskmanager.graceful-decommission.enabled: "true"
    execution.checkpointing.interval: "240s"
    jobmanager.adaptive-scheduler.combined-restart.enabled: "true"
    jobmanager.adaptive-scheduler.combined-restart.window-interval : "1m"
  serviceAccount: flink
  jobManager:
    resource:
      memory: "2048m"
      cpu: 1
    nodeSelector:
      'eks.amazonaws.com/capacityType': 'ON_DEMAND'
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
    nodeSelector:
      'eks.amazonaws.com/capacityType': 'SPOT'
  job:
    jarURI: fLink_job_jar_path
```

Konfigurasi

Bagian ini mencakup sebagian besar konfigurasi yang dapat Anda tentukan untuk kebutuhan penonaktifan Anda.

Key	Deskripsi	Nilai default	Nilai yang dapat diterima
<code>cluster.taskmanager.graceful-decommission.enabled</code>	Aktifkan penonaktifan Task Manager yang anggun.	<code>true</code>	<code>true, false</code>
<code>jobmanager.adaptive-scheduler.combined-restart.enabled</code>	Aktifkan mekanisme restart gabungan di Adaptive Scheduler.	<code>false</code>	<code>true, false</code>
<code>jobmanager.adaptive-scheduler.combined-restart.window-interval</code>	Interval jendela restart gabungan untuk melakukan restart gabungan untuk pekerjaan tersebut. Sebuah integer tanpa unit ditafsirkan sebagai milidetik.	<code>1m</code>	Contoh: <code>30,60s,3m,1h</code>

Menggunakan Autoscaler untuk aplikasi Flink

Autoscaler operator dapat membantu meringankan tekanan balik dengan mengumpulkan metrik dari pekerjaan Flink dan secara otomatis menyesuaikan paralelisme pada tingkat titik pekerjaan. Berikut ini adalah contoh seperti apa konfigurasi Anda:

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  ...
spec:
  ...
```

```
flinkVersion: v1_18
flinkConfiguration:
  job.autoscaler.enabled: "true"
  job.autoscaler.stabilization.interval: 1m
  job.autoscaler.metrics.window: 5m
  job.autoscaler.target.utilization: "0.6"
  job.autoscaler.target.utilization.boundary: "0.2"
  job.autoscaler.restart.time: 2m
  job.autoscaler.catch-up.duration: 5m
  pipeline.max-parallelism: "720"
  ...
```

Konfigurasi ini menggunakan nilai default untuk rilis terbaru Amazon EMR. Jika Anda menggunakan versi lain, Anda mungkin memiliki nilai yang berbeda.

Note

Mulai Amazon EMR 7.2.0, Anda tidak perlu menyertakan awalan `kubernetes.operator` dalam konfigurasi Anda. Jika Anda menggunakan 7.1.0 atau lebih rendah, Anda harus menggunakan awalan sebelum setiap konfigurasi. Misalnya, Anda harus menentukan `kubernetes.operator.job.autoscaler.scaling.enabled`.

Berikut ini adalah opsi konfigurasi untuk autoscaler.

- `job.autoscaler.scaling.enabled`— menentukan apakah akan mengaktifkan eksekusi penskalaan vertikal oleh autoscaler. Nilai default-nya `true`. Jika Anda menonaktifkan konfigurasi ini, penskalaan otomatis hanya mengumpulkan metrik dan mengevaluasi paralelisme yang disarankan untuk setiap simpul tetapi tidak meningkatkan pekerjaan.
- `job.autoscaler.stabilization.interval`— periode stabilisasi di mana tidak ada penskalaan baru yang akan dieksekusi. Default adalah 5 menit.
- `job.autoscaler.metrics.window`— ukuran jendela agregasi metrik penskalaan. Semakin besar jendela, semakin halus dan stabil, tetapi autoscaler mungkin lebih lambat untuk bereaksi terhadap perubahan beban mendadak. Default adalah 15 menit. Kami menyarankan Anda bereksperimen dengan menggunakan nilai antara 3 hingga 60 menit.
- `job.autoscaler.target.utilization`— pemanfaatan simpul target untuk memberikan kinerja pekerjaan yang stabil dan beberapa buffer untuk fluktuasi beban. Defaultnya `0.7` menargetkan 70% utilization/load untuk verteks pekerjaan.

- `job.autoscaler.target.utilization.boundary`— batas pemanfaatan simpul target yang berfungsi sebagai buffer ekstra untuk menghindari penskalaan langsung pada fluktuasi beban. Defaultnya adalah `0.3`, yang berarti 30% deviasi dari pemanfaatan target diperbolehkan sebelum memicu tindakan penskalaan.
- `ob.autoscaler.restart.time`— waktu yang diharapkan untuk me-restart aplikasi. Default adalah 5 menit.
- `job.autoscaler.catch-up.duration`— waktu yang diharapkan untuk catch up, yang berarti sepenuhnya memproses setiap backlog setelah operasi penskalaan selesai. Default adalah 5 menit. Dengan menurunkan durasi catch-up, autoscaler harus memesan lebih banyak kapasitas ekstra untuk tindakan penskalaan.
- `pipeline.max-parallelism`— paralelisme maksimum yang dapat digunakan autoscaler. Autoscaler mengabaikan batas ini jika lebih tinggi dari paralelisme maks yang dikonfigurasi dalam konfigurasi Flink atau langsung pada setiap operator. Defaultnya adalah `-1`. Perhatikan bahwa autoscaler menghitung paralelisme sebagai pembagi bilangan paralelisme maks oleh karena itu disarankan untuk memilih pengaturan paralelisme maks yang memiliki banyak pembagi daripada mengandalkan default yang disediakan Flink. Kami merekomendasikan penggunaan kelipatan 60 untuk konfigurasi ini, seperti 120, 180, 240, 360, 720 dll.

Untuk halaman referensi konfigurasi yang lebih detail, lihat Konfigurasi [Autoscaler](#).

Autotuning parameter Autoscaler

Bagian ini menjelaskan perilaku penyyetelan otomatis untuk berbagai versi EMR Amazon. Ini juga masuk ke detail mengenai konfigurasi auto-scaling yang berbeda.

Note

Amazon EMR 7.2.0 dan yang lebih tinggi menggunakan konfigurasi open source `job.autoscaler.restart.time-tracking.enabled` untuk mengaktifkan estimasi waktu penskalaan ulang. Estimasi waktu skala ulang memiliki fungsionalitas yang sama dengan autotuning Amazon EMR, jadi Anda tidak perlu menetapkan nilai empiris secara manual ke waktu restart.

Anda masih dapat menggunakan autotuning Amazon EMR jika Anda menggunakan Amazon EMR 7.1.0 atau lebih rendah.

7.2.0 and higher

Amazon EMR 7.2.0 dan yang lebih tinggi mengukur waktu restart aktual yang diperlukan untuk menerapkan keputusan penskalaan otomatis. Dalam rilis 7.1.0 dan yang lebih rendah, Anda harus menggunakan konfigurasi `job.autoscaler.restart.time` untuk mengonfigurasi perkiraan waktu restart maksimum secara manual. Dengan menggunakan konfigurasi `job.autoscaler.restart.time-tracking.enabled`, Anda hanya perlu memasukkan waktu restart untuk penskalaan pertama. Setelah itu, operator mencatat waktu restart aktual dan akan menggunakannya untuk penskalaan berikutnya.

Untuk mengaktifkan pelacakan ini, gunakan perintah berikut:

```
job.autoscaler.restart.time-tracking.enabled: true
```

Berikut ini adalah konfigurasi terkait untuk estimasi waktu penskalaan ulang.

Konfigurasi	Diperlukan	Default	Deskripsi
<code>job.autoscaler.restart.time-tracking.enabled</code>	Tidak	False	Menunjukkan apakah Flink Autoscaler harus secara otomatis menyetel konfigurasi dari waktu ke waktu untuk mengoptimalkan desisi penskalaan. Perhatikan bahwa Autoscaler hanya dapat melakukan autotune parameter Autoscaler. <code>restart.time</code>
<code>job.autoscaler.restart.time</code>	Tidak	5m	Waktu restart yang diharapkan yang digunakan Amazon EMR di EKS hingga operator dapat menentukan waktu restart aktual dari penskalaan sebelumnya.
<code>job.autoscaler.restart.time-tracking.limit</code>	Tidak	15m	Waktu restart maksimum yang diamati saat

Konfigurasi	Diperlukan	Default	Deskripsi
			job.autoscaler.restart.time-tracking.enabled diatur ketru.

Berikut ini adalah contoh spesifikasi penerapan yang dapat Anda gunakan untuk mencoba estimasi waktu penskalaan ulang:

```

apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: autoscaling-example
spec:
  flinkVersion: v1_18
  flinkConfiguration:

    # Autoscaler parameters
    job.autoscaler.enabled: "true"
    job.autoscaler.scaling.enabled: "true"
    job.autoscaler.stabilization.interval: "5s"
    job.autoscaler.metrics.window: "1m"

    job.autoscaler.restart.time-tracking.enabled: "true"
    job.autoscaler.restart.time: "2m"
    job.autoscaler.restart.time-tracking.limit: "10m"

    jobmanager.scheduler: adaptive
    taskmanager.numberOfTaskSlots: "1"
    pipeline.max-parallelism: "12"

  executionRoleArn: <JOB ARN>
  emrReleaseLabel: emr-7.13.0-flink-latest
  jobManager:
    highAvailabilityEnabled: false
    storageDir: s3://<s3_bucket>/flink/autoscaling/ha/
    replicas: 1
    resource:
      memory: "1024m"
      cpu: 0.5
  taskManager:
    resource:

```

```

    memory: "1024m"
    cpu: 0.5
  job:
    jarURI: s3://<s3_bucket>/some-job-with-back-pressure
    parallelism: 1
    upgradeMode: stateless

```

Untuk mensimulasikan tekanan balik, gunakan spesifikasi penerapan berikut.

```

job:
  jarURI: s3://<s3_bucket>/pyflink-script.py
  entryClass: "org.apache.flink.client.python.PythonDriver"
  args: ["-py", "/opt/flink/usrlib/pyflink-script.py"]
  parallelism: 1
  upgradeMode: stateless

```

Unggah skrip Python berikut ke bucket S3 Anda.

```

import logging
import sys
import time
import random

from pyflink.datastream import StreamExecutionEnvironment
from pyflink.table import StreamTableEnvironment

TABLE_NAME="orders"
QUERY=f"""
CREATE TABLE {TABLE_NAME} (
  id INT,
  order_time AS CURRENT_TIMESTAMP,
  WATERMARK FOR order_time AS order_time - INTERVAL '5' SECONDS
)
WITH (
  'connector' = 'datagen',
  'rows-per-second'='10',
  'fields.id.kind'='random',
  'fields.id.min'='1',
  'fields.id.max'='100'
);
"""

def create_backpressure(i):

```

```

    time.sleep(2)
    return i

def autoscaling_demo():
    env = StreamExecutionEnvironment.get_execution_environment()
    t_env = StreamTableEnvironment.create(env)
    t_env.execute_sql(QUERY)
    res_table = t_env.from_path(TABLE_NAME)

    stream = t_env.to_data_stream(res_table) \
        .shuffle().map(lambda x: create_backpressure(x)) \
        .print()
    env.execute("Autoscaling demo")

if __name__ == '__main__':
    logging.basicConfig(stream=sys.stdout, level=logging.INFO, format="%(message)s")
    autoscaling_demo()

```

Untuk memverifikasi bahwa estimasi waktu penskalaan ulang berfungsi, pastikan pencatatan DEBUG level operator Flink diaktifkan. Contoh di bawah ini menunjukkan cara memperbarui file bagan helm. `values.yaml` Kemudian instal ulang bagan helm yang diperbarui dan jalankan pekerjaan Flink Anda lagi.

```

log4j-operator.properties: |+
  # Flink Operator Logging Overrides
  rootLogger.level = DEBUG

```

Dapatkan nama pod pemimpin Anda.

```

ip=$(kubectl get configmap -n $NAMESPACE <job-name>-cluster-config-map -o json | jq
-r ".data[\"org.apache.flink.k8s.leader.restserver\"]" | awk -F: '{print $2}' | awk
-F '/' '{print $3}')

kubectl get pods -n $NAMESPACE -o json | jq -r ".items[]" | select(.status.podIP ==
\"$ip\") | .metadata.name"

```

Jalankan perintah berikut untuk mendapatkan waktu restart aktual yang digunakan dalam evaluasi metrik.

```

kubectl logs <FLINK-OPERATOR-POD-NAME> -c flink-kubernetes-operator -n <OPERATOR-
NAMESPACE> -f | grep "Restart time used in scaling summary computation"

```

Anda akan melihat log yang mirip dengan yang berikut ini. Perhatikan bahwa hanya penskalaan pertama yang digunakan `job.autoscaler.restart.time`. Penskalaan selanjutnya menggunakan waktu restart yang diamati.

```
2024-05-16 17:17:32,590 o.a.f.a.ScalingExecutor [DEBUG][default/autoscaler-example] Restart time used in scaling summary computation: PT2M
2024-05-16 17:19:03,787 o.a.f.a.ScalingExecutor [DEBUG][default/autoscaler-example] Restart time used in scaling summary computation: PT14S
2024-05-16 17:19:18,976 o.a.f.a.ScalingExecutor [DEBUG][default/autoscaler-example] Restart time used in scaling summary computation: PT14S
2024-05-16 17:20:50,283 o.a.f.a.ScalingExecutor [DEBUG][default/autoscaler-example] Restart time used in scaling summary computation: PT14S
2024-05-16 17:22:21,691 o.a.f.a.ScalingExecutor [DEBUG][default/autoscaler-example] Restart time used in scaling summary computation: PT14S
```

7.0.0 and 7.1.0

Flink Autoscaler bawaan open source menggunakan banyak metrik untuk membuat keputusan penskalaan terbaik. Namun, nilai default yang digunakan untuk perhitungannya dimaksudkan untuk dapat diterapkan pada sebagian besar beban kerja dan mungkin tidak optimal untuk pekerjaan tertentu. Fitur autotuning yang ditambahkan ke EMR Amazon pada versi EKS dari Operator Flink melihat tren historis yang diamati pada metrik tertentu yang ditangkap dan kemudian mencoba menghitung nilai paling optimal yang disesuaikan untuk pekerjaan yang diberikan.

Konfigurasi	Diperlukan	Default	Deskripsi
<code>kubernetes.operator.job.autoscaler.autotune.enable</code>	Tidak	False	Menunjukkan apakah Flink Autoscaler harus secara otomatis menyetel konfigurasi dari waktu ke waktu untuk mengoptimalkan desisi penskalaan penskalaan otomatis. Saat ini, Autoscaler hanya dapat melakukan autotune parameter Autoscaler. <code>restart.time</code>

Konfigurasi	Diperlukan	Default	Deskripsi
kubernetes.operator.job.autoscaler.autotune.metrics.history.max.count	Tidak	3	Menunjukkan berapa banyak historis Amazon EMR pada metrik EKS yang disimpan Autoscaler di Amazon EMR di peta konfigurasi metrik EKS.
kubernetes.operator.job.autoscaler.autotune.metrics.restart.count	Tidak	3	Menunjukkan berapa banyak jumlah restart yang dilakukan Autoscaler sebelum mulai menghitung waktu restart rata-rata untuk pekerjaan tertentu.

Untuk mengaktifkan autotuning, Anda harus menyelesaikan yang berikut ini:

- Setel `kubernetes.operator.job.autoscaler.autotune.enable`: ke `true`
- Setel `metrics.job.status.enable`: ke `TOTAL_TIME`
- Mengikuti pengaturan [Menggunakan Autoscaler untuk aplikasi Flink untuk mengaktifkan Autoscaling](#).

Berikut ini adalah contoh spesifikasi penerapan yang dapat Anda gunakan untuk mencoba autotuning.

```

apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: autoscaling-example
spec:
  flinkVersion: v1_18
  flinkConfiguration:

    # Autotuning parameters
    kubernetes.operator.job.autoscaler.autotune.enable: "true"
    kubernetes.operator.job.autoscaler.autotune.metrics.history.max.count: "2"
    kubernetes.operator.job.autoscaler.autotune.metrics.restart.count: "1"

```

```
metrics.job.status.enable: TOTAL_TIME

# Autoscaler parameters
kubernetes.operator.job.autoscaler.enabled: "true"
kubernetes.operator.job.autoscaler.scaling.enabled: "true"
kubernetes.operator.job.autoscaler.stabilization.interval: "5s"
kubernetes.operator.job.autoscaler.metrics.window: "1m"

jobmanager.scheduler: adaptive

taskmanager.numberOfTaskSlots: "1"
state.savepoints.dir: s3://<S3_bucket>/autoscaling/savepoint/
state.checkpoints.dir: s3://<S3_bucket>/flink/autoscaling/checkpoint/
pipeline.max-parallelism: "4"

executionRoleArn: <JOB_ARN>
emrReleaseLabel: emr-6.14.0-flink-latest
jobManager:
  highAvailabilityEnabled: true
  storageDir: s3://<S3_bucket>/flink/autoscaling/ha/
  replicas: 1
  resource:
    memory: "1024m"
    cpu: 0.5
taskManager:
  resource:
    memory: "1024m"
    cpu: 0.5
job:
  jarURI: s3://<S3_bucket>/some-job-with-back-pressure
  parallelism: 1
  upgradeMode: last-state
```

Untuk mensimulasikan tekanan balik, gunakan spesifikasi penerapan berikut.

```
job:
  jarURI: s3://<S3_bucket>/pyflink-script.py
  entryClass: "org.apache.flink.client.python.PythonDriver"
  args: ["-py", "/opt/flink/usrlib/pyflink-script.py"]
  parallelism: 1
  upgradeMode: last-state
```

Unggah skrip Python berikut ke bucket S3 Anda.

```
import logging
import sys
import time
import random

from pyflink.datastream import StreamExecutionEnvironment
from pyflink.table import StreamTableEnvironment

TABLE_NAME="orders"
QUERY=f"""
CREATE TABLE {TABLE_NAME} (
  id INT,
  order_time AS CURRENT_TIMESTAMP,
  WATERMARK FOR order_time AS order_time - INTERVAL '5' SECONDS
)
WITH (
  'connector' = 'datagen',
  'rows-per-second'='10',
  'fields.id.kind'='random',
  'fields.id.min'='1',
  'fields.id.max'='100'
);
"""

def create_backpressure(i):
    time.sleep(2)
    return i

def autoscaling_demo():
    env = StreamExecutionEnvironment.get_execution_environment()
    t_env = StreamTableEnvironment.create(env)
    t_env.execute_sql(QUERY)
    res_table = t_env.from_path(TABLE_NAME)

    stream = t_env.to_data_stream(res_table) \
        .shuffle().map(lambda x: create_backpressure(x)) \
        .print()
    env.execute("Autoscaling demo")

if __name__ == '__main__':
    logging.basicConfig(stream=sys.stdout, level=logging.INFO, format="%(message)s")
    autoscaling_demo()
```

Untuk memverifikasi bahwa autotuner Anda berfungsi, gunakan perintah berikut. Perhatikan bahwa Anda harus menggunakan informasi pod pemimpin Anda sendiri untuk Operator Flink.

Pertama dapatkan nama pod pemimpin Anda.

```
ip=$(kubectl get configmap -n $NAMESPACE <job-name>-cluster-config-map -o json | jq
-r ".data[\"org.apache.flink.k8s.leader.restserver\"]" | awk -F: '{print $2}' | awk
-F '/' '{print $3}')

kubectl get pods -n $NAMESPACE -o json | jq -r ".items[] | select(.status.podIP ==
\"$ip\") | .metadata.name"
```

Setelah Anda memiliki nama pod pemimpin Anda, Anda dapat menjalankan perintah berikut.

```
kubectl logs -n $NAMESPACE -c flink-kubernetes-operator --follow <YOUR-FLINK-
OPERATOR-POD-NAME> | grep -E 'EmrEks|autotun|calculating|restart|autoscaler'
```

Anda akan melihat log yang mirip dengan yang berikut ini.

```
[m[33m2023-09-13 20:10:35,941[m [36mc.a.c.f.k.o.a.EmrEksMetricsAutotuner[m
[36m[DEBUG][flink/autoscaling-example] Using the latest
Emr Eks Metric for calculating restart.time for autotuning:
EmrEksMetrics(restartMetric=RestartMetric(restartingTime=65, numRestarts=1))

[m[33m2023-09-13 20:10:35,941[m [36mc.a.c.f.k.o.a.EmrEksMetricsAutotuner[m
[32m[INFO ][flink/autoscaling-example] Calculated average restart.time metric via
autotuning to be: PT0.065S
```

Pemeliharaan dan pemecahan masalah untuk pekerjaan Flink di Amazon EMR di EKS

Bagian berikut menguraikan cara mempertahankan pekerjaan Flink Anda yang sudah berjalan lama, dan memberikan panduan tentang cara memecahkan masalah umum dengan pekerjaan Flink.

Mempertahankan aplikasi Flink

Topik

- [Mode upgrade](#)

Aplikasi Flink biasanya dirancang untuk berjalan dalam jangka waktu yang lama seperti minggu, bulan, atau bahkan bertahun-tahun. Seperti semua layanan yang berjalan lama, aplikasi streaming Flink perlu dipertahankan. Ini termasuk perbaikan bug, peningkatan, dan migrasi ke cluster Flink versi yang lebih baru.

Ketika spesifikasi berubah `FlinkDeployment` dan `FlinkSessionJob` sumber daya, Anda perlu memutakhirkan aplikasi yang sedang berjalan. Untuk melakukan ini, operator menghentikan pekerjaan yang sedang berjalan (kecuali sudah ditangguhkan) dan menerapkannya kembali dengan spesifikasi terbaru dan, untuk aplikasi stateful, status dari proses sebelumnya.

Pengguna mengontrol cara mengelola status saat aplikasi stateful berhenti dan memulihkan dengan `upgradeMode` pengaturan. `JobSpec`

Mode upgrade

Pengenalan opsional

Tanpa kewarganegaraan

Upgrade aplikasi stateless dari status kosong.

Keadaan terakhir

Peningkatan cepat dalam keadaan aplikasi apa pun (bahkan untuk pekerjaan yang gagal), tidak memerlukan pekerjaan yang sehat karena selalu menggunakan pos pemeriksaan terbaru yang berhasil. Pemulihan manual mungkin diperlukan jika metadata HA hilang. Untuk membatasi waktu pekerjaan mungkin mundur saat mengambil pos pemeriksaan terbaru yang dapat Anda konfigurasi `kubernetes.operator.job.upgrade.last-state.max.allowed.checkpoint.age`. Jika pos pemeriksaan lebih tua dari nilai yang dikonfigurasi, savepoint akan diambil sebagai gantinya untuk pekerjaan yang sehat. Ini tidak didukung dalam mode Sesi.

Savepoint

Gunakan savepoint untuk peningkatan, memberikan keamanan maksimal dan kemungkinan untuk berfungsi sebagai titik. backup/fork Savepoint akan dibuat selama proses upgrade. Perhatikan bahwa pekerjaan Flink harus dijalankan untuk memungkinkan savepoint dibuat. Jika pekerjaan dalam keadaan tidak sehat, pos pemeriksaan terakhir akan digunakan (kecuali `kubernetes.operator.job.upgrade.last-state-fallback.enabled` disetel ke `false`). Jika pos pemeriksaan terakhir tidak tersedia, peningkatan pekerjaan akan gagal.

Pemecahan masalah

Bagian ini menjelaskan cara memecahkan masalah dengan Amazon EMR di EKS. Untuk informasi tentang cara memecahkan masalah umum dengan Amazon EMR, lihat [Memecahkan masalah kluster di Panduan Manajemen EMR Amazon](#).

- [Memecahkan masalah pekerjaan yang menggunakan PersistentVolumeClaims \(PVC\)](#)
- [Memecahkan masalah Amazon EMR pada penskalaan otomatis vertikal EKS](#)
- [Memecahkan masalah Amazon EMR pada operator EKS Spark](#)

Memecahkan masalah Apache Flink di Amazon EMR di EKS

Pemetaan sumber daya tidak ditemukan saat menginstal bagan Helm

Anda mungkin menemukan pesan galat berikut saat menginstal bagan Helm.

```
Error: INSTALLATION FAILED: pulling from host 1234567890.dkr.ecr.us-west-2.amazonaws.com failed with status code [manifests 6.13.0]: 403 Forbidden Error: INSTALLATION FAILED: unable to build kubernetes objects from release manifest: [resource mapping not found for name: "flink-operator-serving-cert" namespace: "<the namespace to install your operator>" from "": no matches for kind "Certificate" in version "cert-manager.io/v1"
```

```
ensure CRDs are installed first, resource mapping not found for name: "flink-operator-selfsigned-issuer" namespace: "<the namespace to install your operator>" " from "": no matches for kind "Issuer" in version "cert-manager.io/v1"
```

```
ensure CRDs are installed first].
```

Untuk mengatasi kesalahan ini, instal cert-manager untuk mengaktifkan penambahan komponen webhook. Anda harus menginstal cert-manager ke setiap cluster Amazon EKS yang Anda gunakan.

```
kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.12.0
```

Layanan AWS akses ditolak kesalahan

Jika Anda melihat access denied kesalahan, konfirmasi bahwa peran IAM untuk `operatorExecutionRoleArn` dalam `values.yaml` file bagan Helm memiliki izin yang benar.

Pastikan juga peran IAM `executionRoleArn` di bawah `FlinkDeployment` spesifikasi Anda memiliki izin yang benar.

FlinkDeployment macet

Jika Anda `FlinkDeployment` terhenti dalam keadaan tertangkap, gunakan langkah-langkah berikut untuk menghapus paksa penyebaran:

1. Edit proses penerapan.

```
kubectl edit -n Flink Namespace flinkdeployments/App Name
```

2. Hapus finalizer ini.

```
finalizers:
  - flinkdeployments.flink.apache.org/finalizer
```

3. Hapus penyebaran.

```
kubectl delete -n Flink Namespace flinkdeployments/App Name
```

`AWSBadRequestException` masalah s3a saat menjalankan aplikasi Flink dalam keikutsertaan Wilayah AWS

Jika Anda menjalankan aplikasi Flink dalam [opt-in Wilayah AWS](#), Anda mungkin melihat kesalahan berikut:

```
Caused by: org.apache.hadoop.fs.s3a.AWSBadRequestException: getFileStatus on
s3://flink.txt: com.amazonaws.services.s3.model.AmazonS3Exception: Bad Request
(Service: Amazon S3; Status Code: 400; Error Code: 400 Bad Request; Request ID:
ABCDEFGHIJKL; S3 Extended Request ID:
ABCDEFGHIJKLMNOP=; Proxy: null), S3 Extended Request ID: ABCDEFGHIJKLMNOP=:400 Bad
Request: Bad Request
(Service: Amazon S3; Status Code: 400; Error Code: 400 Bad Request; Request ID:
ABCDEFGHIJKL; S3 Extended Request ID: ABCDEFGHIJKLMNOP=; Proxy: null)
```

```
Caused by: org.apache.hadoop.fs.s3a.AWSBadRequestException: getS3Region on flink-
application: software.amazon.awssdk.services.s3.model.S3Exception: null
(Service: S3, Status Code: 400, Request ID: ABCDEFGHIJKLMNOP, Extended Request ID:
ABCDEFGHIJKLMNOPQRST==):null: null
```

```
(Service: S3, Status Code: 400, Request ID: ABCDEFGHIJKLMNOP, Extended Request ID:
AH142uDNaTUF0us/5IIVNvSakBcMjMCH7dd37ky0vE6jhABCDEFGHIJKLMNQPQRST==)
```

Untuk memperbaiki kesalahan ini, gunakan konfigurasi berikut dalam file `FlinkDeployment` definisi Anda.

```
spec:
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
    fs.s3a.endpoint.region: OPT_IN_AWS_REGION_NAME
```

Kami juga menyarankan Anda menggunakan penyedia SDKv2 kredensi:

```
fs.s3a.aws.credentials.provider:
  software.amazon.awssdk.auth.credentials.WebIdentityTokenFileCredentialsProvider
```

Jika Anda ingin menggunakan penyedia SDKv1 kredensial, pastikan SDK mendukung Wilayah keikutsertaan Anda. Untuk informasi lebih lanjut, lihat [aws-sdk-java GitHub repositori](#).

Jika Anda mendapatkan `S3 AWSSBadRequestException` ketika Anda menjalankan pernyataan SQL Flink di Region opt-in, pastikan bahwa Anda mengatur konfigurasi `fs.s3a.endpoint.region: OPT_IN_AWS_REGION_NAME` dalam spesifikasi konfigurasi flink Anda.

S3A AWSSBad RequestException saat menjalankan pekerjaan sesi Flink di wilayah CN

Untuk Amazon EMR rilis 6.15.0 - 7.2.0, Anda mungkin menemukan pesan galat berikut saat menjalankan tugas sesi Flink di wilayah CN. Ini termasuk Tiongkok (Beijing) dan Tiongkok (Ningxia):

```
Error:
{"type":"org.apache.flink.kubernetes.operator.exception.ReconciliationException","message":"or
      getFileStatus on s3://ABCDPath:
software.amazon.awssdk.services.s3.model.S3Exception: null (Service: S3, Status Code:
400, Request ID: ABCDEFGH, Extended Request ID:
      ABCDEFGH:null: null (Service: S3, Status Code: 400, Request ID:
ABCDEFGH, Extended Request ID: ABCDEFGH","additionalMetadata":{},"throwableList":

[{"type":"org.apache.hadoop.fs.s3a.AWSSBadRequestException","message":"getFileStatus on
s3://ABCDPath: software.amazon.awssdk.services.s3.model.S3Exception:
      null (Service: S3, Status Code: 400, Request ID: ABCDEFGH, Extended
Request ID: ABCDEFGH:null: null (Service: S3, Status Code: 400, Request ID: ABCDEFGH,
```

```

Extended Request ID: ABCDEFGH", "additionalMetadata": {}},
{"type": "software.amazon.awssdk.services.s3.model.S3Exception", "message": "null
(Service: S3, Status Code: 400,
Request ID: ABCDEFGH, Extended Request ID:
ABCDEFGH", "additionalMetadata": {}]]}

```

Ada kesadaran akan masalah ini. Tim sedang bekerja untuk menambal operator flink untuk semua versi rilis ini. Namun, sebelum kita menyelesaikan tambalan, untuk memperbaiki kesalahan ini, Anda perlu mengunduh bagan helm operator flink, menghapusnya (ekstrak file terkompresi) dan membuat perubahan konfigurasi di bagan helm.

Langkah-langkah spesifiknya adalah sebagai berikut:

1. Ubah ke, secara khusus mengubah direktori ke, folder lokal Anda untuk bagan helm, dan jalankan baris perintah berikut untuk menarik bagan helm dan untar (ekstrak) itu.

```

helm pull oci://public.ecr.aws/emr-on-eks/flink-kubernetes-operator \
--version $VERSION \
--namespace $NAMESPACE

```

```

tar -zxvf flink-kubernetes-operator-$VERSION.tgz

```

2. Masuk ke folder bagan helm dan temukan `templates/flink-operator.yaml` filenya.
3. Temukan `flink-operator-config` ConfigMap dan tambahkan `fs.s3a.endpoint.region` konfigurasi berikut di file `flink-conf.yaml`. Contoh:

```

{{- if .Values.defaultConfiguration.create }}
apiVersion: v1
kind: ConfigMap
metadata:
  name: flink-operator-config
  namespace: {{ .Release.Namespace }}
  labels:
    {{- include "flink-operator.labels" . | nindent 4 }}
data:
  flink-conf.yaml: |+
  fs.s3a.endpoint.region: {{ .Values.emrContainers.awsRegion }}

```

4. Instal bagan helm lokal dan jalankan pekerjaan Anda.

Rilis yang didukung untuk Amazon EMR di EKS dengan Apache Flink

Apache Flink tersedia dengan EMR Amazon berikut pada rilis EKS. Untuk informasi tentang semua rilis yang tersedia, lihat [Amazon EMR pada rilis EKS](#).

Label rilis	Java	Flink	Operator Flink
emr-7.2.0-flink-latest	17	1.18.1	-
emr-7.2.0-flink-k8s-operator-latest	11	-	1.8.0
emr-7.1.0-flink-latest	17	1.18.1	-
emr-7.1.0-flink-k8s-operator-latest	11	-	1.6.1
emr-7.0.0-flink-latest	11	1.18.0	-
emr-7.0.0-flink-k8s-operator-latest	11	-	1.6.1
emr-6.15.0-flink-latest	11	1.17.1	-
emr-6.15.0-flink-k8s-operator-latest	11	-	1.6.0
emr-6.14.0-flink-latest	11	1.17.1	-
emr-6.14.0-flink-k8s-operator-latest	11	-	1.6.0
emr-6.13.0-flink-latest	11	1.17.0	-
emr-6.13.0-flink-k8s-operator-latest	11	-	1.5.0

Menjalankan pekerjaan Spark dengan Amazon EMR di EKS

Job run adalah unit kerja, seperti jar Spark, PySpark skrip, atau kueri SparkSQL, yang Anda kirimkan ke Amazon EMR di EKS. Topik ini memberikan ikhtisar tentang mengelola pekerjaan yang dijalankan menggunakan AWS CLI, melihat pekerjaan berjalan menggunakan konsol EMR Amazon, dan memecahkan masalah kesalahan umum menjalankan pekerjaan.

Perhatikan bahwa Anda tidak dapat menjalankan pekerjaan IPv6 Spark di Amazon EMR di EKS

Note

Sebelum Anda mengirimkan pekerjaan dengan Amazon EMR di EKS, Anda harus menyelesaikan langkah-langkahnya. [Menyiapkan Amazon EMR di EKS](#)

Topik

- [Menjalankan pekerjaan Spark dengan StartJobRun](#)
- [Menjalankan pekerjaan Spark dengan operator Spark](#)
- [Menjalankan pekerjaan Spark dengan spark-submit](#)
- [Menggunakan Apache Livy dengan Amazon EMR di EKS](#)
- [Mengelola Amazon EMR pada pekerjaan EKS](#)
- [Menggunakan template pekerjaan](#)
- [Menggunakan templat pod](#)
- [Menggunakan kebijakan coba lagi pekerjaan](#)
- [Menggunakan rotasi log peristiwa Spark](#)
- [Menggunakan rotasi log kontainer Spark](#)
- [Menggunakan penskalaan otomatis vertikal dengan pekerjaan Amazon EMR Spark](#)

Menjalankan pekerjaan Spark dengan **StartJobRun**

Bagian ini mencakup langkah-langkah persiapan terperinci untuk menyiapkan lingkungan Anda untuk menjalankan pekerjaan Spark dan kemudian memberikan step-by-step instruksi untuk mengirimkan pekerjaan yang dijalankan dengan parameter tertentu.

Topik

- [Menyiapkan Amazon EMR di EKS](#)
- [Kirim pekerjaan yang dijalankan dengan StartJobRun](#)
- [Menggunakan klasifikasi pengirim pekerjaan](#)
- [Menggunakan klasifikasi default penampung EMR Amazon](#)

Menyiapkan Amazon EMR di EKS

Selesaikan tugas-tugas berikut untuk menyiapkan Amazon EMR di EKS. Jika Anda telah mendaftar untuk Amazon Web Services (AWS) dan telah menggunakan Amazon EKS, Anda hampir siap untuk menggunakan Amazon EMR di EKS. Lewati salah satu tugas yang telah Anda selesaikan.

Note

Anda juga dapat mengikuti [Amazon EMR di EKS Workshop](#) untuk menyiapkan semua sumber daya yang diperlukan untuk menjalankan pekerjaan Spark di Amazon EMR di EKS. Lokakarya ini juga menyediakan otomatisasi dengan menggunakan CloudFormation templat untuk membuat sumber daya yang diperlukan bagi Anda untuk memulai. Untuk templat dan praktik terbaik lainnya, lihat [Panduan Praktik Terbaik Kontainer EMR](#) kami di GitHub

1. [Instal atau perbarui ke versi terbaru AWS CLI](#)
2. [Siapkan kubectl dan eksctl](#)
3. [Memulai dengan Amazon EKS — eksctl](#)
4. [Aktifkan akses cluster untuk Amazon EMR di EKS](#)
5. [Aktifkan Peran IAM untuk kluster EKS](#)
6. [Memberikan pengguna akses ke Amazon EMR di EKS](#)
7. [Daftarkan kluster Amazon EKS dengan Amazon EMR](#)

Aktifkan akses kluster untuk Amazon EMR di EKS

Bagian berikut menunjukkan beberapa cara untuk mengaktifkan akses cluster. Yang pertama adalah dengan menggunakan Amazon EKS cluster access management (CAM) dan yang terakhir menunjukkan cara mengambil langkah-langkah manual untuk mengaktifkan akses cluster.

Aktifkan akses cluster menggunakan EKS Access Entry (disarankan)

Note

aws-auth ConfigMap itu sudah usang. [Metode yang disarankan untuk mengelola akses ke Kubernetes API adalah Access Entries.](#)

Amazon EMR terintegrasi dengan [Amazon EKS cluster access management \(CAM\)](#), sehingga Anda dapat mengotomatiskan konfigurasi kebijakan AuthN dan AuthZ yang diperlukan untuk menjalankan pekerjaan Amazon EMR Spark di ruang nama cluster Amazon EKS. Saat Anda membuat kluster virtual dari namespace kluster Amazon EKS, Amazon EMR secara otomatis mengonfigurasi semua izin yang diperlukan, sehingga Anda tidak perlu menambahkan langkah tambahan apa pun ke alur kerja Anda saat ini.

Note

Integrasi Amazon EMR dengan Amazon EKS CAM hanya didukung untuk EMR Amazon baru di kluster virtual EKS. Anda tidak dapat memigrasi kluster virtual yang ada untuk menggunakan integrasi ini.

Prasyarat

- Pastikan Anda menjalankan versi 2.15.3 atau lebih tinggi AWS CLI
- Cluster Amazon EKS Anda harus menggunakan versi 1.23 atau lebih tinggi.

Pengaturan

Untuk mengatur integrasi antara Amazon EMR dan operasi AccessEntry API dari Amazon EKS, pastikan Anda telah menyelesaikan item berikut:

- Pastikan authenticationMode kluster Amazon EKS Anda disetel keAPI_AND_CONFIG_MAP.

```
aws eks describe-cluster --name <eks-cluster-name>
```

Jika belum, atur authenticationMode keAPI_AND_CONFIG_MAP.

```
aws eks update-cluster-config
  --name <eks-cluster-name>
  --access-config authenticationMode=API_AND_CONFIG_MAP
```

Untuk informasi selengkapnya tentang mode autentikasi, lihat Mode [otentikasi kluster](#).

- Pastikan bahwa [peran IAM](#) yang Anda gunakan untuk menjalankan operasi `CreateVirtualCluster` dan `DeleteVirtualCluster` API juga memiliki izin berikut:

```
{
  "Effect": "Allow",
  "Action": [
    "eks:CreateAccessEntry"
  ],
  "Resource":
  "arn:<AWS_PARTITION>:eks:<AWS_REGION>:<AWS_ACCOUNT_ID>:cluster/<EKS_CLUSTER_NAME>"
},
{
  "Effect": "Allow",
  "Action": [
    "eks:DescribeAccessEntry",
    "eks>DeleteAccessEntry",
    "eks>ListAssociatedAccessPolicies",
    "eks:AssociateAccessPolicy",
    "eks:DisassociateAccessPolicy"
  ],
  "Resource": "arn:<AWS_PARTITION>:eks:<AWS_REGION>:<AWS_ACCOUNT_ID>:access-entry/
<EKS_CLUSTER_NAME>/role/<AWS_ACCOUNT_ID>/AWSServiceRoleForAmazonEMRContainers/*"
}
```

- Untuk membuat kebijakan yang paling tidak permisif, tambahkan kunci konteks `accessScope` dan `namespaces`. Contoh berikut menunjukkan kebijakan dengan kunci konteks ini:

```
{
  "Effect": "Allow",
  "Action": [
    "eks:CreateAccessEntry"
  ],
  "Resource": [
    "arn:<AWS_PARTITION>:eks:<AWS_REGION>:<AWS_ACCOUNT_ID>:cluster/<EKS_CLUSTER_NAME>"
  ]
}
```

```

    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "eks:DescribeAccessEntry",
      "eks>DeleteAccessEntry",
      "eks:ListAssociatedAccessPolicies"
    ],
    "Resource": [
      "arn:<AWS_PARTITION>:eks:<AWS_REGION>:<AWS_ACCOUNT_ID>:access-
entry/<EKS_CLUSTER_NAME>/role/<AWS_ACCOUNT_ID>/AWSServiceRoleForAmazonEMRContainers/
*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "eks:DisassociateAccessPolicy",
      "eks:AssociateAccessPolicy"
    ],
    "Resource": [
      "arn:<AWS_PARTITION>:eks:<AWS_REGION>:<AWS_ACCOUNT_ID>:access-
entry/<EKS_CLUSTER_NAME>/role/<AWS_ACCOUNT_ID>/AWSServiceRoleForAmazonEMRContainers/
*"
    ],
    "Condition": {
      "StringEquals": {
        "eks:accessScope": "namespace"
      },
      "ForAllValues:StringEquals": {
        "eks:namespaces": [
          "<EKS_NAMESPACE>"
        ]
      }
    }
  }
}

```

Konsep dan terminologi

Berikut ini adalah daftar terminologi dan konsep yang terkait dengan Amazon EKS CAM.

- Virtual cluster (VC) — representasi logis dari namespace yang dibuat di Amazon EKS. Ini adalah tautan 1:1 ke namespace cluster Amazon EKS. Anda dapat menggunakannya untuk menjalankan beban kerja Amazon EMR di kluster Amazon EKS dalam namespace yang ditentukan.
- Namespace — mekanisme untuk mengisolasi kelompok sumber daya dalam satu kluster EKS.
- Kebijakan akses — izin yang memberikan akses dan tindakan ke peran IAM dalam kluster EKS.
- Akses entri — entri yang dibuat dengan peran arn. Anda dapat menautkan entri akses ke kebijakan akses untuk menetapkan izin tertentu di kluster Amazon EKS.
- Kluster virtual terintegrasi entri akses EKS — cluster virtual yang dibuat menggunakan [operasi API entri akses](#) dari Amazon EKS.

Aktifkan akses cluster menggunakan **aws-auth**

Anda harus mengizinkan akses Amazon EMR di EKS ke namespace tertentu di kluster Anda dengan mengambil tindakan berikut: membuat peran Kubernetes, mengikat peran ke pengguna Kubernetes, dan memetakan pengguna Kubernetes dengan peran terkait layanan [AWSServiceRoleForAmazonEMRContainers](#). Tindakan ini diotomatiskan di `eksctl` ketika perintah pemetaan identitas IAM digunakan dengan `emr-containers` sebagai nama layanan. Anda dapat melakukan operasi ini dengan mudah menggunakan perintah berikut.

```
eksctl create iamidentitymapping \  
  --cluster my_eks_cluster \  
  --namespace kubernetes_namespace \  
  --service-name "emr-containers"
```

Ganti *my_eks_cluster* dengan nama cluster Amazon EKS Anda dan ganti *kubernetes_namespace* dengan namespace Kubernetes yang dibuat untuk menjalankan beban kerja Amazon EMR.

Important

Anda harus mengunduh `eksctl` terbaru menggunakan langkah sebelumnya [Mengatur kubectl dan eksctl](#) untuk menggunakan fungsi ini.

Langkah manual untuk mengaktifkan akses kluster untuk Amazon EMR di EKS

Anda juga dapat menggunakan langkah manual berikut untuk mengaktifkan akses kluster untuk Amazon EMR di EKS.

1. Buat peran Kubernetes di namespace tertentu

Amazon EKS 1.22 - 1.29

Dengan Amazon EKS 1.22 - 1.29, jalankan perintah berikut untuk membuat peran Kubernetes dalam namespace tertentu. Peran ini memberikan izin RBAC yang diperlukan untuk Amazon EMR di EKS.

```
namespace=my-namespace
cat - >>EOF | kubectl apply -f - >>namespace "${namespace}"
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: emr-containers
  namespace: ${namespace}
rules:
- apiGroups: [""]
  resources: ["namespaces"]
  verbs: ["get"]
- apiGroups: [""]
  resources: ["serviceaccounts", "services", "configmaps", "events", "pods",
"pods/log"]
  verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"deletecollection", "annotate", "patch", "label"]
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["create", "patch", "delete", "watch"]
- apiGroups: ["apps"]
  resources: ["statefulsets", "deployments"]
  verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"annotate", "patch", "label"]
- apiGroups: ["batch"]
  resources: ["jobs"]
  verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"annotate", "patch", "label"]
- apiGroups: ["extensions", "networking.k8s.io"]
  resources: ["ingresses"]
```

```

    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"annotate", "patch", "label"]
  - apiGroups: ["rbac.authorization.k8s.io"]
    resources: ["roles", "rolebindings"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"deletecollection", "annotate", "patch", "label"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"deletecollection", "annotate", "patch", "label"]
EOF

```

Amazon EKS 1.21 and below

Dengan Amazon EKS 1.21 dan di bawahnya, jalankan perintah berikut untuk membuat peran Kubernetes di namespace tertentu. Peran ini memberikan izin RBAC yang diperlukan untuk Amazon EMR di EKS.

```

namespace=my-namespace
cat - >>EOF | kubectl apply -f - >>namespace "${namespace}"
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: emr-containers
  namespace: ${namespace}
rules:
  - apiGroups: [""]
    resources: ["namespaces"]
    verbs: ["get"]
  - apiGroups: [""]
    resources: ["serviceaccounts", "services", "configmaps", "events", "pods",
"pods/log"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"deletecollection", "annotate", "patch", "label"]
  - apiGroups: [""]
    resources: ["secrets"]
    verbs: ["create", "patch", "delete", "watch"]
  - apiGroups: ["apps"]
    resources: ["statefulsets", "deployments"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"annotate", "patch", "label"]
  - apiGroups: ["batch"]

```

```

resources: ["jobs"]
verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"annotate", "patch", "label"]
- apiGroups: ["extensions"]
resources: ["ingresses"]
verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"annotate", "patch", "label"]
- apiGroups: ["rbac.authorization.k8s.io"]
resources: ["roles", "rolebindings"]
verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"deletecollection", "annotate", "patch", "label"]
- apiGroups: [""]
resources: ["persistentvolumeclaims"]
verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"deletecollection", "annotate", "patch", "label"]
EOF

```

2. Buat ikatan peran Kubernetes yang dicakup ke namespace

Jalankan perintah berikut untuk membuat peran mengikat Kubernetes mengikat di namespace yang diberikan. Pengikatan peran ini memberikan izin yang ditetapkan dalam peran yang dibuat di langkah sebelumnya ke pengguna bernama `emr-containers`. Pengguna ini mengidentifikasi [peran terkait layanan untuk Amazon EMR di EKS](#) dan dengan demikian memungkinkan Amazon EMR di EKS untuk melakukan tindakan seperti yang didefinisikan oleh peran yang Anda buat.

```

namespace=my-namespace

cat - <<EOF | kubectl apply -f - --namespace "${namespace}"
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: emr-containers
  namespace: ${namespace}
subjects:
- kind: User
  name: emr-containers
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: emr-containers
  apiGroup: rbac.authorization.k8s.io

```

EOF

3. Perbarui peta konfigurasi Kubernetes **aws-auth**

Anda dapat menggunakan salah satu opsi berikut untuk memetakan Amazon EMR di EKS peran terkait layanan dengan pengguna `emr-containers` yang terikat dengan peran Kubernetes pada langkah sebelumnya.

Opsi 1: Menggunakan **eksctl**

Jalankan perintah `eksctl` berikut untuk memetakan peran terkait layanan EMR Amazon di EKS dengan pengguna `emr-containers`.

```
eksctl create iamidentitymapping \
  --cluster my-cluster-name \
  --arn "arn:aws:iam::my-account-id:role/AWSServiceRoleForAmazonEMRContainers" \
  --username emr-containers
```

Opsi 2: Tanpa menggunakan `eksctl`

1. Jalankan perintah berikut untuk membuka peta konfigurasi `aws-auth` dalam editor teks.

```
kubectl edit -n kube-system configmap/aws-auth
```

Note

Jika Anda menerima kesalahan yang menyatakan `Error from server (NotFound): configmaps "aws-auth" not found`, lihat langkah-langkah dalam [Menambahkan peran pengguna](#) di Panduan Pengguna Amazon EKS untuk menerapkan stok ConfigMap.

2. Tambahkan detail peran terkait Amazon EMR di EKS ke bagian `mapRoles` dari ConfigMap, di bawah data. Tambahkan bagian ini jika belum ada dalam file. Bagian `mapRoles` yang diperbarui di bawah data terlihat seperti contoh berikut.

```
apiVersion: v1
data:
  mapRoles: |
    - rolearn: arn:aws:iam::<your-account-id>:role/
      AWSServiceRoleForAmazonEMRContainers
```

```
username: emr-containers
- ... <other previously existing role entries, if there's any>.
```

3. Simpan file , dan tutup editor teks Anda.

Aktifkan akses kluster untuk Amazon SageMaker Unified Studio

Amazon EMR di EKS dan Amazon SageMaker Unified Studio memerlukan akses ke cluster Amazon EKS. Silakan ikuti langkah-langkah di [Aktifkan akses kluster EKS untuk EMR di EKS dan SageMaker Unified Studio untuk menyediakan akses](#).

Aktifkan Peran IAM untuk kluster EKS

Topik berikut merinci opsi untuk mengaktifkan peran IAM.

Topik

- [Opsi 1: Aktifkan Identitas Pod EKS pada Kluster EKS](#)
- [Opsi 2: Aktifkan Peran IAM untuk Akun Layanan \(IRSA\) di cluster EKS](#)

Opsi 1: Aktifkan Identitas Pod EKS pada Kluster EKS

Pengaitan Identitas Pod Amazon EKS menyediakan kemampuan untuk mengelola kredensial untuk aplikasi Anda, mirip dengan cara profil instans Amazon EC2 memberikan kredensial ke instans Amazon EC2. Amazon EKS Pod Identity memberikan kredensi ke beban kerja Anda dengan API Auth EKS tambahan dan pod agen yang berjalan di setiap node.

Amazon EMR di EKS mulai mendukung identitas pod EKS sejak rilis emr-7.3.0 untuk model pengiriman. StartJobRun

Untuk informasi lebih lanjut tentang identitas pod EKS, lihat [Memahami cara kerja EKS Pod Identity](#).

Mengapa Identitas EKS Pod?

Sebagai bagian dari pengaturan EMR, Job Execution Role perlu menetapkan batas kepercayaan antara peran IAM dan akun layanan di namespace tertentu (dari cluster virtual EMR). Dengan IRSA, ini dicapai dengan memperbarui kebijakan kepercayaan Peran Pelaksanaan Job EMR. Namun, karena batas keras 4096 karakter pada panjang kebijakan kepercayaan IAM, ada kendala untuk berbagi Peran IAM Eksekusi Job tunggal di maksimum dua belas (12) kluster EKS.

Dengan dukungan EMR untuk Pod Identities, batas kepercayaan antara peran IAM dan akun layanan sekarang dikelola oleh tim EKS melalui API asosiasi identitas pod EKS.

Note

Batas keamanan untuk identitas pod EKS masih pada level akun layanan, bukan pada level pod.

Pertimbangan Identitas Pod

Untuk informasi tentang Batasan Identitas Pod, lihat [pertimbangan Identitas Pod EKS](#).

Siapkan Identitas Pod EKS di EKS Cluster

Periksa apakah izin yang diperlukan ada di `NodeInstanceRole`

Peran node `NodeInstanceRole` memerlukan izin bagi agen untuk melakukan `AssumeRoleForPodIdentity` tindakan di EKS Auth API. Anda dapat menambahkan berikut ini ke [AmazonEKSWorkerNodePolicy](#), yang ditentukan dalam Panduan Pengguna Amazon EKS, atau menggunakan kebijakan khusus.

Jika kluster EKS Anda dibuat dengan versi `eksctl` lebih tinggi dari 0.181.0., termasuk `AssumeRoleForPodIdentity` izin yang diperlukan `AmazonEKSWorkerNodePolicy`, akan dilampirkan ke peran node secara otomatis. Jika izin tidak ada, tambahkan izin berikut secara manual ke dalamnya `AmazonEKSWorkerNodePolicy` yang memungkinkan asumsi peran untuk identitas pod. Izin ini diperlukan oleh agen identitas pod EKS untuk mengambil kredensial untuk pod.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "eks-auth:AssumeRoleForPodIdentity"
      ],
      "Resource": [
        "*"
      ],
    },
  ],
}
```

```

    "Sid": "AllowEKSAUTHAssumeroleforpodidentity"
  }
]
}

```

Buat add-on agen identitas pod EKS

Gunakan perintah berikut untuk membuat add-on EKS Pod Identity Agent dengan versi terbaru:

```

aws eks create-addon --cluster-name cluster-name --addon-name eks-pod-identity-agent

kubectl get pods -n kube-system | grep 'eks-pod-identity-agent'

```

Gunakan langkah-langkah berikut untuk membuat add-on EKS Pod Identity Agent dari konsol Amazon EKS:

1. Buka konsol Amazon EKS: konsol [Amazon EKS](#).
2. Di panel navigasi sebelah kiri, pilih Clusters, lalu pilih nama cluster yang ingin Anda konfigurasi untuk add-on EKS Pod Identity Agent.
3. Pilih Add-onstab.
4. Pilih Get more add-ons
5. Pilih kotak di kanan atas kotak add-on untuk EKS Pod Identity Agent dan kemudian pilih Berikutnya.
6. Pada halaman Konfigurasi pengaturan add-on yang dipilih, pilih versi apa pun di daftar drop-down Versi.
7. (Opsional) Perluas pengaturan konfigurasi opsional untuk memasukkan konfigurasi tambahan. Misalnya, Anda dapat memberikan lokasi gambar kontainer alternatif dan `ImagePullSecrets`. Skema JSON dengan kunci yang diterima ditampilkan dalam skema Add-on konfigurasi.

Masukkan tombol konfigurasi dan nilai dalam nilai Konfigurasi.

8. Pilih Berikutnya.
9. Konfirmasikan bahwa pod agen berjalan di klaster Anda melalui CLI.

```

kubectl get pods -n kube-system | grep 'eks-pod-identity-agent'

```

Contoh output adalah sebagai berikut:

NAME	READY	STATUS	RESTARTS	AGE
eks-pod-identity-agent-gmqp7	1/1	Running	1 (24h ago)	24h
eks-pod-identity-agent-prnsh	1/1	Running	1 (24h ago)	24h

Ini mengatur yang baru DaemonSet di kube-system namespace. Amazon EKS Pod Identity Agent, yang berjalan di setiap node EKS, menggunakan [AssumeRoleForPodIdentity](#) action tersebut untuk mengambil kredensi sementara dari EKS Auth API. Kredensi ini kemudian tersedia untuk AWS SDK yang Anda jalankan di dalam container Anda.

Untuk informasi selengkapnya, periksa prasyarat di dokumen publik: [Siapkan Agen Identitas Pod Amazon EKS](#).

Membuat Peran Eksekusi Job

Membuat atau memperbarui peran eksekusi pekerjaan yang memungkinkan EKS Pod Identity

Untuk menjalankan beban kerja dengan Amazon EMR di EKS, Anda perlu membuat peran IAM. Kami menyebut peran ini sebagai peran eksekusi tugas dalam dokumentasi ini. Untuk informasi selengkapnya tentang cara membuat peran IAM, lihat [Membuat peran IAM](#) di Panduan pengguna.

Selain itu, Anda harus membuat kebijakan IAM yang menentukan izin yang diperlukan untuk peran eksekusi pekerjaan dan kemudian melampirkan kebijakan ini ke peran untuk mengaktifkan EKS Pod Identity.

Misalnya, Anda memiliki peran eksekusi pekerjaan berikut. Untuk informasi selengkapnya, lihat [Membuat peran eksekusi pekerjaan](#).

```
arn:aws:iam::111122223333:role/PodIdentityJobExecutionRole
```

Important

Amazon EMR di EKS secara otomatis membuat Akun Layanan Kubernetes, berdasarkan nama peran eksekusi pekerjaan Anda. Pastikan nama peran tidak terlalu panjang, karena pekerjaan Anda mungkin gagal jika kombinasicluster_name,pod_name, dan service_account_name melebihi batas panjang.

Job Execution Role Configuration - Pastikan peran eksekusi pekerjaan dibuat dengan izin kepercayaan di bawah ini untuk EKS Pod Identity. Untuk memperbarui peran eksekusi pekerjaan

yang ada, konfigurasi untuk mempercayai prinsip layanan EKS berikut sebagai izin tambahan dalam kebijakan kepercayaan. Izin kepercayaan ini dapat hidup berdampingan dengan kebijakan kepercayaan IRSA yang ada.

```
cat >trust-relationship.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
      "Effect": "Allow",
      "Principal": {
        "Service": "pods.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
EOF
```

Izin Pengguna: Pengguna memerlukan `iam:PassRole` izin untuk menjalankan panggilan `StartJobRun` API atau mengirimkan pekerjaan. Izin ini memungkinkan pengguna untuk meneruskan peran eksekusi pekerjaan ke EMR di EKS. Administrator Job harus memiliki izin secara default.

Di bawah ini adalah izin yang diperlukan untuk pengguna:

```
{
  "Effect": "Allow",
  "Action": "iam:PassRole",
  "Resource": "arn:aws:iam::111122223333:role/PodIdentityJobExecutionRole",
  "Condition": {
    "StringEquals": {
      "iam:PassedToService": "pods.eks.amazonaws.com"
    }
  }
}
```

Untuk lebih membatasi akses pengguna ke kluster EKS tertentu, tambahkan filter `AssociatedResourceArn` atribut ke kebijakan IAM. Ini membatasi asumsi peran untuk kluster EKS resmi, memperkuat kontrol keamanan tingkat sumber daya Anda.

```
"Condition": {
  "ArnLike": {
    "iam:AssociatedResourceARN": [
      "arn:aws:eks:us-west-2:111122223333:cluster/*"
    ]
  }
}
```

Siapkan asosiasi identitas pod EKS

Prasyarat

Pastikan IAM Identity yang membuat asosiasi identitas pod, seperti pengguna admin EKS, memiliki izin `eks:CreatePodIdentityAssociation` dan `iam:PassRole`.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "eks:CreatePodIdentityAssociation"
      ],
      "Resource": [
        "arn:aws:eks:*:*:cluster/*"
      ],
      "Sid": "AllowEKSCreatepodidentityassociation"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
        "arn:aws:iam:*:*:role/*"
      ],
      "Condition": {
```

```
    "StringEquals": {
      "iam:PassedToService": "pods.eks.amazonaws.com"
    }
  },
  "Sid": "AllowIAMPassrole"
}
]
```

Buat Asosiasi untuk peran dan akun layanan EMR

Create EMR role associations through the AWS CLI

Ketika Anda mengirimkan pekerjaan ke namespace Kubernetes, administrator harus membuat asosiasi antara peran eksekusi pekerjaan dan identitas akun layanan terkelola EMR. Perhatikan bahwa akun layanan terkelola EMR secara otomatis dibuat pada pengiriman tugas, dicakup ke namespace di mana tugas dikirimkan.

Dengan AWS CLI (versi di atas 2.24.0), jalankan perintah berikut untuk membuat asosiasi peran dengan identitas pod.

Jalankan perintah berikut untuk membuat asosiasi peran dengan identitas pod:

```
aws emr-containers create-role-associations \
  --cluster-name mycluster \
  --namespace mynamespace \
  --role-name JobExecutionRoleIRSAv2
```

Catatan:

- Setiap cluster dapat memiliki batas 1.000 asosiasi. Setiap peran eksekusi pekerjaan - pemetaan namespace akan membutuhkan 3 asosiasi untuk pod pengirim pekerjaan, driver, dan pelaksana.
- Anda hanya dapat mengaitkan peran yang berada di AWS akun yang sama dengan cluster. Anda dapat mendelegasikan akses dari akun lain ke peran di akun ini yang Anda konfigurasi untuk Identitas Pod EKS untuk digunakan. Untuk tutorial tentang mendelegasikan akses dan `AssumeRole`, lihat [tutorial IAM: Mendelegasikan akses di seluruh AWS akun menggunakan](#) peran IAM.

Create EMR role associations through Amazon EKS

EMR membuat akun layanan dengan pola penamaan tertentu saat pekerjaan dikirimkan. Untuk membuat asosiasi manual atau mengintegrasikan alur kerja ini dengan AWS SDK, ikuti langkah-langkah berikut:

Buat Nama Akun Layanan:

```
emr-containers-sa-spark-%(SPARK_ROLE)s-%(AWS_ACCOUNT_ID)s-
%(BASE36_ENCODED_ROLE_NAME)s
```

Contoh di bawah ini membuat asosiasi peran untuk contoh peran eksekusi Job
JobExecutionRoleIRSAv2.

Contoh Asosiasi Peran:

```
RoleName: JobExecutionRoleIRSAv2
Base36EncodingOfRoleName: 2eum5fah1jc1kwyjc19ikdhdkdegh1n26vbe
```

Contoh perintah CLI:

```
# setup for the client service account (used by job runner pod)
# emr-containers-sa-spark-client-111122223333-2eum5fah1jc1kwyjc19ikdhdkdegh1n26vbe
aws eks create-pod-identity-association --cluster-name mycluster
  --role-arn arn:aws:iam::111122223333:role/JobExecutionRoleIRSAv2
  --namespace mynamespace --service-account emr-containers-sa-spark-
client-111122223333-2eum5fah1jc1kwyjc19ikdhdkdegh1n26vbe

# driver service account
# emr-containers-sa-spark-driver-111122223333-2eum5fah1jc1kwyjc19ikdhdkdegh1n26vbe

aws eks create-pod-identity-association --cluster-name mycluster
  --role-arn arn:aws:iam::111122223333:role/JobExecutionRoleIRSAv2
  --namespace mynamespace --service-account emr-containers-sa-spark-
driver-111122223333-2eum5fah1jc1kwyjc19ikdhdkdegh1n26vbe

# executor service account
# emr-containers-sa-spark-executor-111122223333-2eum5fah1jc1kwyjc19ikdhdkdegh1n26vbe
aws eks create-pod-identity-association --cluster-name mycluster
  --role-arn arn:aws:iam::111122223333:role/JobExecutionRoleIRSAv2
  --namespace mynamespace --service-account emr-containers-sa-spark-
executor-111122223333-2eum5fah1jc1kwyjc19ikdhdkdegh1n26vbe
```

Setelah Anda menyelesaikan semua langkah yang diperlukan untuk identitas pod EKS, Anda dapat melewati langkah-langkah berikut untuk pengaturan IRSA:

- [Aktifkan Peran IAM untuk Akun Layanan \(IRSA\) di kluster EKS](#)
- [Buat peran eksekusi pekerjaan](#)
- [Perbarui kebijakan kepercayaan dari peran pelaksanaan pekerjaan](#)

Anda dapat langsung melewati langkah berikut: [Berikan pengguna akses ke Amazon EMR](#) di EKS

Hapus Asosiasi Peran

Setiap kali Anda menghapus kluster virtual atau peran eksekusi pekerjaan dan Anda tidak lagi ingin memberikan akses ke EMR ke akun layanannya, Anda harus menghapus asosiasi untuk peran tersebut. Ini karena EKS memungkinkan asosiasi dengan sumber daya yang tidak ada (namespace dan akun layanan). Amazon EMR di EKS merekomendasikan untuk menghapus asosiasi jika namespace dihapus atau peran tidak lagi digunakan, untuk mengosongkan ruang bagi asosiasi lain.

Note

Asosiasi yang tersisa berpotensi memengaruhi kemampuan Anda untuk menskalakan jika Anda tidak menghapusnya, karena EKS memiliki batasan pada jumlah asosiasi yang dapat Anda buat (batas lunak: 1000 asosiasi per cluster). Anda dapat mencantumkan asosiasi identitas pod di namespace tertentu untuk memeriksa apakah Anda memiliki asosiasi yang masih ada yang perlu dibersihkan:

```
aws eks list-pod-identity-associations --cluster-name mycluster --namespace mynamespace
```

Dengan AWS CLI (versi 2.24.0 atau lebih tinggi), jalankan perintah `emr-container` berikut untuk menghapus asosiasi peran EMR:

```
aws emr-containers delete-role-associations \  
  --cluster-name mycluster \  
  --namespace mynamespace \  
  --role-name JobExecutionRoleIRSAv2
```

Secara Otomatis Migrasi IRSA yang Ada ke Identitas Pod

Anda dapat menggunakan alat eksctl untuk memigrasikan Peran IAM yang ada untuk Akun Layanan (IRSA) ke asosiasi identitas pod:

```
eksctl utils migrate-to-pod-identity \  
  --cluster mycluster \  
  --remove-oidc-provider-trust-relationship \  
  --approve
```

Menjalankan perintah tanpa `--approve` bendera hanya akan menampilkan rencana yang mencerminkan langkah migrasi, dan tidak ada migrasi aktual yang akan terjadi.

Pemecahan masalah

Pekerjaan saya gagal dengan `NoClassDefinitionFound` atau `ClassNotFound` Pengecualian untuk Penyedia Kredensial, atau gagal mendapatkan penyedia kredensi.

EKS Pod Identity menggunakan Container Credentials Provider untuk mengambil kredensi yang diperlukan. Jika Anda telah menentukan penyedia kredensi khusus, pastikan itu berfungsi dengan benar. Atau, pastikan Anda menggunakan versi AWS SDK yang benar yang mendukung EKS Pod Identity. Untuk informasi selengkapnya, lihat [Memulai Amazon EKS](#).

Job gagal dengan kesalahan “Failed to Retrieve Credentials Due to [x] Size Limit” yang ditampilkan di log eks-pod-identity-agent.

EMR di EKS membuat Akun Layanan Kubernetes berdasarkan nama peran eksekusi pekerjaan. Jika nama peran terlalu panjang, EKS Auth akan gagal mengambil kredensial karena kombinasi `cluster_name`, `pod_name`, dan `service_account_name` melebihi batas panjang. Identifikasi komponen mana yang paling banyak mengambil ruang dan sesuaikan ukurannya.

Job gagal dengan kesalahan “Failed to Retrieve Credentials xxx” yang ditampilkan di log eks-pod-identity.

Salah satu kemungkinan penyebab masalah ini adalah kluster EKS dikonfigurasi di bawah subnet pribadi tanpa mengonfigurasi PrivateLink cluster dengan benar. Periksa apakah cluster Anda berada di jaringan pribadi dan konfigurasi AWS PrivateLink untuk mengatasi masalah tersebut. Untuk petunjuk terperinci, lihat [Memulai Amazon EKS](#).

Opsi 2: Aktifkan Peran IAM untuk Akun Layanan (IRSA) di cluster EKS

IAM role untuk fitur akun layanan tersedia di Amazon EKS versi 1.14 dan yang lebih baru dan untuk kluster EKS yang diperbarui ke versi 1.13 atau yang lebih baru pada atau setelah 3 September 2019. Untuk menggunakan fitur ini, Anda dapat memperbarui kluster EKS yang ada ke versi 1.14 atau yang lebih baru. Untuk informasi lebih lanjut, lihat [Memperbarui versi Kubernetes kluster Amazon EKS](#).

Jika kluster Anda mendukung IAM role untuk akun layanan, kluster Anda memiliki URL penerbit [Connect OpenID](#) yang terkait dengannya. Anda dapat melihat URL ini di konsol Amazon EKS, atau Anda dapat menggunakan AWS CLI perintah berikut untuk mengambilnya.

Important

Anda harus menggunakan versi terbaru dari AWS CLI untuk menerima output yang tepat dari perintah ini.

```
aws eks describe-cluster --name cluster_name --query "cluster.identity.oidc.issuer" --output text
```

Output yang diharapkan adalah sebagai berikut.

```
https://oidc.eks.<region-code>.amazonaws.com/id/EXAMPLED539D4633E53DE1B716D3041E
```

Untuk menggunakan IAM role untuk akun layanan di kluster Anda, Anda harus membuat penyedia identitas OIDC menggunakan baik [eksctl](#) atau [Konsol Manajemen AWS](#).

Untuk membuat penyedia identitas IAM OIDC untuk cluster Anda dengan eksctl

Periksa versi eksctl Anda dengan perintah berikut. Prosedur ini mengasumsikan bahwa Anda telah menginstal eksctl dan bahwa eksctl versi 0.32.0 atau yang lebih baru.

```
eksctl version
```

Untuk informasi selengkapnya tentang menginstal atau meningkatkan eksctl, lihat [Menginstal atau meningkatkan eksctl](#).

Buat penyedia identitas OIDC Anda untuk kluster Anda dengan perintah berikut. Ganti *cluster_name* dengan nilai milik Anda sendiri.

```
eksctl utils associate-iam-oidc-provider --cluster cluster_name --approve
```

Untuk membuat penyedia identitas IAM OIDC untuk cluster Anda dengan Konsol Manajemen AWS

Ambil URL penerbit OIDC dari deskripsi konsol Amazon EKS klaster Anda, atau gunakan perintah berikut. AWS CLI

Gunakan perintah berikut untuk mengambil URL penerbit OIDC dari AWS CLI.

```
aws eks describe-cluster --name <cluster_name> --query "cluster.identity.oidc.issuer" --output text
```

Gunakan langkah-langkah berikut untuk mengambil URL penerbit OIDC dari konsol Amazon EKS.

1. Buka konsol IAM di <https://console.aws.amazon.com/iam/>.
2. Di panel navigasi, pilih Penyedia Identitas, lalu pilih Buat Penyedia.
 1. Untuk Jenis Penyedia, pilih Pilih jenis penyedia, lalu pilih OpenID Connect.
 2. Untuk Penyedia URL, tempelkan URL penerbit OIDC untuk klaster Anda.
 3. Untuk Audiens, ketik sts.amazonaws.com dan pilih Langkah Selanjutnya.
3. Verifikasi bahwa informasi penyedia sudah benar, kemudian pilih Buat untuk membuat penyedia identitas Anda.

Untuk membuat peran eksekusi tugas

Untuk menjalankan beban kerja di Amazon EMR di EKS, Anda perlu membuat peran IAM.

Kami menyebut peran ini sebagai peran eksekusi tugas dalam dokumentasi ini. Untuk informasi selengkapnya tentang cara membuat peran IAM, lihat [Membuat peran IAM](#) di Panduan pengguna IAM.

Anda juga harus membuat kebijakan IAM yang menentukan izin untuk peran eksekusi pekerjaan dan kemudian melampirkan kebijakan IAM ke peran eksekusi pekerjaan.

Kebijakan berikut untuk peran eksekusi pekerjaan memungkinkan akses ke target sumber daya, Amazon S3, dan CloudWatch Izin ini diperlukan untuk memantau tugas dan log akses. Untuk mengikuti proses yang sama menggunakan AWS CLI:

Buat Peran IAM untuk eksekusi pekerjaan: Mari kita buat peran yang akan digunakan EMR untuk eksekusi pekerjaan. Ini adalah peran, pekerjaan EMR akan diasumsikan ketika mereka berjalan di EKS.

```
cat <<EoF > ~/environment/emr-trust-policy.json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "elasticmapreduce.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EoF

aws iam create-role --role-name EMRContainers-JobExecutionRole --assume-role-policy-document file://~/environment/emr-trust-policy.json
```

Selanjutnya, kita perlu melampirkan kebijakan IAM yang diperlukan ke peran sehingga dapat menulis log ke s3 dan cloudwatch.

```
cat <<EoF > ~/environment/EMRContainers-JobExecutionRole.json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket"
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:PutLogEvents",

```

```

        "logs:CreateLogStream",
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams"
    ],
    "Resource": [
        "arn:aws:logs:*:*:*"
    ]
}
]
}
}
EoF
aws iam put-role-policy --role-name EMRContainers-JobExecutionRole --policy-name
EMR-Containers-Job-Execution --policy-document file://~/environment/EMRContainers-
JobExecutionRole.json

```

Note

Akses harus dicakup dengan tepat, tidak diberikan ke semua objek S3 dalam peran eksekusi pekerjaan.

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket"
      ],
      "Sid": "AllowS3Putobject"
    },
    {
      "Effect": "Allow",
      "Action": [

```

```

        "logs:PutLogEvents",
        "logs:CreateLogStream",
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams"
    ],
    "Resource": [
        "arn:aws:logs:*:*:*"
    ],
    "Sid": "AllowLOGSPutlogevents"
}
]
}

```

Untuk informasi selengkapnya, lihat [Menggunakan peran eksekusi pekerjaan](#), [Mengonfigurasi proses pekerjaan untuk menggunakan log S3](#), dan [Mengonfigurasi proses pekerjaan untuk menggunakan CloudWatch Log](#).

Perbarui kebijakan kepercayaan dari peran eksekusi tugas

Ketika Anda menggunakan IAM Role untuk Akun Layanan (IRSA) untuk menjalankan tugas di namespace Kubernetes, administrator harus membuat hubungan kepercayaan antara peran eksekusi tugas dan identitas akun layanan terkelola EMR. Hubungan kepercayaan dapat dibuat dengan memperbarui kebijakan kepercayaan dari peran pelaksanaan tugas. Perhatikan bahwa akun layanan terkelola EMR secara otomatis dibuat pada pengiriman tugas, dicakup ke namespace di mana tugas dikirimkan.

Jalankan perintah berikut untuk memperbarui kebijakan kepercayaan.

```

aws emr-containers update-role-trust-policy \
  --cluster-name cluster \
  --namespace namespace \
  --role-name iam_role_name_for_job_execution

```

Untuk informasi selengkapnya, lihat [Menggunakan peran eksekusi tugas dengan Amazon EMR di EKS](#).

Important

Operator yang menjalankan perintah di atas harus memiliki izin berikut:
 eks:DescribeCluster, iam:GetRole, iam:UpdateAssumeRolePolicy.

Memberikan pengguna akses ke Amazon EMR di EKS

Untuk setiap tindakan yang Anda lakukan di Amazon EMR di EKS, Anda memerlukan izin IAM yang sesuai untuk tindakan itu. Anda harus membuat kebijakan IAM yang memungkinkan Anda untuk melakukan tindakan Amazon EMR di EKS dan melampirkan kebijakan pada pengguna atau peran IAM yang Anda gunakan.

Topik ini menyediakan langkah-langkah untuk membuat kebijakan baru dan melampirkannya ke pengguna. Hal ini juga mencakup izin dasar yang Anda butuhkan untuk mengatur lingkungan Amazon EMR di EKS Anda. Sebaiknya Anda menyempurnakan izin ke sumber daya tertentu bila memungkinkan berdasarkan kebutuhan bisnis Anda.

Membuat kebijakan IAM baru dan melampirkannya ke pengguna di konsol IAM

Buat kebijakan IAM baru

1. Masuk ke Konsol Manajemen AWS dan buka konsol IAM di <https://console.aws.amazon.com/iam/>.
2. Di panel navigasi kiri konsol IAM, pilih Kebijakan.
3. Pada halaman Kebijakan, pilih Buat Kebijakan.
4. Di jendela Buat Kebijakan, navigasikan ke tab Edit JSON. Buat dokumen kebijakan dengan satu atau lebih pernyataan JSON seperti yang ditunjukkan dalam contoh prosedur berikut ini. Selanjutnya, pilih Tinjau kebijakan.
5. Pada layar Tinjau Kebijakan, masukkan Nama kebijakan Anda, misalnya AmazonEMR0nEKSPo1icy. Masukkan deskripsi opsional, lalu pilih Buat kebijakan.

Lampirkan kebijakan ke pengguna atau peran

1. Masuk ke Konsol Manajemen AWS dan buka konsol IAM di <https://console.aws.amazon.com/iam/>.
2. Di panel navigasi, pilih Kebijakan.
3. Dalam daftar kebijakan, pilih kotak centang di samping kebijakan yang dibuat dalam bagian sebelumnya. Anda bisa memakai menu Filter dan kotak pencarian untuk memfilter daftar kebijakan.
4. Pilih Tindakan kebijakan, lalu pilih Lampirkan.

5. Pilih pengguna atau peran untuk melampirkan kebijakan. Anda bisa memakai menu Filter dan kotak pencarian untuk memfilter daftar entitas prinsipiell. Setelah memilih pengguna atau peran untuk melampirkan kebijakan, pilih Lampirkan kebijakan.

Izin untuk mengelolaklaster virtual

Untuk mengelola klaster virtual di AWS akun Anda, buat kebijakan IAM dengan izin berikut. Izin ini memungkinkan Anda membuat, membuat daftar, mendeskripsikan, dan menghapus klaster virtual di akun Anda AWS .

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringLike": {
          "iam:AWSServiceName": "emr-containers.amazonaws.com"
        }
      },
      "Sid": "AllowIAMCreateservicelinkedrole"
    },
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:CreateVirtualCluster",
        "emr-containers:ListVirtualClusters",
        "emr-containers:DescribeVirtualCluster",
        "emr-containers>DeleteVirtualCluster"
      ],
      "Resource": [
        "*"
      ],
    },
  ],
}
```

```

    "Sid": "AllowEMRCONTAINERSCreatevirtualcluster"
  }
]
}

```

Amazon EMR terintegrasi dengan Amazon EKS cluster access management (CAM), sehingga Anda dapat mengotomatiskan konfigurasi kebijakan AuthN dan AuthZ yang diperlukan untuk menjalankan pekerjaan Amazon EMR Spark di ruang nama cluster Amazon EKS. Untuk melakukannya, Anda harus memiliki izin berikut:

```

{
  "Effect": "Allow",
  "Action": [
    "eks:CreateAccessEntry"
  ],
  "Resource":
  "arn:<AWS_PARTITION>:eks:<AWS_REGION>:<AWS_ACCOUNT_ID>:cluster/<EKS_CLUSTER_NAME>"
},
{
  "Effect": "Allow",
  "Action": [
    "eks:DescribeAccessEntry",
    "eks>DeleteAccessEntry",
    "eks:ListAssociatedAccessPolicies",
    "eks:AssociateAccessPolicy",
    "eks:DisassociateAccessPolicy"
  ],
  "Resource": "arn:<AWS_PARTITION>:eks:<AWS_REGION>:<AWS_ACCOUNT_ID>:access-
entry/<EKS_CLUSTER_NAME>/role/<AWS_ACCOUNT_ID>/AWSServiceRoleForAmazonEMRContainers/*"
}

```

Untuk informasi selengkapnya, lihat [Mengotomatiskan mengaktifkan akses klaster untuk Amazon EMR di EKS](#).

Saat `CreateVirtualCluster` operasi dipanggil untuk pertama kalinya dari AWS akun, Anda juga memerlukan `CreateServiceLinkedRole` izin untuk membuat peran terkait layanan untuk Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [Menggunakan peran terkait layanan untuk Amazon EMR di EKS](#).

Izin untuk mengirimkan tugas

Untuk mengirimkan pekerjaan di kluster virtual di AWS akun Anda, buat kebijakan IAM dengan izin berikut. Izin ini memungkinkan Anda untuk memulai, mendaftar, menjelaskan, dan membatalkan kluster virtual di akun Anda. Anda harus mempertimbangkan menambahkan izin untuk membuat daftar atau menjelaskan kluster virtual, yang memungkinkan Anda untuk memeriksa keadaan kluster virtual sebelum mengirimkan tugas.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:StartJobRun",
        "emr-containers:ListJobRuns",
        "emr-containers:DescribeJobRun",
        "emr-containers:CancelJobRun"
      ],
      "Resource": [
        "*"
      ],
      "Sid": "AllowEMRCONTAINERSStartjobrun"
    }
  ]
}
```

Izin untuk melakukan debug dan pemantauan

Untuk mendapatkan akses ke log yang didorong ke Amazon S3 dan CloudWatch, atau untuk melihat log peristiwa aplikasi di konsol EMR Amazon, buat kebijakan IAM dengan izin berikut. Sebaiknya Anda menyempurnakan izin ke sumber daya tertentu bila memungkinkan berdasarkan kebutuhan bisnis Anda.

⚠ Important

Jika Anda belum membuat bucket Amazon S3, Anda perlu menambahkan izin `s3:CreateBucket` pada pernyataan kebijakan. Jika Anda belum membuat grup log, Anda perlu menambahkan `logs:CreateLogGroup` pada pernyataan kebijakan.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:DescribeJobRun",
        "elasticmapreduce:CreatePersistentAppUI",
        "elasticmapreduce:DescribePersistentAppUI",
        "elasticmapreduce:GetPersistentAppUIPresignedURL"
      ],
      "Resource": [
        "*"
      ],
      "Sid": "AllowEMRCONTAINERSDescribejobrun"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "*"
      ],
      "Sid": "AllowS3GetObject"
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:Get*",
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams"
      ]
    }
  ]
}
```

```

    ],
    "Resource": [
      "*"
    ],
    "Sid": "AllowLOGSGet"
  }
]
}

```

Untuk informasi selengkapnya tentang cara mengonfigurasi job run untuk mendorong log ke Amazon S3 dan CloudWatch, lihat [Mengonfigurasi proses pekerjaan untuk menggunakan log S3](#) dan [Mengonfigurasi pekerjaan yang dijalankan untuk menggunakan Log. CloudWatch](#)

Daftarkan klaster Amazon EKS dengan Amazon EMR

Mendaftarkan klaster Anda adalah langkah terakhir yang diperlukan untuk mengatur Amazon EMR di EKS untuk menjalankan beban kerja.

Gunakan perintah berikut untuk membuat klaster virtual dengan nama pilihan Anda untuk klaster Amazon EKS dan namespace yang Anda atur dalam langkah-langkah sebelumnya.

Note

Setiap cluster virtual harus memiliki nama unik di semua cluster EKS. Jika dua cluster virtual memiliki nama yang sama, proses penyebaran akan gagal bahkan jika dua cluster virtual milik cluster EKS yang berbeda.

```

aws emr-containers create-virtual-cluster \
--name virtual_cluster_name \
--container-provider '{
  "id": "cluster_name",
  "type": "EKS",
  "info": {
    "eksInfo": {
      "namespace": "namespace_name"
    }
  }
}'

```

Atau, Anda dapat membuat file JSON yang mencakup parameter yang diperlukan untuk kluster virtual dan kemudian jalankan perintah `create-virtual-cluster` dengan jalur ke file JSON. Untuk informasi selengkapnya, lihat [Mengelola kluster virtual](#).

Note

Untuk memvalidasi keberhasilan pembuatan kluster virtual, lihat status kluster virtual menggunakan operasi `list-virtual-clusters` atau dengan masuk ke halaman Kluster Virtual di konsol Amazon EMR.

Kirim pekerjaan yang dijalankan dengan **StartJobRun**

Untuk mengirimkan pekerjaan yang dijalankan dengan file JSON dengan parameter tertentu

1. Buat file `start-job-run-request.json` dan tentukan parameter yang diperlukan untuk menjalankan tugas Anda, seperti contoh yang ditunjukkan file JSON berikut. Untuk informasi tentang parameter, lihat [Pilihan untuk mengonfigurasi tugas berjalan](#).

```
{
  "name": "myjob",
  "virtualClusterId": "123456",
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "emr-6.2.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "entryPoint_location",
      "entryPointArguments": ["argument1", "argument2", ...],
      "sparkSubmitParameters": "--class <main_class> --conf
spark.executor.instances=2 --conf spark.executor.memory=2G --conf
spark.executor.cores=2 --conf spark.driver.cores=1"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.driver.memory": "2G"
        }
      }
    ]
  }
}
```

```

    ],
    "monitoringConfiguration": {
      "persistentAppUI": "ENABLED",
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "my_log_group",
        "logStreamNamePrefix": "log_stream_prefix"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://my_s3_log_location"
      }
    }
  }
}

```

- Gunakan `start-job-run` perintah dengan path ke `start-job-run-request.json` file yang disimpan secara lokal.

```

aws emr-containers start-job-run \
--cli-input-json file:///./start-job-run-request.json

```

Untuk memulai tugas berjalan menggunakan perintah **start-job-run**

- Pasokan semua parameter yang ditentukan dalam perintah `StartJobRun`, seperti yang ditunjukkan contoh berikut.

```

aws emr-containers start-job-run \
--virtual-cluster-id 123456 \
--name myjob \
--execution-role-arn execution-role-arn \
--release-label emr-6.2.0-latest \
--job-driver '{"sparkSubmitJobDriver": {"entryPoint": "entryPoint_location",
"entryPointArguments": ["argument1", "argument2", ...], "sparkSubmitParameters":
"--class <main_class> --conf spark.executor.instances=2 --conf
spark.executor.memory=2G --conf spark.executor.cores=2 --conf
spark.driver.cores=1"}}' \
--configuration-overrides '{"applicationConfiguration": [{"classification":
"spark-defaults", "properties": {"spark.driver.memory": "2G"}}],
"monitoringConfiguration": {"cloudWatchMonitoringConfiguration":
{"logGroupName": "log_group_name", "logStreamNamePrefix": "log_stream_prefix"},
"persistentAppUI": "ENABLED", "s3MonitoringConfiguration": {"logUri":
"s3://my_s3_log_location" } }'

```

- Untuk Spark SQL, sediakan semua parameter yang ditentukan dalam StartJobRun perintah, seperti yang ditunjukkan contoh berikut.

```
aws emr-containers start-job-run \
--virtual-cluster-id 123456 \
--name myjob \
--execution-role-arn execution-role-arn \
--release-label emr-6.7.0-latest \
--job-driver '{"sparkSqlJobDriver": {"entryPoint": "entryPoint_location",
"sparkSqlParameters": "--conf spark.executor.instances=2 --conf
spark.executor.memory=2G --conf spark.executor.cores=2 --conf
spark.driver.cores=1"}}' \
--configuration-overrides '{"applicationConfiguration": [{"classification":
"spark-defaults", "properties": {"spark.driver.memory": "2G"}}],
"monitoringConfiguration": {"cloudWatchMonitoringConfiguration":
{"logGroupName": "log_group_name", "logStreamNamePrefix": "log_stream_prefix"},
"persistentAppUI": "ENABLED", "s3MonitoringConfiguration": {"logUri":
"s3://my_s3_log_location" }}}
```

Menggunakan klasifikasi pengirim pekerjaan

Ikhtisar

EMR Amazon StartJobRun berdasarkan permintaan EKS membuat pod pengirim pekerjaan (juga dikenal sebagai pod job-runner) untuk menelurkan driver Spark. Anda dapat menggunakan `emr-job-submitter` klasifikasi untuk mengonfigurasi pemilih node, menambahkan toleransi, menyesuaikan logging, dan membuat modifikasi lain pada pod pengirim pekerjaan.

Pengaturan berikut tersedia di bawah `emr-job-submitter` klasifikasi:

`jobsubmitter.node.selector.[selectorKey]`

Menambahkan ke pemilih node dari pod pengirim pekerjaan, dengan kunci `selectorKey` dan nilai sebagai nilai konfigurasi. Misalnya, Anda dapat mengatur `jobsubmitter.node.selector.identifier` ke `myIdentifier` dan pod pengirim pekerjaan akan memiliki pemilih node dengan kunci `identifier` dan nilai `myIdentifier`. Ini dapat digunakan untuk menentukan node mana pod pengirim pekerjaan dapat ditempatkan. Untuk menambahkan beberapa kunci pemilih node, atur beberapa konfigurasi dengan awalan ini.

jobsubmitter.label.[*labelKey*]

Menambahkan label pod pengirim pekerjaan, dengan kunci *labelKey* dan nilai sebagai nilai konfigurasi. Untuk menambahkan beberapa label, atur beberapa konfigurasi dengan awalan ini.

jobsubmitter.annotation.[*annotationKey*]

Menambahkan anotasi pod pengirim pekerjaan, dengan kunci *annotationKey* dan nilai sebagai nilai konfigurasi. Untuk menambahkan beberapa anotasi, atur beberapa konfigurasi dengan awalan ini.

jobsubmitter.node.toleration.[*tolerationKey*]

Menambahkan [toleransi ke pod pengirim](#) pekerjaan. Secara default tidak ada toleransi yang ditambahkan ke pod. Kunci toleransi akan menjadi *tolerationKey* dan nilai toleransi akan menjadi nilai konfigurasi. Jika nilai konfigurasi diatur ke string yang tidak kosong, operator akan menjadi `Equals`. Jika nilai konfigurasi diatur ke "", maka operator akan `Exists`.

jobsubmitter.node.toleration.[*tolerationKey*].[*effect*]

Menambahkan efek toleransi ke awalan *tolerationKey*. Bidang ini diperlukan saat menambahkan toleransi. Nilai yang diizinkan untuk bidang efek adalah `NoExecute`, `NoSchedule`, dan `PreferNoSchedule`.

jobsubmitter.node.toleration.[*tolerationKey*].[*tolerationSeconds*]

Menambahkan `TolerationSeconds` ke awalan. *tolerationKey* Bidang opsional. Hanya berlaku ketika efeknya `NoExecute`.

jobsubmitter.scheduler.name

Menetapkan `schedulerName` kustom untuk pod pengirim pekerjaan.

jobsubmitter.logging

Mengaktifkan atau menonaktifkan logging pada pod pengirim pekerjaan. Ketika ini disetel ke `DISABLED` container logging akan dihapus dari pod pengirim pekerjaan, yang akan menonaktifkan setiap logging untuk pod ini yang ditentukan dalam `monitoringConfiguration`, seperti `s3MonitoringConfiguration` or `cloudWatchMonitoringConfiguration`. Ketika pengaturan ini tidak disetel atau disetel ke nilai lain, logging pada pod pengirim pekerjaan diaktifkan.

jobsubmitter.logging.image

Menetapkan gambar kustom yang akan digunakan untuk penampung logging pada pod pengirim pekerjaan.

jobsubmitter.logging.request.cores

Menetapkan nilai kustom untuk jumlah CPUs, dalam unit CPU, untuk penampung logging pada pod pengirim pekerjaan. Secara default, ini diatur ke 100m.

jobsubmitter.logging.request.memory

Menetapkan nilai kustom untuk jumlah memori, dalam byte, untuk penampung logging pada pod pengirim pekerjaan. Secara default, ini diatur ke 200Mi. Mebibyte adalah satuan ukuran yang mirip dengan megabyte.

jobsubmitter.container.image

Menetapkan image kustom untuk container pod job submitter. `job-runner`

jobsubmitter.container.image.pullPolicy

Menetapkan [imagePullPolicy](#) container untuk job submitter pod.

Kami merekomendasikan untuk menempatkan pod pengirim pekerjaan pada Instans Sesuai Permintaan. Menempatkan pod pengirim pekerjaan pada instans Spot dapat mengakibatkan kegagalan pekerjaan jika instance tempat pod pengirim pekerjaan berjalan tunduk pada interupsi Instance Spot. Kamu juga bisa [menempatkan job submitter pod dalam Availability Zone tunggal atau menggunakan label Kubernetes yang diterapkan](#) pada node.

Contoh klasifikasi pengirim pekerjaan

Dalam bagian ini

- [StartJobRunpermintaan dengan penempatan node On-Demand untuk pod pengirim pekerjaan](#)
- [StartJobRunrequest dengan penempatan node single-AZ dan penempatan tipe instans Amazon EC2 untuk pod pengirim pekerjaan](#)
- [StartJobRunpermintaan dengan label, anotasi, dan penjadwal khusus untuk pod pengirim pekerjaan](#)
- [StartJobRunpermintaan dengan toleransi yang diterapkan pada pod pengirim pekerjaan dengan kuncidedicated, nilai, efek graviton_machinesNoExecute, dan 60 detik tolerationSeconds](#)
- [StartJobRunpermintaan dengan logging dinonaktifkan untuk pod pengirim pekerjaan](#)
- [StartJobRunpermintaan dengan image container logging kustom, CPU, dan memori untuk pod pengirim pekerjaan](#)
- [StartJobRunpermintaan dengan gambar kontainer pengirim pekerjaan khusus dan kebijakan tarik](#)

StartJobRun permintaan dengan penempatan node On-Demand untuk pod pengirim pekerjaan

```
cat >spark-python-in-s3-nodeselector-job-submitter.json << EOF
{
  "name": "spark-python-in-s3-nodeselector",
  "virtualClusterId": "virtual-cluster-id",
  "executionRoleArn": "execution-role-arn",
  "releaseLabel": "emr-6.11.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "s3://S3-prefix/trip-count.py",
      "sparkSubmitParameters": "--conf spark.driver.cores=5 --conf
spark.executor.memory=20G --conf spark.driver.memory=15G --conf
spark.executor.cores=6"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.dynamicAllocation.enabled":"false"
        }
      },
      {
        "classification": "emr-job-submitter",
        "properties": {
          "jobsubmitter.node.selector.eks.amazonaws.com/capacityType": "ON_DEMAND"
        }
      }
    ]
  },
  "monitoringConfiguration": {
    "cloudWatchMonitoringConfiguration": {
      "logGroupName": "/emr-containers/jobs",
      "logStreamNamePrefix": "demo"
    },
    "s3MonitoringConfiguration": {
      "logUri": "s3://joblogs"
    }
  }
}
EOF
```

```
aws emr-containers start-job-run --cli-input-json file:///spark-python-in-s3-
nodeselector-job-submitter.json
```

StartJobRunrequest dengan penempatan node single-AZ dan penempatan tipe instans Amazon EC2 untuk pod pengirim pekerjaan

```
"configurationOverrides": {
  "applicationConfiguration": [
    {
      "classification": "emr-job-submitter",
      "properties": {
        "jobsubmitter.node.selector.topology.kubernetes.io/zone": "Availability Zone",
        "jobsubmitter.node.selector.node.kubernetes.io/instance-type": "m5.4xlarge"
      }
    }
  ]
}
```

StartJobRun permintaan dengan label, anotasi, dan penjadwal khusus untuk pod pengirim pekerjaan

```
"configurationOverrides": {
  "applicationConfiguration": [
    {
      "classification": "emr-job-submitter",
      "properties": {
        "jobsubmitter.label.label1": "value1",
        "jobsubmitter.label.label2": "value2",
        "jobsubmitter.annotation.ann1": "value1",
        "jobsubmitter.annotation.ann2": "value2",
        "jobsubmitter.scheduler.name": "custom-scheduler"
      }
    }
  ]
}
```

StartJobRun permintaan dengan toleransi yang diterapkan pada pod pengirim pekerjaan dengan kunci **dedicated**, nilai, efek **graviton_machinesNoExecute**, dan 60 detik **tolerationSeconds**

```
"configurationOverrides": {
  "applicationConfiguration": [
```

```

{
  "classification": "emr-job-submitter",
  "properties": {
    "jobsubmitter.node.toleration.dedicated": "graviton_machines",
    "jobsubmitter.node.toleration.dedicated.effect": "NoExecute",
    "jobsubmitter.node.toleration.dedicated.tolerationSeconds": "60"
  }
}
]
}

```

StartJobRun permintaan dengan logging dinonaktifkan untuk pod pengirim pekerjaan

```

"configurationOverrides": {
  "applicationConfiguration": [
    {
      "classification": "emr-job-submitter",
      "properties": {
        "jobsubmitter.logging": "DISABLED"
      }
    }
  ],
  "monitoringConfiguration": {
    "cloudWatchMonitoringConfiguration": {
      "logGroupName": "/emr-containers/jobs",
      "logStreamNamePrefix": "demo"
    },
    "s3MonitoringConfiguration": {
      "logUri": "s3://joblogs"
    }
  }
}
}

```

StartJobRun permintaan dengan image container logging kustom, CPU, dan memori untuk pod pengirim pekerjaan

```

"configurationOverrides": {
  "applicationConfiguration": [
    {
      "classification": "emr-job-submitter",
      "properties": {
        "jobsubmitter.logging.image": "YOUR_ECR_IMAGE_URL",
        "jobsubmitter.logging.request.memory": "200Mi",

```

```

        "jobsubmitter.logging.request.cores": "0.5"
    }
}
],
"monitoringConfiguration": {
    "cloudWatchMonitoringConfiguration": {
        "logGroupName": "/emr-containers/jobs",
        "logStreamNamePrefix": "demo"
    },
    "s3MonitoringConfiguration": {
        "logUri": "s3://joblogs"
    }
}
}
}

```

StartJobRun permintaan dengan gambar kontainer pengirim pekerjaan khusus dan kebijakan tarik

```

"configurationOverrides": {
    "applicationConfiguration": [
        {
            "classification": "emr-job-submitter",
            "properties": {
                "jobsubmitter.container.image": "123456789012.dkr.ecr.us-west-2.amazonaws.com/emr6.11_custom_repo",
                "jobsubmitter.container.image.pullPolicy": "kubernetes pull policy"
            }
        }
    ]
}

```

Menggunakan klasifikasi default penampung EMR Amazon

Ikhtisar

Pengaturan berikut tersedia di bawah `emr-containers-defaults` klasifikasi:

job-start-timeout

Secara default, pekerjaan akan habis jika tidak dapat dimulai dan menunggu di `SUBMITTED` negara bagian selama 15 menit. Konfigurasi ini mengubah jumlah detik untuk menunggu sebelum waktu kerja habis.

executor.logging

Mengaktifkan atau menonaktifkan logging pada pod eksekutor. Ketika ini disetel ke DISABLED container logging akan dihapus dari pod eksekutor, yang akan menonaktifkan setiap logging untuk pod ini yang ditentukan dalam `monitoringConfiguration`, seperti `s3MonitoringConfiguration` atau `cloudWatchMonitoringConfiguration`. Ketika pengaturan ini tidak disetel atau disetel ke nilai lain, logging pada pod pelaksana diaktifkan.

logging.image

Menetapkan gambar kustom yang akan digunakan untuk penampung logging pada driver dan pod eksekutor.

logging.request.cores

Menetapkan nilai kustom untuk jumlah CPU, dalam unit CPU, untuk container logging pada driver dan executor pod. Secara default, ini tidak diatur.

logging.request.memory

Menetapkan nilai kustom untuk jumlah memori, dalam byte, untuk container logging pada driver dan executor pod. Secara default, ini diatur ke 512Mi. Mebibyte adalah satuan ukuran yang mirip dengan megabyte.

Contoh klasifikasi pengirim pekerjaan

Dalam bagian ini

- [StartJobRunpermintaan dengan batas waktu kerja khusus](#)
- [StartJobRunpermintaan dengan logging dinonaktifkan untuk pod pelaksana](#)
- [StartJobRunpermintaan dengan image container logging kustom, CPU, dan memori untuk driver dan pod pelaksana](#)

StartJobRunpermintaan dengan batas waktu kerja khusus

```
{
  "name": "spark-python",
  "virtualClusterId": "virtual-cluster-id",
  "executionRoleArn": "execution-role-arn",
  "releaseLabel": "emr-6.11.0-latest",
  "jobDriver": {
```

```

    "sparkSubmitJobDriver": {
      "entryPoint": "s3://S3-prefix/trip-count.py"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "emr-containers-defaults",
        "properties": {
          "job-start-timeout": "1800"
        }
      }
    ],
    "monitoringConfiguration": {
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "/emr-containers/jobs",
        "logStreamNamePrefix": "demo"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://joblogs"
      }
    }
  }
}

```

StartJobRun permintaan dengan logging dinonaktifkan untuk pod pelaksana

```

"configurationOverrides": {
  "applicationConfiguration": [
    {
      "classification": "emr-containers-defaults",
      "properties": {
        "executor.logging": "DISABLED"
      }
    }
  ],
  "monitoringConfiguration": {
    "cloudWatchMonitoringConfiguration": {
      "logGroupName": "/emr-containers/jobs",
      "logStreamNamePrefix": "demo"
    },
    "s3MonitoringConfiguration": {
      "logUri": "s3://joblogs"
    }
  }
}

```

```

    }
  }
}

```

StartJobRun permintaan dengan image container logging kustom, CPU, dan memori untuk driver dan pod pelaksana

```

"configurationOverrides": {
  "applicationConfiguration": [
    {
      "classification": "emr-containers-defaults",
      "properties": {
        "logging.image": "YOUR_ECR_IMAGE_URL",
        "logging.request.memory": "200Mi",
        "logging.request.cores": "0.5"
      }
    }
  ],
  "monitoringConfiguration": {
    "cloudWatchMonitoringConfiguration": {
      "logGroupName": "/emr-containers/jobs",
      "logStreamNamePrefix": "demo"
    },
    "s3MonitoringConfiguration": {
      "logUri": "s3://joblogs"
    }
  }
}

```

Note

Jika container logging Fluentd mengalami error out-of-memory (OOM), tingkatkan nilainya. `logging.request.memory` Misalnya, atur untuk 1Gi mengalokasikan lebih banyak memori ke wadah logging dan mencegah masalah OOM.

Menjalankan pekerjaan Spark dengan operator Spark

Amazon EMR merilis 6.10.0 dan lebih tinggi mendukung operator Kubernetes untuk Apache Spark, atau operator Spark, sebagai model pengiriman pekerjaan untuk Amazon EMR di EKS. Dengan operator Spark, Anda dapat menerapkan dan mengelola aplikasi Spark dengan runtime rilis Amazon

EMR di kluster Amazon EKS Anda sendiri. Setelah Anda menerapkan operator Spark di cluster Amazon EKS Anda, Anda dapat langsung mengirimkan aplikasi Spark dengan operator. Operator mengelola siklus hidup aplikasi Spark.

Note

Amazon EMR menghitung harga di Amazon EKS berdasarkan vCPU dan konsumsi memori. Perhitungan ini berlaku untuk pod driver dan executor. Perhitungan ini dimulai dari saat Anda mengunduh image aplikasi Amazon EMR hingga pod Amazon EKS berakhir dan dibulatkan ke detik terdekat.

Topik

- [Menyiapkan operator Spark untuk Amazon EMR di EKS](#)
- [Memulai dengan operator Spark untuk Amazon EMR di EKS](#)
- [Gunakan penskalaan otomatis vertikal dengan operator Spark untuk Amazon EMR di EKS](#)
- [Menghapus instalasi operator Spark untuk Amazon EMR di EKS](#)
- [Menggunakan konfigurasi pemantauan untuk memantau operator Spark Kubernetes dan pekerjaan Spark](#)
- [Keamanan dan operator Spark dengan Amazon EMR di EKS](#)

Menyiapkan operator Spark untuk Amazon EMR di EKS

Selesaikan tugas-tugas berikut untuk menyiapkan sebelum Anda menginstal operator Spark di Amazon EKS. Jika Anda sudah mendaftar untuk Amazon Web Services (AWS) dan telah menggunakan Amazon EKS, Anda hampir siap untuk menggunakan Amazon EMR di EKS. Selesaikan tugas-tugas berikut untuk menyiapkan operator Spark di Amazon EKS. Jika Anda telah menyelesaikan salah satu prasyarat, Anda dapat melewatinya dan melanjutkan ke yang berikutnya.

- [Instal atau perbarui ke versi terbaru AWS CLI](#) - Jika Anda sudah menginstal AWS CLI, konfirmasi bahwa Anda memiliki versi terbaru.
- [Siapkan kubectl dan eksctl - eksctl](#) adalah alat baris perintah yang Anda gunakan untuk berkomunikasi dengan Amazon EKS.
- [Instal Helm](#) — Manajer paket Helm untuk Kubernetes membantu Anda menginstal dan mengelola aplikasi di kluster Kubernetes Anda.

- [Memulai Amazon EKS — eksctl](#) — Ikuti langkah-langkah untuk membuat cluster Kubernetes baru dengan node di Amazon EKS.
- [Pilih URI gambar dasar EMR Amazon](#) (rilis 6.10.0 atau lebih tinggi) — operator Spark didukung dengan rilis Amazon EMR 6.10.0 dan yang lebih tinggi.

Memulai dengan operator Spark untuk Amazon EMR di EKS

Topik ini membantu Anda mulai menggunakan operator Spark di Amazon EKS dengan menerapkan aplikasi Spark dan aplikasi Schedule Spark.

Instal operator Spark

Gunakan langkah-langkah berikut untuk menginstal operator Kubernetes untuk Apache Spark.

1. Jika Anda belum melakukannya, selesaikan langkah-langkahnya [Menyiapkan operator Spark untuk Amazon EMR di EKS](#).
2. Otentikasi klien Helm Anda ke registri Amazon ECR. Dalam perintah berikut, ganti *region-id* nilai dengan pilihan Anda Wilayah AWS, dan *ECR-registry-account* nilai yang sesuai untuk Wilayah dari [Akun registri Amazon ECR berdasarkan Wilayah](#) halaman.

```
aws ecr get-login-password \  
--region region-id | helm registry login \  
--username AWS \  
--password-stdin ECR-registry-account.dkr.ecr.region-id.amazonaws.com
```

3. Instal operator Spark dengan perintah berikut.

Untuk `--version` parameter bagan Helm, gunakan label rilis Amazon EMR Anda dengan awalan dan `emr-` akhiran tanggal dihapus. Misalnya, dengan `emr-6.12.0-java17-latest` rilis, tentukan `6.12.0-java17`. Contoh dalam perintah berikut menggunakan `emr-7.13.0-latest` rilis, sehingga menentukan `7.13.0` bagan Helm. `--version`

```
helm install spark-operator-demo \  
oci://895885662937.dkr.ecr.region-id.amazonaws.com/spark-operator \  
--set emrContainers.awsRegion=region-id \  
--version 7.13.0 \  
--namespace spark-operator \  
--create-namespace
```

Secara default, perintah membuat akun layanan `emr-containers-sa-spark-operator` untuk operator Spark. Untuk menggunakan akun layanan yang berbeda, berikan argumennya `serviceAccounts.sparkoperator.name`. Contoh:

```
--set serviceAccounts.sparkoperator.name my-service-account-for-spark-operator
```

Jika Anda ingin [mengggunakan penskalaan otomatis vertikal dengan operator Spark](#), tambahkan baris berikut ke perintah instalasi untuk mengizinkan webhook untuk operator:

```
--set webhook.enable=true
```

4. Verifikasi bahwa Anda menginstal bagan Helm dengan `helm list` perintah:

```
helm list --namespace spark-operator -o yaml
```

`helm list` Perintah harus mengembalikan informasi rilis bagan Helm yang baru Anda gunakan:

```
app_version: v1beta2-1.3.8-3.1.1
chart: spark-operator-7.13.0
name: spark-operator-demo
namespace: spark-operator
revision: "1"
status: deployed
updated: 2023-03-14 18:20:02.721638196 +0000 UTC
```

5. Instalasi lengkap dengan opsi tambahan apa pun yang Anda butuhkan. Untuk informasi lebih lanjut, lihat [spark-on-k8s-operator](#) dokumentasi di GitHub.

Jalankan aplikasi Spark

Operator Spark didukung dengan Amazon EMR 6.10.0 atau lebih tinggi. Ketika Anda menginstal operator Spark, itu membuat akun layanan `emr-containers-sa-spark` untuk menjalankan aplikasi Spark secara default. Gunakan langkah-langkah berikut untuk menjalankan aplikasi Spark dengan operator Spark di Amazon EMR di EKS 6.10.0 atau lebih tinggi.

1. Sebelum Anda dapat menjalankan aplikasi Spark dengan operator Spark, selesaikan langkah-langkah di [Menyiapkan operator Spark untuk Amazon EMR di EKS](#) dan [Instal operator Spark](#)
2. Buat file SparkApplication definisi `spark-pi.yaml` dengan isi contoh berikut:

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: spark-pi
  namespace: spark-operator
spec:
  type: Scala
  mode: cluster
  image: "895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.10.0:latest"
  imagePullPolicy: Always
  mainClass: org.apache.spark.examples.SparkPi
  mainApplicationFile: "local:///usr/lib/spark/examples/jars/spark-examples.jar"
  sparkVersion: "3.3.1"
  restartPolicy:
    type: Never
  volumes:
    - name: "test-volume"
      hostPath:
        path: "/tmp"
        type: Directory
  driver:
    cores: 1
    coreLimit: "1200m"
    memory: "512m"
    labels:
      version: 3.3.1
    serviceAccount: emr-containers-sa-spark
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
  executor:
    cores: 1
    instances: 1
    memory: "512m"
    labels:
      version: 3.3.1
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
```

3. Sekarang, kirimkan aplikasi Spark dengan perintah berikut. Ini juga akan membuat SparkApplication objek bernama `spark-pi`:

```
kubectl apply -f spark-pi.yaml
```

4. Periksa peristiwa untuk SparkApplication objek dengan perintah berikut:

```
kubectl describe sparkapplication spark-pi --namespace spark-operator
```

Untuk informasi selengkapnya tentang mengirimkan aplikasi ke Spark melalui operator Spark, lihat [Menggunakan a SparkApplication](#) dalam dokumentasi pada. spark-on-k8s-operator GitHub

Gunakan Amazon S3 untuk penyimpanan

Untuk menggunakan Amazon S3 sebagai opsi penyimpanan file Anda, tambahkan konfigurasi berikut ke file YAMM Anda.

```
hadoopConf:
# EMRFS filesystem
  fs.s3.customAWSCredentialsProvider:
com.amazonaws.auth.WebIdentityTokenCredentialsProvider
  fs.s3.impl: com.amazon.ws.emr.hadoop.fs.EmrFileSystem
  fs.AbstractFileSystem.s3.impl: org.apache.hadoop.fs.s3.EMRFSDelegate
  fs.s3.buffer.dir: /mnt/s3
  fs.s3.getObject.initialSocketTimeoutMilliseconds: "2000"
  mapreduce.fileoutputcommitter.algorithm.version.emr_internal_use_only.EmrFileSystem:
"2"
  mapreduce.fileoutputcommitter.cleanup-
failures.ignored.emr_internal_use_only.EmrFileSystem: "true"
sparkConf:
# Required for EMR Runtime
  spark.driver.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/hadoop-
aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/share/aws/
emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/security/conf:/
usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-glue-datacatalog-
spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-serde.jar:/usr/share/aws/
sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/hadoop/extrajars/*
  spark.driver.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/lib/
native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/native
  spark.executor.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/hadoop-
aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/share/aws/
emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/security/conf:/
```

```
usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-glue-datacatalog-  
spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-serde.jar:/usr/share/aws/  
sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/hadoop/extrajars/*  
spark.executor.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/lib/  
native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/native
```

Jika Anda menggunakan Amazon EMR rilis 7.2.0 dan yang lebih tinggi, konfigurasi disertakan secara default. Dalam hal ini, Anda dapat mengatur jalur file `s3://<bucket_name>/<file_path>` alih-alih `local://<file_path>` di file YAMM aplikasi Spark.

Kemudian kirimkan aplikasi Spark seperti biasa.

Gunakan penskalaan otomatis vertikal dengan operator Spark untuk Amazon EMR di EKS

Dimulai dengan Amazon EMR 7.0, Anda dapat menggunakan Amazon EMR pada penskalaan otomatis vertikal EKS untuk menyederhanakan manajemen sumber daya. Ini secara otomatis menyetel memori dan sumber daya CPU untuk beradaptasi dengan kebutuhan beban kerja yang Anda sediakan untuk aplikasi Amazon EMR Spark. Untuk informasi selengkapnya, lihat [Menggunakan penskalaan otomatis vertikal dengan pekerjaan Amazon EMR Spark](#).

Bagian ini menjelaskan cara mengkonfigurasi operator Spark untuk menggunakan penskalaan otomatis vertikal.

Prasyarat

Sebelum Anda mengonfigurasi pemantauan, pastikan untuk menyelesaikan tugas persiapan berikut:

- Selesaikan langkah-langkah dalam [Menyiapkan operator Spark untuk Amazon EMR di EKS](#).
- (opsional) Jika sebelumnya Anda menginstal versi operator Spark yang lebih lama, hapus `SparkApplication/ScheduledSparkApplication` CRD.

```
kubectl delete crd sparkApplication  
kubectl delete crd scheduledSparkApplication
```

- Selesaikan langkah-langkah dalam [Instal operator Spark](#). Pada langkah 3, tambahkan baris berikut ke perintah instalasi untuk mengizinkan webhook untuk operator:

```
--set webhook.enable=true
```

- Selesaikan langkah-langkah dalam [Menyiapkan penskalaan otomatis vertikal untuk Amazon EMR di EKS](#).
- Berikan akses ke file di lokasi Amazon S3 Anda:
 1. Beri anotasi akun layanan driver dan operator Anda dengan JobExecutionRole yang memiliki izin S3.

```
kubectl annotate serviceaccount -n spark-operator emr-containers-sa-spark
eks.amazonaws.com/role-arn=JobExecutionRole
kubectl annotate serviceaccount -n spark-operator emr-containers-sa-spark-
operator eks.amazonaws.com/role-arn=JobExecutionRole
```

2. Perbarui kebijakan kepercayaan peran eksekusi pekerjaan Anda di namespace tersebut.

```
aws emr-containers update-role-trust-policy \
--cluster-name cluster \
--namespace ${Namespace}\
--role-name iam_role_name_for_job_execution
```

3. Edit kebijakan kepercayaan peran IAM dari peran eksekusi pekerjaan Anda dan perbarui serviceaccount dari emr-containers-sa-spark-**-*-xxxx* ke emr-containers-sa-***.

```
{
  "Effect": "Allow",
  "Principal": {
    "Federated": "OIDC-provider"
  },
  "Action": "sts:AssumeRoleWithWebIdentity",
  "Condition": {
    "StringLike": {
      "OIDC": "system:serviceaccount:${Namespace}:emr-containers-sa-*"
    }
  }
}
```

4. Jika Anda menggunakan Amazon S3 sebagai penyimpanan file, tambahkan default berikut ke file yaml Anda.

```
hadoopConf:
# EMRFS filesystem
```

```
fs.s3.customAWSCredentialsProvider:
com.amazonaws.auth.WebIdentityTokenCredentialsProvider
fs.s3.impl: com.amazon.ws.emr.hadoop.fs.EmrFileSystem
fs.AbstractFileSystem.s3.impl: org.apache.hadoop.fs.s3.EMRFSDelegate
fs.s3.buffer.dir: /mnt/s3
fs.s3.getObject.initialSocketTimeoutMilliseconds: "2000"

mapreduce.fileoutputcommitter.algorithm.version.emr_internal_use_only.EmrFileSystem:
"2"
mapreduce.fileoutputcommitter.cleanup-
failures.ignored.emr_internal_use_only.EmrFileSystem: "true"
sparkConf:
# Required for EMR Runtime
spark.driver.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/hadoop-
aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/share/
aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/
security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-
glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-
serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/
hadoop/extrajars/*
spark.driver.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/
lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/
native
spark.executor.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/
hadoop-aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/
share/aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/
security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-
glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-
serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/
hadoop/extrajars/*
spark.executor.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-
lzo/lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/
native
```

Jalankan pekerjaan dengan penskalaan otomatis vertikal pada operator Spark

Sebelum Anda dapat menjalankan aplikasi Spark dengan operator Spark, Anda harus menyelesaikan langkah-langkahnya. [Prasyarat](#)

Untuk menggunakan penskalaan otomatis vertikal dengan operator Spark, tambahkan konfigurasi berikut ke driver untuk spesifikasi Aplikasi Spark Anda untuk mengaktifkan penskalaan otomatis vertikal:

```
dynamicSizing:  
  mode: Off  
  signature: "my-signature"
```

Konfigurasi ini memungkinkan penskalaan otomatis vertikal dan merupakan konfigurasi tanda tangan wajib yang memungkinkan Anda memilih tanda tangan untuk pekerjaan Anda.

Untuk informasi selengkapnya tentang konfigurasi dan nilai parameter, lihat [Mengonfigurasi penskalaan otomatis vertikal untuk Amazon EMR](#) di EKS. Secara default, pekerjaan Anda dikirimkan dalam mode Monitoring-Only Off dari penskalaan otomatis vertikal. Status pemantauan ini memungkinkan Anda menghitung dan melihat rekomendasi sumber daya tanpa melakukan penskalaan otomatis. Untuk informasi selengkapnya, lihat Mode [penskalaan otomatis vertikal](#).

Berikut ini adalah file SparkApplication definisi sampel bernama `spark-pi.yaml` dengan konfigurasi yang diperlukan untuk menggunakan penskalaan otomatis vertikal.

```
apiVersion: "sparkoperator.k8s.io/v1beta2"  
kind: SparkApplication  
metadata:  
  name: spark-pi  
  namespace: spark-operator  
spec:  
  type: Scala  
  mode: cluster  
  image: "895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-7.13.0:latest"  
  imagePullPolicy: Always  
  mainClass: org.apache.spark.examples.SparkPi  
  mainApplicationFile: "local:///usr/lib/spark/examples/jars/spark-examples.jar"  
  sparkVersion: "3.4.1"  
  dynamicSizing:  
    mode: Off  
    signature: "my-signature"  
  restartPolicy:  
    type: Never  
  volumes:  
    - name: "test-volume"  
      hostPath:  
        path: "/tmp"  
        type: Directory  
  driver:  
    cores: 1  
    coreLimit: "1200m"
```

```

memory: "512m"
labels:
  version: 3.4.1
serviceAccount: emr-containers-sa-spark
volumeMounts:
  - name: "test-volume"
    mountPath: "/tmp"
executor:
  cores: 1
  instances: 1
  memory: "512m"
  labels:
    version: 3.4.1
  volumeMounts:
    - name: "test-volume"
      mountPath: "/tmp"

```

Sekarang, kirimkan aplikasi Spark dengan perintah berikut. Ini juga akan membuat `SparkApplication` objek bernama `spark-pi`:

```
kubectl apply -f spark-pi.yaml
```

Untuk informasi selengkapnya tentang mengirimkan aplikasi ke Spark melalui operator Spark, lihat [Menggunakan a SparkApplication](#) dalam dokumentasi pada `spark-on-k8s-operator` GitHub

Memverifikasi fungsionalitas penskalaan otomatis vertikal

Untuk memverifikasi bahwa penskalaan otomatis vertikal berfungsi dengan benar untuk pekerjaan yang dikirimkan, gunakan `kubectl` untuk mendapatkan sumber daya `verticalpodautoscaler` kustom dan melihat rekomendasi penskalaan Anda.

```
kubectl get verticalpodautoscalers --all-namespaces \
-l=emr-containers.amazonaws.com/dynamic.sizing.signature=my-signature
```

Output dari kueri ini harus menyerupai yang berikut:

NAMESPACE	NAME	MODE		
CPU	MEM	PROVIDED	AGE	
spark-operator	ds-p73j6mkosvc4xeb3gr7x4xol2bfcw5evqimzqojrlysvj3giozuq-vpa	Off		
	580026651	True	15m	

Jika output Anda tidak terlihat serupa atau berisi kode kesalahan, lihat langkah-langkah [Memecahkan masalah Amazon EMR pada penskalaan otomatis vertikal EKS](#) untuk membantu menyelesaikan masalah.

Untuk menghapus pod dan aplikasi, jalankan perintah berikut:

```
kubectl delete sparkapplication spark-pi
```

Menghapus instalasi operator Spark untuk Amazon EMR di EKS

Gunakan langkah-langkah berikut untuk menghapus instalasi operator Spark.

1. Hapus operator Spark menggunakan namespace yang benar. Untuk contoh ini, namespace adalah `spark-operator-demo`

```
helm uninstall spark-operator-demo -n spark-operator
```

2. Hapus akun layanan operator Spark:

```
kubectl delete sa emr-containers-sa-spark-operator -n spark-operator
```

3. Hapus operator Spark CustomResourceDefinitions (CRDs):

```
kubectl delete crd sparkapplications.sparkoperator.k8s.io  
kubectl delete crd scheduledsparkapplications.sparkoperator.k8s.io
```

Menggunakan konfigurasi pemantauan untuk memantau operator Spark Kubernetes dan pekerjaan Spark

Konfigurasi pemantauan memungkinkan Anda dengan mudah mengatur pengarsipan log aplikasi Spark dan log operator Anda ke Amazon S3 atau ke Amazon CloudWatch. Anda dapat memilih salah satu atau keduanya. Melakukannya menambahkan sespan agen log ke pod operator spark, driver, dan pod pelaksana Anda, dan selanjutnya meneruskan log komponen ini ke sink yang dikonfigurasi.

Prasyarat

Sebelum Anda mengonfigurasi pemantauan, pastikan untuk menyelesaikan tugas penyiapan berikut:

1. (Opsional) Jika sebelumnya Anda menginstal versi operator Spark yang lebih lama, hapus SparkApplication/ScheduledSparkApplicationCRD.

```
kubectl delete crd scheduledsparkapplications.sparkoperator.k8s.io
kubectl delete crd sparkapplications.sparkoperator.k8s.io
```

2. Buat peran operator/job eksekusi di IAM jika Anda belum memilikinya.
3. Jalankan perintah berikut untuk memperbarui kebijakan kepercayaan dari peran operator/job eksekusi yang baru saja Anda buat:

```
aws emr-containers update-role-trust-policy \
--cluster-name cluster \
--namespace namespace \
--role-name iam_role_name_for_operator/job_execution_role
```

4. Edit kebijakan kepercayaan peran IAM dari peran operator/job eksekusi Anda menjadi berikut:

```
{
  "Effect": "Allow",
  "Principal": {
    "Federated": "${OIDC-provider}"
  },
  "Action": "sts:AssumeRoleWithWebIdentity",
  "Condition": {
    "StringLike": {
      "OIDC_PROVIDER:sub": "system:serviceaccount:${Namespace}:emr-
containers-sa-*"
    }
  }
}
```

5. Buat kebijakan MonitoringConfiguration di IAM dengan izin berikut:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
```

```

        "logs:DescribeLogStreams",
        "logs:CreateLogStream",
        "logs:CreateLogGroup",
        "logs:PutLogEvents"
    ],
    "Resource": [
        "arn:aws:logs:*:*:log-group:log_group_name",
        "arn:aws:logs:*:*:log-group:log_group_name:*"
    ],
    "Sid": "AllowLOGSDescribeLogStreams"
},
{
    "Effect": "Allow",
    "Action": [
        "logs:DescribeLogGroups"
    ],
    "Resource": [
        "*"
    ],
    "Sid": "AllowLOGSDescribeLogGroups"
},
{
    "Effect": "Allow",
    "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:ListBucket"
    ],
    "Resource": [
        "arn:aws:s3:::bucket_name",
        "arn:aws:s3:::bucket_name/*"
    ],
    "Sid": "AllowS3PutObject"
}
]
}

```

6. Lampirkan kebijakan di atas ke peran operator/job eksekusi Anda.

Log Operator Spark

Anda dapat menentukan konfigurasi pemantauan dengan cara berikut saat melakukan `helm install`:

```

helm install spark-operator spark-operator \
--namespace namespace \
--set emrContainers.awsRegion=aws_region \
--set emrContainers.monitoringConfiguration.image=log_agent_image_url \
--set
  emrContainers.monitoringConfiguration.s3MonitoringConfiguration.logUri=S3_bucket_uri \
--set
  emrContainers.monitoringConfiguration.cloudWatchMonitoringConfiguration.logGroupName=log_group
\
--set
  emrContainers.monitoringConfiguration.cloudWatchMonitoringConfiguration.logStreamNamePrefix=log_group
\
--set emrContainers.monitoringConfiguration.sideCarResources.limits.cpuLimit=500m \
--set emrContainers.monitoringConfiguration.sideCarResources.limits.memoryLimit=512Mi \
--set
  emrContainers.monitoringConfiguration.containerLogRotationConfiguration.rotationSize=2GB
\
--set
  emrContainers.monitoringConfiguration.containerLogRotationConfiguration.maxFilesToKeep=10
\
--set webhook.enable=true \
--set emrContainers.operatorExecutionRoleArn=operator_execution_role_arn

```

Konfigurasi pemantauan

Berikut ini adalah opsi konfigurasi yang tersedia di bawah MonitoringConfiguration.

- Gambar (opsional) - URL gambar agen log. Akan diambil emrReleaseLabel jika tidak disediakan.
- s3 MonitoringConfiguration - Atur opsi ini untuk mengarsipkan ke Amazon S3.
 - LoGuri — (wajib) — Jalur bucket Amazon S3 tempat Anda ingin menyimpan log Anda.
 - Berikut ini adalah contoh format untuk jalur bucket Amazon S3, setelah log diunggah. Contoh pertama menunjukkan tidak ada rotasi log yang diaktifkan.

```

s3://{logUri}/{POD NAME}/operator/stdout.gz
s3://{logUri}/{POD NAME}/operator/stderr.gz

```

Rotasi log diaktifkan secara default. Anda dapat melihat file yang diputar, dengan indeks incrementing, dan file saat ini, yang sama dengan sampel sebelumnya.

```

s3://{logUri}/{POD NAME}/operator/stdout_YYYYMMDD_index.gz

```

```
s3://${logUri}/${POD_NAME}/operator/stderr_YYYYMMDD_index.gz
```

- `cloudWatchMonitoringKonfigurasi` - Kunci konfigurasi untuk mengatur penerusan ke Amazon CloudWatch
 - `logGroupName(wajib)` - Nama grup Amazon CloudWatch log yang ingin Anda kirim log. Grup secara otomatis akan dibuat jika tidak ada.
 - `logStreamNameAwalan (opsional)` - Nama aliran log yang ingin Anda kirim log ke. Nilai default adalah string kosong. Formatnya Amazon CloudWatch adalah sebagai berikut:

```
${logStreamNamePrefix}/${POD_NAME}/STDOUT or STDERR
```

- `sideCarResources(opsional)` — Kunci konfigurasi untuk menetapkan batas sumber daya pada wadah sidecar Fluentd yang diluncurkan.
 - `MemoryLimit (opsional)` - Batas memori. Sesuaikan sesuai dengan kebutuhan Anda. Defaultnya adalah 512Mi.
 - `CPULimit (opsional)` — Batas CPU. Sesuaikan sesuai dengan kebutuhan Anda. Defaultnya adalah 500m.
- `containerLogRotationKonfigurasi (opsional)` - Mengontrol perilaku rotasi log kontainer. Agen tidak diaktifkan secara default.
 - `RotationSize (wajib)` - Menentukan ukuran file untuk rotasi log. Kisaran nilai yang mungkin adalah dari 2KB hingga 2GB. Bagian unit numerik dari parameter `RotationSize` dilewatkan sebagai bilangan bulat. Karena nilai desimal tidak didukung, Anda dapat menentukan ukuran rotasi 1,5GB, misalnya, dengan nilai 1500MB. Defaultnya adalah 2GB.
 - `maxFilesToKeep (required)` - Menentukan jumlah maksimum file untuk mempertahankan dalam wadah setelah rotasi telah terjadi. Nilai minimum adalah 1, dan nilai maksimum adalah 50. Default-nya adalah 10.

Setelah `MonitoringConfiguration` dikonfigurasi, Anda harus dapat memeriksa log pod operator spark pada bucket Amazon S3 atau atau keduanya. Amazon CloudWatch Untuk bucket Amazon S3, Anda harus menunggu 2 menit agar file log pertama dibilas.

Untuk menemukan log in Amazon CloudWatch, Anda dapat menavigasi ke berikut: `CloudWatch> Grup log > > Log group namePod name/operator/stderr`

Atau Anda dapat menavigasi ke: `CloudWatch> Grup log > > Log group namePod name/operator/stdout`

Log Aplikasi Spark

Anda dapat menentukan konfigurasi ini dengan cara berikut.

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: spark-pi
  namespace: namespace
spec:
  type: Scala
  mode: cluster
  imagePullPolicy: Always
  mainClass: org.apache.spark.examples.SparkPi
  mainApplicationFile: "local:///usr/lib/spark/examples/jars/spark-examples.jar"
  sparkVersion: "3.3.1"
  emrReleaseLabel: emr_release_label
  executionRoleArn: job_execution_role_arn
  restartPolicy:
    type: Never
  volumes:
    - name: "test-volume"
      hostPath:
        path: "/tmp"
        type: Directory
  driver:
    cores: 1
    coreLimit: "1200m"
    memory: "512m"
    labels:
      version: 3.3.1
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
  executor:
    cores: 1
    instances: 1
    memory: "512m"
    labels:
      version: 3.3.1
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
  monitoringConfiguration:
```

```

image: "log_agent_image"
s3MonitoringConfiguration:
  logUri: "S3_bucket_uri"
cloudWatchMonitoringConfiguration:
  logGroupName: "log_group_name"
  logStreamNamePrefix: "log_stream_prefix"
sidecarResources:
  limits:
    cpuLimit: "500m"
    memoryLimit: "250Mi"
containerLogRotationConfiguration:
  rotationSize: "2GB"
  maxFilesToKeep: "10"

```

Berikut ini adalah opsi konfigurasi yang tersedia di bawah MonitoringConfiguration.

- Gambar (opsional) - URL gambar agen log. Akan diambil emrReleaseLabel jika tidak disediakan.
- s3 MonitoringConfiguration - Atur opsi ini untuk mengarsipkan ke Amazon S3.
- LoGuri (wajib) - Jalur bucket Amazon S3 tempat Anda ingin menyimpan log Anda. Contoh pertama menunjukkan tidak ada rotasi log yang diaktifkan:

```

s3://${logUri}/${APPLICATION_NAME}-${APPLICATION_UID}/${POD_NAME}/stdout.gz
s3://${logUri}/${APPLICATION_NAME}-${APPLICATION_UID}/${POD_NAME}/stderr.gz

```

Rotasi log diaktifkan secara default. Anda dapat menggunakan file yang diputar (dengan indeks incrementing) dan file saat ini (satu tanpa cap tanggal).

```

s3://${logUri}/${APPLICATION_NAME}-${APPLICATION_UID}/${POD_NAME}/
stdout_YYYYMMDD_index.gz
s3://${logUri}/${APPLICATION_NAME}-${APPLICATION_UID}/${POD_NAME}/
stderr_YYYYMMDD_index.gz

```

- cloudWatchMonitoringKonfigurasi - Kunci konfigurasi untuk mengatur penerusan ke Amazon CloudWatch
- logGroupName(wajib) — Nama grup log Cloudwatch yang ingin Anda kirim log. Grup secara otomatis dibuat jika tidak ada.
- logStreamNameAwalan (opsional) - Nama aliran log yang ingin Anda kirim log ke. Nilai default adalah string kosong. Formatnya CloudWatch adalah sebagai berikut:

```

${logStreamNamePrefix}/${APPLICATION_NAME}-${APPLICATION_UID}/${POD_NAME}/stdout

```

```
${LogStreamNamePrefix}/${APPLICATION_NAME}-${APPLICATION_UID}/${POD_NAME}/stderr
```

- `sideCarResources`(opsional) — Kunci konfigurasi untuk menetapkan batas sumber daya pada wadah sidecar Fluentd yang diluncurkan.
 - `MemoryLimit` (opsional) - Batas memori. Sesuaikan sesuai dengan kebutuhan Anda. Defaultnya adalah 250Mi.
 - `CPUlimit` — Batas CPU. Sesuaikan sesuai dengan kebutuhan Anda. Defaultnya adalah 500m.
- `containerLogRotationKonfigurasi` (opsional) - Mengontrol perilaku rotasi log kontainer. Agen tidak diaktifkan secara default.
 - `RotationSize` (wajib) - Menentukan ukuran file untuk rotasi log. Kisaran nilai yang mungkin adalah dari 2KB hingga 2GB. Bagian unit numerik dari parameter `RotationSize` dilewatkan sebagai bilangan bulat. Karena nilai desimal tidak didukung, Anda dapat menentukan ukuran rotasi 1,5GB, misalnya, dengan nilai 1500MB. Defaultnya adalah 2GB.
 - `maxFilesToKeep` (required) - Menentukan jumlah maksimum file untuk mempertahankan dalam wadah setelah rotasi telah terjadi. Nilai minimum adalah 1. Nilai maksimumnya adalah 50. Default-nya adalah 10.

Setelah mengonfigurasi `MonitoringConfiguration`, Anda harus dapat memeriksa driver aplikasi spark dan log pelaksana Anda di bucket Amazon S3 atau atau keduanya. CloudWatch Untuk bucket Amazon S3, Anda harus menunggu 2 menit agar file log pertama dibilas. Misalnya, di Amazon S3, jalur bucket muncul seperti berikut:

```
Amazon S3 > Ember > > Bucket name> stderr.gz Spark application name - UUID Pod Name
```

Atau:

```
Amazon S3 > Ember > > Bucket name> stdout.gz Spark application name - UUID Pod Name
```

Di CloudWatch, jalur muncul seperti berikut:

```
CloudWatch> Grup log > Log group name>Spark application name - UUID/Pod name/stderr
```

Atau:

```
CloudWatch> Grup log > Log group name>Spark application name - UUID/Pod name/stdout
```

Keamanan dan operator Spark dengan Amazon EMR di EKS

Ada beberapa cara untuk mengatur izin akses cluster saat Anda menggunakan operator Spark. Yang pertama adalah menggunakan kontrol akses berbasis peran, kontrol akses berbasis peran (RBAC) membatasi akses berdasarkan peran seseorang dalam suatu organisasi. Ini telah menjadi cara utama untuk menangani akses. Metode akses kedua adalah untuk mengambil AWS Identity and Access Management peran, yang menyediakan akses sumber daya melalui izin tertentu yang ditetapkan.

Topik

- [Menyiapkan izin akses cluster dengan kontrol akses berbasis peran \(RBAC\)](#)
- [Menyiapkan izin akses klaster dengan peran IAM untuk akun layanan \(IRSA\)](#)

Menyiapkan izin akses cluster dengan kontrol akses berbasis peran (RBAC)

Untuk menyebarkan operator Spark, Amazon EMR di EKS membuat dua peran dan akun layanan untuk operator Spark dan aplikasi Spark.

Topik

- [Akun dan peran layanan operator](#)
- [Akun dan peran layanan Spark](#)

Akun dan peran layanan operator

Amazon EMR di EKS membuat akun layanan operator dan peran SparkApplications untuk mengelola pekerjaan Spark dan sumber daya lain seperti layanan.

Nama default untuk akun layanan ini adalah `emr-containers-sa-spark-operator`.

Aturan berikut berlaku untuk peran layanan ini:

```
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - "*"
- apiGroups:
```

```
- ""
resources:
- services
- configmaps
- secrets
verbs:
- create
- get
- delete
- update
- apiGroups:
- extensions
- networking.k8s.io
resources:
- ingresses
verbs:
- create
- get
- delete
- apiGroups:
- ""
resources:
- nodes
verbs:
- get
- apiGroups:
- ""
resources:
- events
verbs:
- create
- update
- patch
- apiGroups:
- ""
resources:
- resourcequotas
verbs:
- get
- list
- watch
- apiGroups:
- apiextensions.k8s.io
resources:
```

```
- customresourcedefinitions
verbs:
- create
- get
- update
- delete
- apiGroups:
- admissionregistration.k8s.io
resources:
- mutatingwebhookconfigurations
- validatingwebhookconfigurations
verbs:
- create
- get
- update
- delete
- apiGroups:
- sparkoperator.k8s.io
resources:
- sparkapplications
- sparkapplications/status
- scheduledsparkapplications
- scheduledsparkapplications/status
verbs:
- "*"
{{- if .Values.batchScheduler.enable }}
# required for the `volcano` batch scheduler
- apiGroups:
- scheduling.incubator.k8s.io
- scheduling.sigs.dev
- scheduling.volcano.sh
resources:
- podgroups
verbs:
- "*"
{{- end }}
{{ if .Values.webhook.enable }}
- apiGroups:
- batch
resources:
- jobs
verbs:
- delete
```

```
{{- end }}
```

Akun dan peran layanan Spark

Pod driver Spark membutuhkan akun layanan Kubernetes di namespace yang sama dengan pod. Akun layanan ini membutuhkan izin untuk membuat, mendapatkan, membuat daftar, menambal, dan menghapus pod pelaksana, dan untuk membuat layanan tanpa kepala Kubernetes untuk driver. Driver gagal dan keluar tanpa akun layanan kecuali akun layanan default di namespace pod memiliki izin yang diperlukan.

Nama default untuk akun layanan ini adalah `emr-containers-sa-spark`.

Aturan berikut berlaku untuk peran layanan ini:

```
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - services
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - configmaps
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - persistentvolumeclaims
  verbs:
  - "*"

```

Menyiapkan izin akses klaster dengan peran IAM untuk akun layanan (IRSA)

Bagian ini menggunakan contoh untuk menunjukkan cara mengonfigurasi akun layanan Kubernetes untuk mengambil peran. AWS Identity and Access Management Pod yang menggunakan akun layanan kemudian dapat mengakses AWS layanan apa pun yang perannya memiliki izin untuk diakses.

Contoh berikut menjalankan aplikasi Spark untuk menghitung kata-kata dari file di Amazon S3. Untuk melakukan ini, Anda dapat mengatur peran IAM untuk akun layanan (IRSA) untuk mengautentikasi dan mengotorisasi akun layanan Kubernetes.

Note

Contoh ini menggunakan namespace “spark-operator” untuk operator Spark dan untuk namespace tempat Anda mengirimkan aplikasi Spark.

Prasyarat

Sebelum Anda mencoba contoh di halaman ini, lengkapi prasyarat berikut:

- [Siapkan untuk operator Spark](#).
- [Instal operator Spark](#).
- [Buat bucket Amazon S3](#).
- Simpan puisi favorit Anda dalam file teks bernama `poem.txt`, dan unggah file ke bucket S3 Anda. Aplikasi Spark yang Anda buat di halaman ini akan membaca isi file teks. Untuk informasi selengkapnya tentang mengunggah file ke S3, lihat [Mengunggah objek ke bucket](#) di Panduan Pengguna Layanan Penyimpanan Sederhana Amazon.

Konfigurasi akun layanan Kubernetes untuk mengambil peran IAM

Gunakan langkah-langkah berikut untuk mengonfigurasi akun layanan Kubernetes untuk mengambil peran IAM yang dapat digunakan pod untuk mengakses AWS layanan yang memiliki izin untuk diakses oleh peran tersebut.

1. Setelah menyelesaikan [Prasyarat](#), gunakan AWS Command Line Interface untuk membuat `example-policy.json` file yang memungkinkan akses hanya-baca ke file yang Anda unggah ke Amazon S3:

```

cat >example-policy.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::my-pod-bucket",
        "arn:aws:s3:::my-pod-bucket/*"
      ]
    }
  ]
}
EOF

```

2. Kemudian, buat kebijakan `example-policy` IAM:

```

aws iam create-policy --policy-name example-policy --policy-document file://
example-policy.json

```

3. Selanjutnya, buat peran IAM `example-role` dan kaitkan dengan akun layanan Kubernetes untuk driver Spark:

```

eksctl create iamserviceaccount --name driver-account-sa --namespace spark-operator
\
--cluster my-cluster --role-name "example-role" \
--attach-policy-arn arn:aws:iam::111122223333:policy/example-policy --approve

```

4. Buat file `yaml` dengan binding peran cluster yang diperlukan untuk akun layanan driver Spark:

```

cat >spark-rbac.yaml <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  name: driver-account-sa
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding

```

```
metadata:
  name: spark-role
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: edit
subjects:
- kind: ServiceAccount
  name: driver-account-sa
  namespace: spark-operator
EOF
```

5. Menerapkan konfigurasi pengikatan peran cluster:

```
kubectl apply -f spark-rbac.yaml
```

Perintah kubectl harus mengonfirmasi keberhasilan pembuatan akun:

```
serviceaccount/driver-account-sa created
clusterrolebinding.rbac.authorization.k8s.io/spark-role configured
```

Menjalankan aplikasi dari operator Spark

Setelah Anda [mengonfigurasi akun layanan Kubernetes](#), Anda dapat menjalankan aplikasi Spark yang menghitung jumlah kata dalam file teks yang Anda unggah sebagai bagian dari [Prasyarat](#)

1. Buat file baru `word-count.yaml`, dengan `SparkApplication` definisi untuk aplikasi penghitungan kata Anda, berdasarkan Amazon EMR versi 6.

```
cat >word-count.yaml <<EOF
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: word-count
  namespace: spark-operator
spec:
  type: Java
  mode: cluster
  image: "895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.10.0:latest"
  imagePullPolicy: Always
  mainClass: org.apache.spark.examples.JavaWordCount
```

```
mainApplicationFile: local:///usr/lib/spark/examples/jars/spark-examples.jar
arguments:
  - s3://my-pod-bucket/poem.txt
hadoopConf:
  # EMRFS filesystem
  fs.s3.customAWSCredentialsProvider:
com.amazonaws.auth.WebIdentityTokenCredentialsProvider
  fs.s3.impl: com.amazon.ws.emr.hadoop.fs.EmrFileSystem
  fs.AbstractFileSystem.s3.impl: org.apache.hadoop.fs.s3.EMRFSDelegate
  fs.s3.buffer.dir: /mnt/s3
  fs.s3.getObject.initialSocketTimeoutMilliseconds: "2000"

mapreduce.fileoutputcommitter.algorithm.version.emr_internal_use_only.EmrFileSystem:
"2"
  mapreduce.fileoutputcommitter.cleanup-
failures.ignored.emr_internal_use_only.EmrFileSystem: "true"
  sparkConf:
    # Required for EMR Runtime
    spark.driver.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/
hadoop-aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/
share/aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/
security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-
glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-
serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/
hadoop/extrajars/*
    spark.driver.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/
lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/native
    spark.executor.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/
hadoop-aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/
share/aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/
security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-
glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-
serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/
hadoop/extrajars/*
    spark.executor.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-
lzo/lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/
native
  sparkVersion: "3.3.1"
  restartPolicy:
    type: Never
  driver:
    cores: 1
    coreLimit: "1200m"
    memory: "512m"
```

```

labels:
  version: 3.3.1
  serviceAccount: my-spark-driver-sa
executor:
  cores: 1
  instances: 1
  memory: "512m"
  labels:
    version: 3.3.1
EOF

```

Jika Anda menggunakan operator percikan dengan rilis versi 7, Anda menyesuaikan beberapa nilai konfigurasi:

```

cat >word-count.yaml <<EOF
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: word-count
  namespace: spark-operator
spec:
  type: Java
  mode: cluster
  image: "895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-7.7.0:latest"
  imagePullPolicy: Always
  mainClass: org.apache.spark.examples.JavaWordCount
  mainApplicationFile: local:///usr/lib/spark/examples/jars/spark-examples.jar
  arguments:
    - s3://my-pod-bucket/poem.txt
  hadoopConf:
    # EMRFS filesystem
    fs.s3.customAWSCredentialsProvider:
com.amazonaws.auth.WebIdentityTokenCredentialsProvider
    fs.s3.impl: com.amazon.ws.emr.hadoop.fs.EmrFileSystem
    fs.AbstractFileSystem.s3.impl: org.apache.hadoop.fs.s3.EMRFSDelegate
    fs.s3.buffer.dir: /mnt/s3
    fs.s3.getObject.initialSocketTimeoutMilliseconds: "2000"

  mapreduce.fileoutputcommitter.algorithm.version.emr_internal_use_only.EmrFileSystem:
"2"
    mapreduce.fileoutputcommitter.cleanup-
failures.ignored.emr_internal_use_only.EmrFileSystem: "true"
  sparkConf:

```

```
# Required for EMR Runtime
spark.driver.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/hadoop-aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/aws-java-sdk-v2/*:/usr/share/aws/emr/emrfs/conf:/usr/share/aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/hadoop/extrajars/*
spark.driver.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/native
spark.executor.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/hadoop-aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/aws-java-sdk-v2/*:/usr/share/aws/emr/emrfs/conf:/usr/share/aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/hadoop/extrajars/*
spark.executor.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/native
sparkVersion: "3.3.1"
restartPolicy:
  type: Never
driver:
  cores: 1
  coreLimit: "1200m"
  memory: "512m"
  labels:
    version: 3.3.1
  serviceAccount: my-spark-driver-sa
executor:
  cores: 1
  instances: 1
  memory: "512m"
  labels:
    version: 3.3.1
EOF
```

2. Kirim aplikasi Spark.

```
kubectl apply -f word-count.yaml
```

Perintah `kubectl` harus mengembalikan konfirmasi bahwa Anda berhasil membuat objek yang `SparkApplication` dipanggil. `word-count`

```
sparkapplication.sparkoperator.k8s.io/word-count configured
```

3. Untuk memeriksa peristiwa untuk `SparkApplication` objek, jalankan perintah berikut:

```
kubectl describe sparkapplication word-count -n spark-operator
```

Perintah `kubectl` harus mengembalikan deskripsi `SparkApplication` dengan peristiwa:

```
Events:
  Type            Reason                                     Age                From
  Message
  ----            -
  -----
  Normal          SparkApplicationSpecUpdateProcessed      3m2s (x2 over 17h) spark-
operator        Successfully processed spec update for SparkApplication word-count
  Warning         SparkApplicationPendingRerun            3m2s (x2 over 17h) spark-
operator        SparkApplication word-count is pending rerun
  Normal          SparkApplicationSubmitted                2m58s (x2 over 17h) spark-
operator        SparkApplication word-count was submitted successfully
  Normal          SparkDriverRunning                      2m56s (x2 over 17h) spark-
operator        Driver word-count-driver is running
  Normal          SparkExecutorPending                    2m50s              spark-
operator        Executor [javawordcount-fdd1698807392c66-exec-1] is pending
  Normal          SparkExecutorRunning                    2m48s              spark-
operator        Executor [javawordcount-fdd1698807392c66-exec-1] is running
  Normal          SparkDriverCompleted                    2m31s (x2 over 17h) spark-
operator        Driver word-count-driver completed
  Normal          SparkApplicationCompleted                2m31s (x2 over 17h) spark-
operator        SparkApplication word-count completed
  Normal          SparkExecutorCompleted                   2m31s (x2 over 2m31s) spark-
operator        Executor [javawordcount-fdd1698807392c66-exec-1] completed
```

Aplikasi ini sekarang menghitung kata-kata dalam file S3 Anda. Untuk menemukan jumlah kata, lihat file log untuk driver Anda:

```
kubectl logs pod/word-count-driver -n spark-operator
```

Perintah `kubectl` harus mengembalikan isi file log dengan hasil aplikasi hitungan kata Anda.

```
INFO DAGScheduler: Job 0 finished: collect at JavaWordCount.java:53, took 5.146519 s
Software: 1
```

Untuk informasi selengkapnya tentang cara mengirimkan aplikasi ke Spark melalui operator Spark, lihat dokumentasi [Using a SparkApplication](#) in the Kubernetes Operator for Apache Spark (8s-operator). `spark-on-k` GitHub

Menjalankan pekerjaan Spark dengan `spark-submit`

Amazon EMR merilis 6.10.0 dan dukungan yang lebih tinggi `spark-submit` sebagai alat baris perintah yang dapat Anda gunakan untuk mengirimkan dan menjalankan aplikasi Spark ke EMR Amazon di kluster EKS.

Note

Amazon EMR menghitung harga di Amazon EKS berdasarkan vCPU dan konsumsi memori. Perhitungan ini berlaku untuk pod driver dan executor. Perhitungan ini dimulai dari saat Anda mengunduh image aplikasi Amazon EMR hingga pod Amazon EKS berakhir dan dibulatkan ke detik terdekat.

Topik

- [Menyiapkan `spark-submit` untuk Amazon EMR di EKS](#)
- [Memulai dengan `spark-submit` untuk Amazon EMR di EKS](#)
- [Verifikasi persyaratan keamanan akun layanan driver Spark untuk pengiriman spark](#)

Menyiapkan `spark-submit` untuk Amazon EMR di EKS

Selesaikan tugas-tugas berikut untuk menyiapkan sebelum Anda dapat menjalankan aplikasi dengan `spark-submit` di Amazon EMR di EKS. Jika Anda sudah mendaftar untuk Amazon Web Services (AWS) dan telah menggunakan Amazon EKS, Anda hampir siap untuk menggunakan Amazon EMR di EKS. Jika Anda telah menyelesaikan salah satu prasyarat, Anda dapat melewatinya dan melanjutkan ke yang berikutnya.

- [Instal atau perbarui ke versi terbaru AWS CLI](#) - Jika Anda sudah menginstal AWS CLI, konfirmasikan bahwa Anda memiliki versi terbaru.
- [Siapkan kubectl dan eksctl - eksctl](#) adalah alat baris perintah yang Anda gunakan untuk berkomunikasi dengan Amazon EKS.
- [Memulai Amazon EKS — eksctl](#) — Ikuti langkah-langkah untuk membuat cluster Kubernetes baru dengan node di Amazon EKS.
- [Pilih URI gambar dasar EMR Amazon](#) (rilis 6.10.0 atau lebih tinggi) — `spark-submit` perintah ini didukung dengan rilis Amazon EMR 6.10.0 dan yang lebih tinggi.
- Konfirmasikan bahwa akun layanan driver memiliki izin yang sesuai untuk membuat dan menonton pod pelaksana. Untuk informasi selengkapnya, lihat [Verifikasi persyaratan keamanan akun layanan driver Spark untuk pengiriman spark](#).
- Siapkan profil [AWS kredensial](#) lokal Anda.
- Dari konsol Amazon EKS, pilih kluster EKS Anda, lalu temukan titik akhir kluster EKS, yang terletak di bawah Ikhtisar, Detail, lalu titik akhir server API.

Memulai dengan `spark-submit` untuk Amazon EMR di EKS

Amazon EMR 6.10.0 dan yang lebih tinggi mendukung `spark-submit` untuk menjalankan aplikasi Spark di cluster Amazon EKS. Bagian berikut menunjukkan kepada Anda cara mengirimkan perintah untuk aplikasi Spark.

Jalankan aplikasi Spark

Untuk menjalankan aplikasi Spark, ikuti langkah-langkah berikut:

1. Sebelum Anda dapat menjalankan aplikasi Spark dengan `spark-submit` perintah, selesaikan langkah-langkahnya. [Menyiapkan `spark-submit` untuk Amazon EMR di EKS](#)
2. Jalankan wadah dengan EMR Amazon pada gambar dasar EKS. Lihat [Cara memilih URI gambar dasar](#) untuk informasi selengkapnya.

```
kubectl run -it containerName --image=EMRonEKSIImage --command -n namespace /bin/  
bash
```

3. Tetapkan nilai untuk variabel lingkungan berikut:


```
export SPARK_HOME=spark-home
```

```
export MASTER_URL=k8s://Amazon EKS-cluster-endpoint
```

4. Sekarang, kirimkan aplikasi Spark dengan perintah berikut:

```
$SPARK_HOME/bin/spark-submit \  
  --class org.apache.spark.examples.SparkPi \  
  --master $MASTER_URL \  
  --conf spark.kubernetes.container.image=895885662937.dkr.ecr.us-  
west-2.amazonaws.com/spark/emr-6.10.0:latest \  
  --conf spark.kubernetes.authenticate.driver.serviceAccountName=spark \  
  --deploy-mode cluster \  
  --conf spark.kubernetes.namespace=spark-operator \  
  local:///usr/lib/spark/examples/jars/spark-examples.jar 20
```

Untuk informasi selengkapnya tentang mengirimkan aplikasi ke Spark, lihat [Mengirimkan aplikasi](#) dalam dokumentasi Apache Spark.

 Important

spark-submitnya mendukung mode cluster sebagai mekanisme pengiriman.

Verifikasi persyaratan keamanan akun layanan driver Spark untuk pengiriman spark

Pod driver Spark menggunakan akun layanan Kubernetes untuk mengakses server API Kubernetes untuk membuat dan menonton pod pelaksana. Akun layanan driver harus memiliki izin yang sesuai untuk membuat daftar, membuat, mengedit, menambal, dan menghapus pod di klaster Anda. Anda dapat memverifikasi bahwa Anda dapat mencantumkan sumber daya ini dengan menjalankan perintah berikut:

```
kubectl auth can-i list/create/edit/delete/patch pods
```

Verifikasi bahwa Anda memiliki izin yang diperlukan dengan menjalankan setiap perintah.

```
kubectl auth can-i list pods  
kubectl auth can-i create pods  
kubectl auth can-i edit pods  
kubectl auth can-i delete pods
```

```
kubectl auth can-i patch pods
```

Aturan berikut berlaku untuk peran layanan ini:

```
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - services
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - configmaps
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - persistentvolumeclaims
  verbs:
  - "*"
```

Menyiapkan peran IAM untuk akun layanan (IRSA) untuk spark-submit

Bagian berikut menjelaskan cara mengatur peran IAM untuk akun layanan (IRSA) untuk mengautentikasi dan mengotorisasi akun layanan Kubernetes sehingga Anda dapat menjalankan aplikasi Spark yang disimpan di Amazon S3.

Prasyarat

Sebelum mencoba salah satu contoh dalam dokumentasi ini, pastikan Anda telah menyelesaikan prasyarat berikut:

- [Selesai menyiapkan spark-submit](#)

- [Membuat ember S3](#) dan [mengunggah toples aplikasi](#) percikan

Mengonfigurasi akun layanan Kubernetes untuk mengambil peran IAM

Langkah-langkah berikut mencakup cara mengkonfigurasi akun layanan Kubernetes untuk mengambil peran AWS Identity and Access Management (IAM). Setelah Anda mengonfigurasi pod untuk menggunakan akun layanan, mereka kemudian dapat mengakses apa pun Layanan AWS yang memiliki izin untuk diakses oleh peran tersebut.

1. [Buat file kebijakan untuk mengizinkan akses hanya-baca ke objek Amazon S3 yang Anda unggah:](#)

```
cat >my-policy.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::<my-spark-jar-bucket>",
        "arn:aws:s3:::<my-spark-jar-bucket>/*"
      ]
    }
  ]
}
EOF
```

2. Buat kebijakan IAM.

```
aws iam create-policy --policy-name my-policy --policy-document file://my-policy.json
```

3. Buat peran IAM dan kaitkan dengan akun layanan Kubernetes untuk driver Spark

```
eksctl create iamserviceaccount --name my-spark-driver-sa --namespace spark-operator \
--cluster my-cluster --role-name "my-role" \
```

```
--attach-policy-arn arn:aws:iam::111122223333:policy/my-policy --approve
```

4. Buat file YAMAL dengan [izin yang diperlukan untuk akun](#) layanan driver Spark:

```
cat >spark-rbac.yaml <<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: emr-containers-role-spark
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - services
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - configmaps
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - persistentvolumeclaims
  verbs:
  - "*"
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: spark-role-binding
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
```

```

name: emr-containers-role-spark
subjects:
- kind: ServiceAccount
  name: emr-containers-sa-spark
  namespace: default
EOF

```

5. Menerapkan konfigurasi pengikatan peran cluster.

```
kubectl apply -f spark-rbac.yaml
```

6. kubectlPerintah harus mengembalikan konfirmasi akun yang dibuat.

```

serviceaccount/emr-containers-sa-spark created
clusterrolebinding.rbac.authorization.k8s.io/emr-containers-role-spark configured

```

Menjalankan aplikasi Spark

Amazon EMR 6.10.0 dan yang lebih tinggi mendukung spark-submit untuk menjalankan aplikasi Spark di cluster Amazon EKS. Untuk menjalankan aplikasi Spark, ikuti langkah-langkah berikut:

1. Pastikan Anda telah menyelesaikan langkah-langkah dalam [Menyiapkan spark-submit untuk Amazon EMR](#) di EKS.
2. Tetapkan nilai untuk variabel lingkungan berikut:

```

export SPARK_HOME=spark-home
export MASTER_URL=k8s://Amazon EKS-cluster-endpoint

```

3. Sekarang, kirimkan aplikasi Spark dengan perintah berikut:

```

$SPARK_HOME/bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master $MASTER_URL \
  --conf spark.kubernetes.container.image=895885662937.dkr.ecr.us-
west-2.amazonaws.com/spark/emr-6.15.0:latest \
  --conf spark.kubernetes.authenticate.driver.serviceAccountName=emr-containers-sa-
spark \
  --deploy-mode cluster \
  --conf spark.kubernetes.namespace=default \
  --conf "spark.driver.extraClassPath=/usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/
hadoop-aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/

```

```

share/aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/
security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-
glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-
serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/
hadoop/extrajars/*" \
--conf "spark.driver.extraLibraryPath=/usr/lib/hadoop/lib/native:/usr/lib/hadoop-
lzo/lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/
native" \
--conf "spark.executor.extraClassPath=/usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/
hadoop-aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/
share/aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/
security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-
glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-
serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/
hadoop/extrajars/*" \
--conf "spark.executor.extraLibraryPath=/usr/lib/hadoop/lib/native:/usr/lib/
hadoop-lzo/lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/
lib/native" \
--conf
spark.hadoop.fs.s3.customAWSCredentialsProvider=com.amazonaws.auth.WebIdentityTokenCredent
\
--conf spark.hadoop.fs.s3.impl=com.amazon.ws.emr.hadoop.fs.EmrFileSystem \
--conf
spark.hadoop.fs.AbstractFileSystem.s3.impl=org.apache.hadoop.fs.s3.EMRFSDelegate \
--conf spark.hadoop.fs.s3.buffer.dir=/mnt/s3 \
--conf spark.hadoop.fs.s3.getObject.initialSocketTimeoutMilliseconds="2000" \
--conf
spark.hadoop.mapreduce.fileoutputcommitter.algorithm.version.emr_internal_use_only.EmrFile
\
--conf spark.hadoop.mapreduce.fileoutputcommitter.cleanup-
failures.ignored.emr_internal_use_only.EmrFileSystem="true" \
s3://my-pod-bucket/spark-examples.jar 20

```

4. Setelah driver percikan menyelesaikan pekerjaan Spark, Anda akan melihat baris log di akhir kiriman yang menunjukkan bahwa pekerjaan Spark telah selesai.

```

23/11/24 17:02:14 INFO LoggingPodStatusWatcherImpl: Application
org.apache.spark.examples.SparkPi with submission ID default:org-apache-spark-
examples-sparkpi-4980808c03ff3115-driver finished
23/11/24 17:02:14 INFO ShutdownHookManager: Shutdown hook called

```

Pembersihan

Setelah selesai menjalankan aplikasi, Anda dapat melakukan pembersihan dengan perintah berikut.

```
kubectl delete -f spark-rbac.yaml
```

Menggunakan Apache Livy dengan Amazon EMR di EKS

Dengan Amazon EMR rilis 7.1.0 dan yang lebih tinggi, Anda dapat menggunakan Apache Livy untuk mengirimkan pekerjaan di Amazon EMR di EKS. Menggunakan Apache Livy, Anda dapat mengatur titik akhir Apache Livy REST Anda sendiri dan menggunakannya untuk menyebarkan dan mengelola aplikasi Spark di kluster Amazon EKS Anda. Setelah Anda menginstal Livy di kluster Amazon EKS Anda, Anda dapat menggunakan titik akhir Livy untuk mengirimkan aplikasi Spark ke server Livy Anda. Server mengelola siklus hidup aplikasi Spark.

Note

Amazon EMR menghitung harga di Amazon EKS berdasarkan vCPU dan konsumsi memori. Perhitungan ini berlaku untuk pod driver dan executor. Perhitungan ini dimulai dari saat Anda mengunduh image aplikasi Amazon EMR hingga pod Amazon EKS berakhir dan dibulatkan ke detik terdekat.

Topik

- [Menyiapkan Apache Livy untuk Amazon EMR di EKS](#)
- [Memulai dengan Apache Livy di Amazon EMR di EKS](#)
- [Menjalankan aplikasi Spark dengan Apache Livy untuk Amazon EMR di EKS](#)
- [Mencopot instalasi Apache Livy dengan Amazon EMR di EKS](#)
- [Keamanan untuk Apache Livy dengan Amazon EMR di EKS](#)
- [Properti instalasi untuk Apache Livy di Amazon EMR pada rilis EKS](#)
- [Memecahkan masalah kesalahan format variabel lingkungan yang umum](#)

Menyiapkan Apache Livy untuk Amazon EMR di EKS

Sebelum Anda dapat menginstal Apache Livy di kluster Amazon EKS Anda, Anda harus menginstal dan mengkonfigurasi satu set alat prasyarat. Ini termasuk AWS CLI, yang merupakan alat baris

perintah dasar untuk bekerja dengan AWS sumber daya, alat baris perintah untuk bekerja dengan Amazon EKS, dan pengontrol yang digunakan dalam kasus penggunaan ini untuk membuat aplikasi cluster Anda tersedia ke internet dan untuk merutekan lalu lintas jaringan.

- [Instal atau perbarui ke versi terbaru AWS CLI](#) - Jika Anda sudah menginstal AWS CLI, konfirmasi bahwa Anda memiliki versi terbaru.
- [Siapkan kubectl dan eksctl - eksctl](#) adalah alat baris perintah yang Anda gunakan untuk berkomunikasi dengan Amazon EKS.
- [Instal Helm](#) — Manajer paket Helm untuk Kubernetes membantu Anda menginstal dan mengelola aplikasi di kluster Kubernetes Anda.
- [Memulai Amazon EKS — eksctl](#) — Ikuti langkah-langkah untuk membuat cluster Kubernetes baru dengan node di Amazon EKS.
- [Pilih label rilis Amazon EMR](#) - Apache Livy didukung dengan rilis Amazon EMR 7.1.0 dan yang lebih tinggi.
- [Instal pengontrol ALB — pengontrol](#) ALB mengelola AWS Elastic Load Balancing untuk kluster Kubernetes. Ini menciptakan AWS Network Load Balancer (NLB) ketika Anda membuat Kubernetes Ingress saat menyiapkan Apache Livy.

Memulai dengan Apache Livy di Amazon EMR di EKS

Selesaikan langkah-langkah berikut untuk menginstal Apache Livy. Mereka termasuk mengonfigurasi manajer paket, membuat namespace untuk menjalankan beban kerja Spark, menginstal Livy, menyiapkan load balancing, dan langkah-langkah verifikasi. Anda harus menyelesaikan langkah-langkah ini untuk menjalankan pekerjaan batch dengan Spark.

1. Jika Anda belum melakukannya, siapkan [Apache Livy untuk Amazon EMR](#) di EKS.
2. Otentikasi klien Helm Anda ke registri Amazon ECR. Anda dapat menemukan ECR-registry-account nilai yang sesuai untuk [akun Wilayah AWS registri Amazon ECR Anda berdasarkan Wilayah](#).

```
aws ecr get-login-password --region <AWS_REGION> | helm registry login \
--username AWS \
--password-stdin <ECR-registry-account>.dkr.ecr.<region-id>.amazonaws.com
```

3. Menyiapkan Livy membuat akun layanan untuk server Livy dan akun lain untuk aplikasi Spark. Untuk menyiapkan IRSA untuk akun layanan, lihat [Menyiapkan izin akses dengan peran IAM untuk akun layanan \(IRSA\)](#).

4. Buat namespace untuk menjalankan beban kerja Spark Anda.

```
kubectl create ns <spark-ns>
```

5. Gunakan perintah berikut untuk menginstal Livy.

Endpoint Livy ini hanya tersedia secara internal untuk VPC di cluster EKS. Untuk mengaktifkan akses di luar VPC, atur perintah instalasi `--set loadbalancer.internal=false` Helm Anda.

Note

Secara default, SSL tidak diaktifkan dalam titik akhir Livy ini dan titik akhir hanya terlihat di dalam VPC cluster EKS. Jika Anda mengatur `loadbalancer.internal=false` dan `ssl.enabled=false`, Anda mengekspos titik akhir yang tidak aman di luar VPC Anda. Untuk menyiapkan titik akhir Livy yang aman, lihat [Mengonfigurasi titik akhir Apache Livy yang aman dengan TLS/SSL](#).

```
helm install livy-demo \  
  oci://895885662937.dkr.ecr.region-id.amazonaws.com/livy \  
  --version 7.13.0 \  
  --namespace livy-ns \  
  --set image=ECR-registry-account.dkr.ecr.region-id.amazonaws.com/livy/  
emr-7.13.0:latest \  
  --set sparkNamespace=<spark-ns> \  
  --create-namespace
```

Anda akan melihat output berikut.

```
NAME: livy-demo  
LAST DEPLOYED: Mon Mar 18 09:23:23 2024  
NAMESPACE: livy-ns  
STATUS: deployed  
REVISION: 1  
TEST SUITE: None  
NOTES:  
The Livy server has been installed.  
Check installation status:  
1. Check Livy Server pod is running
```

```
kubectl --namespace livy-ns get pods -l "app.kubernetes.io/instance=livy-demo"
```

2. Verify created NLB is in Active state and it's target groups are healthy (if loadbalancer.enabled is true)

Access LIVY APIs:

```
# Ensure your NLB is active and healthy
# Get the Livy endpoint using command:
LIVY_ENDPOINT=$(kubectl get svc -n livy-ns -l app.kubernetes.io/instance=livy-demo,emr-containers.amazonaws.com/type=loadbalancer -o jsonpath='{.items[0].status.loadBalancer.ingress[0].hostname}' | awk '{printf "%s:8998\n", $0}')
# Access Livy APIs using http://$LIVY_ENDPOINT or https://$LIVY_ENDPOINT (if SSL is enabled)
# Note: While uninstalling Livy, makes sure the ingress and NLB are deleted after running the helm command to avoid dangling resources
```

Nama akun layanan default untuk server Livy dan sesi Spark adalah `emr-containers-sa-livy` dan `emr-containers-sa-spark-livy`. Untuk menggunakan nama kustom, gunakan `sparkServiceAccount.name` parameter `serviceAccounts.name` dan.

```
--set serviceAccounts.name=my-service-account-for-livy
--set sparkServiceAccount.name=my-service-account-for-spark
```

6. Verifikasi bahwa Anda menginstal bagan Helm.

```
helm list -n livy-ns -o yaml
```

`helm list` Perintah harus mengembalikan informasi tentang bagan Helm baru Anda.

```
app_version: 0.7.1-incubating
chart: livy-emr-7.13.0
name: livy-demo
namespace: livy-ns
revision: "1"
status: deployed
updated: 2024-02-08 22:39:53.539243 -0800 PST
```

7. Verifikasi bahwa Network Load Balancer aktif.

```
LIVY_NAMESPACE=<livy-ns>
LIVY_APP_NAME=<livy-app-name>
```

```

AWS_REGION=<AWS_REGION>

# Get the NLB Endpoint URL
NLB_ENDPOINT=$(kubectl --namespace $LIVY_NAMESPACE get svc -l "app.kubernetes.io/instance=$LIVY_APP_NAME,emr-containers.amazonaws.com/type=loadbalancer" -o jsonpath='{.items[0].status.loadBalancer.ingress[0].hostname}')

# Get all the load balancers in the account's region
ELB_LIST=$(aws elbv2 describe-load-balancers --region $AWS_REGION)

# Get the status of the NLB that matching the endpoint from the Kubernetes service
NLB_STATUS=$(echo $ELB_LIST | grep -A 8 "\"DNSName\": \"$NLB_ENDPOINT\"" | awk '/Code/{print $2}/' | tr -d '",\n')
echo $NLB_STATUS

```

8. Sekarang verifikasi bahwa kelompok target di Network Load Balancer sehat.

```

LIVY_NAMESPACE=<livy-ns>
LIVY_APP_NAME=<livy-app-name>
AWS_REGION=<AWS_REGION>

# Get the NLB endpoint
NLB_ENDPOINT=$(kubectl --namespace $LIVY_NAMESPACE get svc -l "app.kubernetes.io/instance=$LIVY_APP_NAME,emr-containers.amazonaws.com/type=loadbalancer" -o jsonpath='{.items[0].status.loadBalancer.ingress[0].hostname}')

# Get all the load balancers in the account's region
ELB_LIST=$(aws elbv2 describe-load-balancers --region $AWS_REGION)

# Get the NLB ARN from the NLB endpoint
NLB_ARN=$(echo $ELB_LIST | grep -B 1 "\"DNSName\": \"$NLB_ENDPOINT\"" | awk '/"LoadBalancerArn":/,/' | awk '/:/{print $2}' | tr -d '\',)

# Get the target group from the NLB. Livy setup only deploys 1 target group
TARGET_GROUP_ARN=$(aws elbv2 describe-target-groups --load-balancer-arn $NLB_ARN --region $AWS_REGION | awk '/"TargetGroupArn":/,/' | awk '/:/{print $2}' | tr -d '\',)

# Get health of target group
aws elbv2 describe-target-health --target-group-arn $TARGET_GROUP_ARN

```

Berikut ini adalah contoh output yang menunjukkan status kelompok target:

```
{
  "TargetHealthDescriptions": [
    {
      "Target": {
        "Id": "<target IP>",
        "Port": 8998,
        "AvailabilityZone": "us-west-2d"
      },
      "HealthCheckPort": "8998",
      "TargetHealth": {
        "State": "healthy"
      }
    }
  ]
}
```

Setelah status NLB Anda menjadi active dan kelompok target Anda healthy, Anda dapat melanjutkan. Mungkin butuh beberapa menit.

9. Ambil titik akhir Livy dari instalasi Helm. Apakah titik akhir Livy Anda aman atau tidak tergantung pada apakah Anda mengaktifkan SSL.

```
LIVY_NAMESPACE=<livy-ns>
LIVY_APP_NAME=livy-app-name
LIVY_ENDPOINT=$(kubectl get svc -n livy-ns -l app.kubernetes.io/
instance=livy-app-name,emr-containers.amazonaws.com/type=loadbalancer -o
jsonpath='{.items[0].status.loadBalancer.ingress[0].hostname}' | awk '{printf
"%s:8998\n", $0}')
echo "$LIVY_ENDPOINT"
```

10. Ambil akun layanan Spark dari instalasi Helm

```
SPARK_NAMESPACE=spark-ns
LIVY_APP_NAME=<livy-app-name>
SPARK_SERVICE_ACCOUNT=$(kubectl --namespace $SPARK_NAMESPACE
get sa -l "app.kubernetes.io/instance=$LIVY_APP_NAME" -o
jsonpath='{.items[0].metadata.name}')
echo "$SPARK_SERVICE_ACCOUNT"
```

Anda akan melihat sesuatu yang mirip dengan output berikut:

```
emr-containers-sa-spark-livy
```

11. Jika Anda mengatur `internalALB=true` untuk mengaktifkan akses dari luar VPC, buat instans Amazon EC2 dan pastikan Network Load Balancer memungkinkan lalu lintas jaringan yang berasal dari instans EC2. Anda harus melakukannya misalnya untuk memiliki akses ke titik akhir Livy Anda. Untuk informasi selengkapnya tentang mengekspos titik akhir Anda dengan aman di luar VPC Anda, lihat [Menyiapkan dengan titik akhir Apache Livy yang aman dengan TLS/SSL](#).
12. Menginstal Livy membuat akun layanan `emr-containers-sa-spark` untuk menjalankan aplikasi Spark. Jika aplikasi Spark Anda menggunakan AWS sumber daya apa pun seperti S3 atau memanggil operasi AWS API atau CLI, Anda harus menautkan peran IAM dengan izin yang diperlukan ke akun layanan spark Anda. Untuk informasi selengkapnya, lihat [Menyiapkan izin akses dengan peran IAM untuk akun layanan \(IRSA\)](#).

Apache Livy mendukung konfigurasi tambahan yang dapat Anda gunakan saat menginstal Livy. Untuk informasi selengkapnya, lihat Properti instalasi untuk Apache Livy di Amazon EMR pada rilis EKS.

Menjalankan aplikasi Spark dengan Apache Livy untuk Amazon EMR di EKS

Sebelum Anda dapat menjalankan aplikasi Spark dengan Apache Livy, pastikan bahwa Anda telah menyelesaikan langkah-langkah dalam [Menyiapkan Apache Livy untuk Amazon EMR di EKS dan Memulai dengan Apache Livy untuk Amazon EMR di EKS](#).

Anda dapat menggunakan Apache Livy untuk menjalankan dua jenis aplikasi:

- Batch session — jenis beban kerja Livy untuk mengirimkan pekerjaan batch Spark.
- Sesi interaktif — jenis beban kerja Livy yang menyediakan antarmuka terprogram dan visual untuk menjalankan kueri Spark.

Note

Pod driver dan eksekutor dari sesi yang berbeda dapat berkomunikasi satu sama lain. Namespace tidak menjamin keamanan apa pun di antara pod. Kubernetes tidak mengizinkan izin selektif pada subset pod di dalam namespace tertentu.

Menjalankan sesi batch

Untuk mengirimkan pekerjaan batch, gunakan perintah berikut.

```
curl -s -k -H 'Content-Type: application/json' -X POST \  
  -d '{  
    "name": "my-session",  
    "file": "entryPoint_location (S3 or local)",  
    "args": ["argument1", "argument2", ...],  
    "conf": {  
      "spark.kubernetes.namespace": "<spark-namespace>",  
      "spark.kubernetes.container.image": "public.ecr.aws/emr-on-eks/spark/  
emr-7.13.0:latest",  
      "spark.kubernetes.authenticate.driver.serviceAccountName": "<spark-  
service-account>"  
    }  
  }' <livy-endpoint>/batches
```

Untuk memantau pekerjaan batch Anda, gunakan perintah berikut.

```
curl -s -k -H 'Content-Type: application/json' -X GET <livy-endpoint>/batches/my-  
session
```

Menjalankan sesi interaktif

Untuk menjalankan sesi interaktif dengan Apache Livy, lihat langkah-langkah berikut.

1. Pastikan Anda memiliki akses ke notebook Jupyter yang dihosting sendiri atau yang dikelola, seperti notebook SageMaker AI Jupyter. Notebook jupyter Anda harus memiliki [sparkmagic](#) yang diinstal.
2. Buat ember untuk konfigurasi `spark.kubernetes.file.upload.path` Spark. Pastikan akun layanan Spark telah membaca dan menulis akses ke bucket. Untuk detail selengkapnya tentang cara mengonfigurasi akun layanan spark, lihat [Menyiapkan izin akses dengan peran IAM untuk akun layanan \(IRSA\)](#)
3. Muat `sparkmagic` di notebook Jupyter dengan perintah. `%load_ext sparkmagic.magics`
4. Jalankan perintah `%manage_spark` untuk mengatur titik akhir Livy Anda dengan notebook Jupyter. Pilih tab [Tambah Titik Akhir](#), pilih jenis autentikasi yang dikonfigurasi, tambahkan titik akhir Livy ke buku catatan, lalu pilih [Tambah titik akhir](#).

5. Jalankan `%manage_spark` lagi untuk membuat konteks Spark dan kemudian pergi ke sesi Create. Pilih titik akhir Livy, tentukan nama sesi unik pilih bahasa, lalu tambahkan properti berikut.

```
{
  "conf": {
    "spark.kubernetes.namespace": "livy-namespace",
    "spark.kubernetes.container.image": "public.ecr.aws/emr-on-eks/spark/
emr-7.13.0:latest",
    "spark.kubernetes.authenticate.driver.serviceAccountName": "<spark-service-
account>",
    "spark.kubernetes.file.upload.path": "<URI_TO_S3_LOCATION>"
  }
}
```

6. Kirim aplikasi dan tunggu sampai membuat konteks Spark.
7. Untuk memantau status sesi interaktif, jalankan perintah berikut.

```
curl -s -k -H 'Content-Type: application/json' -X GET livy-endpoint/sessions/my-
interactive-session
```

Memantau aplikasi Spark

Untuk memantau kemajuan aplikasi Spark Anda dengan Livy UI, gunakan tautan. `http://<livy-endpoint>/ui`

Mencopot instalasi Apache Livy dengan Amazon EMR di EKS

Ikuti langkah-langkah ini untuk menghapus instalasi Apache Livy.

1. Hapus pengaturan Livy menggunakan nama namespace dan nama aplikasi Anda. Dalam contoh ini, nama aplikasi adalah `livy-demo` dan namespace adalah `livy-ns`

```
helm uninstall livy-demo -n livy-ns
```

2. Saat mencopot pemasangan, Amazon EMR di EKS menghapus layanan Kubernetes di Livy, penyeimbang beban, dan grup AWS target yang Anda buat selama instalasi. Menghapus sumber daya dapat memakan waktu beberapa menit. Pastikan sumber daya dihapus sebelum menginstal Livy di namespace lagi.

3. Hapus namespace Spark.

```
kubectl delete namespace spark-ns
```

Keamanan untuk Apache Livy dengan Amazon EMR di EKS

Lihat topik berikut untuk mempelajari lebih lanjut tentang mengonfigurasi keamanan Apache Livy dengan Amazon EMR di EKS. Opsi ini termasuk menggunakan keamanan lapisan transportasi, kontrol akses berbasis peran, yang merupakan akses berdasarkan peran seseorang dalam suatu organisasi, dan menggunakan peran IAM, yang menyediakan akses ke sumber daya, berdasarkan izin yang diberikan.

Topik

- [Menyiapkan endpoint Apache Livy yang aman dengan TLS/SSL](#)
- [Menyiapkan izin aplikasi Apache Livy dan Spark dengan kontrol akses berbasis peran \(RBAC\)](#)
- [Menyiapkan izin akses dengan peran IAM untuk akun layanan \(IRSA\)](#)

Menyiapkan endpoint Apache Livy yang aman dengan TLS/SSL

Lihat bagian berikut untuk mempelajari lebih lanjut tentang pengaturan Apache Livy untuk Amazon EMR di EKS dengan enkripsi end-to-end TLS dan SSL.

Menyiapkan enkripsi TLS dan SSL

Untuk mengatur enkripsi SSL pada endpoint Apache Livy Anda, ikuti langkah-langkah berikut.

- [Instal Secrets Store CSI Driver dan AWS Secrets and Configuration Provider \(ASCP\)](#) — Secrets Store CSI Driver dan ASCP dengan aman menyimpan sertifikat dan kata sandi JKS Livy yang dibutuhkan pod server Livy untuk mengaktifkan SSL. Anda juga dapat menginstal hanya Secrets Store CSI Driver dan menggunakan penyedia rahasia lain yang didukung.
- [Buat sertifikat ACM — sertifikat](#) ini diperlukan untuk mengamankan koneksi antara klien dan titik akhir ALB.
- Siapkan sertifikat JKS, kata sandi kunci, dan kata sandi keystore untuk AWS Secrets Manager — diperlukan untuk mengamankan koneksi antara titik akhir ALB dan server Livy.
- Tambahkan izin ke akun layanan Livy untuk mengambil rahasia dari AWS Secrets Manager — server Livy memerlukan izin ini untuk mengambil rahasia dari ASCP dan menambahkan konfigurasi

Livy untuk mengamankan server Livy. Untuk menambahkan izin IAM ke akun layanan, lihat [Menyiapkan izin akses dengan peran IAM untuk akun layanan \(IRSA\)](#).

Menyiapkan sertifikat JKS dengan kunci dan kata sandi keystore untuk AWS Secrets Manager

Ikuti langkah-langkah ini untuk menyiapkan sertifikat JKS dengan kunci dan kata sandi keystore.

1. Hasilkan file keystore untuk server Livy.

```
keytool -genkey -alias <host> -keyalg RSA -keysize 2048 -dname
  CN=<host>,OU=hw,O=hw,L=<your_location>,ST=<state>,C=<country> -
keypass <keyPassword> -keystore <keystore_file> -storepass <storePassword> --
  validity 3650
```

2. Buat sertifikat.

```
keytool -export -alias <host> -keystore mykeystore.jks -rfc -
  file mycertificate.cert -storepass <storePassword>
```

3. Buat file truststore.

```
keytool -import -noprompt -alias <host>-file <cert_file> -
  keystore <truststore_file> -storepass <truststorePassword>
```

4. Simpan sertifikat JKS di AWS Secrets Manager. Ganti `livy-jks-secret` dengan rahasia Anda dan `fileb://mykeystore.jks` dengan jalur ke sertifikat JKS keystore Anda.

```
aws secretsmanager create-secret \
  --name livy-jks-secret \
  --description "My Livy keystore JKS secret" \
  --secret-binary fileb://mykeystore.jks
```

5. Simpan keystore dan kata sandi kunci di Secrets Manager. Pastikan untuk menggunakan parameter Anda sendiri.

```
aws secretsmanager create-secret \
  --name livy-jks-secret \
  --description "My Livy key and keystore password secret" \
  --secret-string "{\"keyPassword\": \"<test-key-password>\", \"keyStorePassword\": \"<test-key-store-password>\"}"
```

6. Buat namespace server Livy dengan perintah berikut.

```
kubectl create ns <livy-ns>
```

7. Buat ServiceProviderClass objek untuk server Livy yang memiliki sertifikat JKS dan kata sandi.

```
cat >livy-secret-provider-class.yaml << EOF
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: aws-secrets
spec:
  provider: aws
  parameters:
    objects: |
      - objectName: "livy-jks-secret"
        objectType: "secretsmanager"
      - objectName: "livy-passwords"
        objectType: "secretsmanager"

EOF
kubectl apply -f livy-secret-provider-class.yaml -n <livy-ns>
```

Memulai dengan Apache Livy yang mendukung SSL

Setelah mengaktifkan SSL di server Livy Anda, Anda harus mengatur serviceAccount agar memiliki akses ke keyStore dan keyPasswords rahasia. AWS Secrets Manager

1. Buat namespace server Livy.

```
kubectl create namespace <livy-ns>
```

2. Siapkan akun layanan Livy untuk memiliki akses ke rahasia di Secrets Manager. Untuk informasi selengkapnya tentang pengaturan IRSA, lihat [Menyiapkan IRSA saat menginstal Apache Livy](#).

```
aws ecr get-login-password --region region-id | helm registry login \
--username AWS \
--password-stdin ECR-registry-account.dkr.ecr.region-id.amazonaws.com
```

3. Instal Livy. Untuk parameter Helm chart `--version`, gunakan label rilis Amazon EMR Anda, seperti `7.1.0` Anda juga harus mengganti ID akun registri Amazon ECR dan ID Wilayah dengan ID Anda sendiri IDs. Anda dapat menemukan ECR-registry-account nilai yang sesuai untuk [akun Wilayah AWS registri Amazon ECR Anda berdasarkan Wilayah](#).

```
helm install <livy-app-name> \
  oci://895885662937.dkr.ecr.region-id.amazonaws.com/livy \
  --version 7.13.0 \
  --namespace livy-namespace-name \
  --set image=<ECR-registry-account.dkr.ecr>.<region>.amazonaws.com/livy/
emr-7.13.0:latest \
  --set sparkNamespace=spark-namespace \
  --set ssl.enabled=true
  --set ssl.CertificateArn=livy-acm-certificate-arn
  --set ssl.secretProviderClassName=aws-secrets
  --set ssl.keyStoreObjectName=livy-jks-secret
  --set ssl.keyPasswordsObjectName=livy-passwords
  --create-namespace
```

4. Lanjutkan dari langkah 5 dari [Menginstal Apache Livy di Amazon EMR](#) di EKS.

Menyiapkan izin aplikasi Apache Livy dan Spark dengan kontrol akses berbasis peran (RBAC)

Untuk menyebarkan Livy, Amazon EMR di EKS membuat akun dan peran layanan server serta akun dan peran layanan Spark. Peran ini harus memiliki izin RBAC yang diperlukan untuk menyelesaikan pengaturan dan menjalankan aplikasi Spark.

Izin RBAC untuk akun dan peran layanan server

Amazon EMR di EKS membuat akun layanan server Livy dan berperan untuk mengelola sesi Livy untuk pekerjaan Spark dan merutekan lalu lintas ke dan dari ingress dan sumber daya lainnya.

Nama default untuk akun layanan ini adalah `emr-containers-sa-livy`. Itu harus memiliki izin berikut.

```
rules:
- apiGroups:
  - ""
  resources:
  - "namespaces"
```

```
verbs:
- "get"
- apiGroups:
- ""
resources:
- "serviceaccounts"
  "services"
  "configmaps"
  "events"
  "pods"
  "pods/log"
verbs:
- "get"
  "list"
  "watch"
  "describe"
  "create"
  "edit"
  "delete"
  "deletecollection"
  "annotate"
  "patch"
  "label"
- apiGroups:
- ""
resources:
- "secrets"
verbs:
- "create"
  "patch"
  "delete"
  "watch"
- apiGroups:
- ""
resources:
- "persistentvolumeclaims"
verbs:
- "get"
  "list"
  "watch"
  "describe"
  "create"
  "edit"
  "delete"
```

```
"annotate"  
"patch"  
"label"
```

Izin RBAC untuk akun dan peran layanan spark

Pod driver Spark membutuhkan akun layanan Kubernetes di namespace yang sama dengan pod. Akun layanan ini memerlukan izin untuk mengelola pod pelaksana dan sumber daya apa pun yang diperlukan oleh pod driver. Kecuali akun layanan default di namespace memiliki izin yang diperlukan, driver gagal dan keluar. Izin RBAC berikut diperlukan.

```
rules:  
- apiGroups:  
  - ""  
    "batch"  
    "extensions"  
    "apps"  
  resources:  
  - "configmaps"  
    "serviceaccounts"  
    "events"  
    "pods"  
    "pods/exec"  
    "pods/log"  
    "pods/portforward"  
    "secrets"  
    "services"  
    "persistentvolumeclaims"  
    "statefulsets"  
  verbs:  
  - "create"  
    "delete"  
    "get"  
    "list"  
    "patch"  
    "update"  
    "watch"  
    "describe"  
    "edit"  
    "deletecollection"  
    "patch"  
    "label"
```

Menyiapkan izin akses dengan peran IAM untuk akun layanan (IRSA)

Secara default, server Livy dan driver dan pelaksana aplikasi Spark tidak memiliki akses ke sumber daya. AWS Akun layanan server dan akun layanan spark mengontrol akses ke AWS sumber daya untuk server Livy dan pod aplikasi spark. Untuk memberikan akses, Anda perlu memetakan akun layanan dengan peran IAM yang memiliki AWS izin yang diperlukan.

Anda dapat mengatur pemetaan IRSA sebelum Anda menginstal Apache Livy, selama instalasi, atau setelah Anda menyelesaikan instalasi.

Menyiapkan IRSA saat menginstal Apache Livy (untuk akun layanan server)

Note

Pemetaan ini hanya didukung untuk akun layanan server.

1. Pastikan Anda telah selesai [menyiapkan Apache Livy untuk Amazon EMR di EKS](#) dan sedang [menginstal Apache Livy dengan Amazon EMR di EKS](#).
2. Buat namespace Kubernetes untuk server Livy. Dalam contoh ini, nama namespace adalah `livy-ns`
3. Buat kebijakan IAM yang menyertakan izin untuk yang Anda inginkan Layanan AWS untuk mengakses pod Anda. Contoh berikut membuat kebijakan IAM untuk mendapatkan sumber daya Amazon S3 untuk titik masuk Spark.

```
cat >my-policy.json <<EOF{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::my-spark-entrypoint-bucket"
    }
  ]
}
EOF

aws iam create-policy --policy-name my-policy --policy-document file://my-policy.json
```

- Gunakan perintah berikut untuk mengatur Akun AWS ID Anda ke variabel.

```
account_id=$(aws sts get-caller-identity --query "Account" --output text)
```

- Setel penyedia identitas OpenID Connect (OIDC) cluster Anda ke variabel lingkungan.

```
oidc_provider=$(aws eks describe-cluster --name my-cluster --region $AWS_REGION --query "cluster.identity.oidc.issuer" --output text | sed -e "s/^https:\\\\\\")
```

- Tetapkan variabel untuk namespace dan nama akun layanan. Pastikan untuk menggunakan nilai Anda sendiri.

```
export namespace=default
export service_account=my-service-account
```

- Buat file kebijakan kepercayaan dengan perintah berikut. Jika Anda ingin memberikan akses peran ke semua akun layanan dalam namespace, salin perintah berikut, dan ganti dengan dan ganti `StringEquals` dengan `StringLike`. `$service_account *`

```
cat >trust-relationship.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::$account_id:oidc-provider/$oidc_provider"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "$oidc_provider:aud": "sts.amazonaws.com",
          "$oidc_provider:sub": "system:serviceaccount:$namespace:$service_account"
        }
      }
    }
  ]
}
EOF
```

- Buat peran.

```
aws iam create-role --role-name my-role --assume-role-policy-document file://trust-relationship.json --description "my-role-description"
```

- Gunakan perintah Helm install berikut untuk mengatur untuk memetakan IRSAserviceAccount.executionRoleArn. Berikut ini adalah contoh perintah Helm install. Anda dapat menemukan ECR-registry-account nilai yang sesuai untuk [akun Wilayah AWS registri Amazon ECR Anda berdasarkan Wilayah](#).

```
helm install livy-demo \
  oci://895885662937.dkr.ecr.us-west-2.amazonaws.com/livy \
  --version 7.13.0 \
  --namespace livy-ns \
  --set image=ECR-registry-account.dkr.ecr.region-id.amazonaws.com/livy/emr-7.13.0:latest \
  --set sparkNamespace=spark-ns \
  --set serviceAccount.executionRoleArn=arn:aws:iam::123456789012:role/my-role
```

Memetakan IRSA ke akun layanan Spark

Sebelum Anda memetakan IRSA ke akun layanan Spark, pastikan Anda telah menyelesaikan item berikut:

- Pastikan Anda telah selesai [menyiapkan Apache Livy untuk Amazon EMR di EKS](#) dan sedang [menginstal Apache Livy dengan Amazon EMR di EKS](#).
- Anda harus memiliki provdider IAM OpenID Connect (OIDC) yang ada untuk klaster Anda. Untuk melihat apakah Anda sudah memilikinya atau cara membuatnya, lihat [Membuat penyedia IAM OIDC untuk](#) klaster Anda.
- Pastikan Anda telah menginstal versi 0.171.0 atau yang lebih baru dari eksctl CLI yang diinstal atau. AWS CloudShell Untuk menginstal atau memperbaruieksctl, lihat [Instalasi eksctl](#) dokumentasi.

Ikuti langkah-langkah berikut untuk memetakan IRSA ke akun layanan Spark Anda:

- Gunakan perintah berikut untuk mendapatkan akun layanan Spark.

```
SPARK_NAMESPACE=<spark-ns>
LIVY_APP_NAME=<livy-app-name>
```

```
kubectl --namespace $SPARK_NAMESPACE describe sa -l "app.kubernetes.io/instance=$LIVY_APP_NAME" | awk '/^Name:/ {print $2}'
```

2. Tetapkan variabel Anda untuk namespace dan nama akun layanan.

```
export namespace=default
export service_account=my-service-account
```

3. Gunakan perintah berikut untuk membuat file kebijakan kepercayaan untuk peran IAM. Contoh berikut memberikan izin ke semua akun layanan dalam namespace untuk menggunakan peran. Untuk melakukannya, ganti `StringEquals` dengan `StringLike` dan ganti `$service_account` dengan*.

```
cat >trust-relationship.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::$account_id:oidc-provider/$oidc_provider"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "$oidc_provider:aud": "sts.amazonaws.com",
          "$oidc_provider:sub": "system:serviceaccount:$namespace:$service_account"
        }
      }
    }
  ]
}
EOF
```

4. Buat peran.

```
aws iam create-role --role-name my-role --assume-role-policy-document file://trust-relationship.json --description "my-role-description"
```

5. Petakan server atau akun layanan spark dengan `eksctl` perintah berikut. Pastikan untuk menggunakan nilai Anda sendiri.

```
eksctl create iamserviceaccount --name spark-sa \
--namespace spark-namespace --cluster livy-eks-cluster \
--attach-role-arn arn:aws:iam::0123456789012:role/my-role \
--approve --override-existing-serviceaccounts
```

Properti instalasi untuk Apache Livy di Amazon EMR pada rilis EKS

Instalasi Apache Livy memungkinkan Anda memilih versi bagan Livy Helm. Bagan Helm menawarkan berbagai properti untuk menyesuaikan pengalaman instalasi dan penyiapan Anda. Properti ini didukung untuk Amazon EMR pada rilis EKS 7.1.0 dan yang lebih tinggi.

Topik

- [Properti instalasi Amazon EMR 7.1.0](#)

Properti instalasi Amazon EMR 7.1.0

Tabel berikut menjelaskan semua properti Livy yang didukung. Saat menginstal Apache Livy, Anda dapat memilih versi bagan Livy Helm. Untuk mengatur properti selama instalasi, gunakan perintah `--set <property>=<value>`.

Properti	Deskripsi	Default
gambar	URI rilis EMR Amazon dari server Livy. Ini adalah konfigurasi yang diperlukan.	""
SparkNamespace	Namespace untuk menjalankan sesi Livy Spark. Misalnya, tentukan "livy". Ini adalah konfigurasi yang diperlukan.	""
NameOverride	Berikan nama, bukan <code>livy</code> . Nama ditetapkan sebagai label untuk semua sumber daya Livy	"hidup"

Properti	Deskripsi	Default
FullNameOverride	Berikan nama untuk digunakan alih-alih nama lengkap sumber daya.	""
ssl.enabled	Mengaktifkan end-to-end SSL dari titik akhir Livy ke server Livy.	SALAH
SSL.CertificatEarn	Jika SSL diaktifkan, ini adalah sertifikat ACM ARN untuk NLB yang dibuat oleh layanan..	""
ssl. secretProviderClassNama	Jika SSL diaktifkan, ini adalah nama kelas penyedia rahasia untuk mengamankan NLB untuk koneksi server Livy dengan SSL.	""
ssl. keyStoreObjectNama	Jika SSL diaktifkan, nama objek untuk sertifikat keystore di kelas penyedia rahasia.	""
ssl. keyPasswordsObject Nama	Jika SSL diaktifkan, nama objek untuk rahasia yang memiliki keystore dan kata sandi kunci.	""
rbac.create	Jika benar, membuat sumber daya RBAC.	SALAH
ServiceAccount.Create	Jika benar, buat akun layanan Livy.	BETUL

Properti	Deskripsi	Default
ServiceAccount.name	Nama akun layanan yang akan digunakan untuk Livy. Jika Anda tidak menyetel properti ini dan membuat akun layanan, Amazon EMR di EKS secara otomatis menghasilkan nama menggunakan properti <code>fullname override</code> .	"emr-containers-sa-livy"
ServiceAccount.executionRoleArn	Peran eksekusi ARN dari akun layanan Livy.	""
sparkServiceAccount.buat	JIKA benar, buat akun layanan Spark di <code>.Release.Namespace</code>	BETUL
sparkServiceAccount.nama	Nama akun layanan yang akan digunakan untuk Spark. Jika Anda tidak menyetel properti ini dan membuat akun layanan Spark, Amazon EMR di EKS secara otomatis menghasilkan nama dengan properti <code>-spark-livy</code> dengan <code>fullname0 verride</code> akhiran.	"emr-containers-sa-spark-hidup"
service.name	Nama layanan Livy	"emr-containers-livy"
service.annotations	Anotasi layanan Livy	{}

Properti	Deskripsi	Default
loadbalancer.enabled	Apakah akan membuat penyeimbang beban untuk layanan Livy yang digunakan untuk mengekspos titik akhir Livy di luar klaster Amazon EKS.	FALSE
loadbalancer.internal	Apakah akan mengonfigurasi titik akhir Livy sebagai internal ke VPC atau eksternal. Menyetel properti ini untuk FALSE mengekspos titik akhir ke sumber di luar VPC. Kami merekomendasikan mengamankan titik akhir Anda dengan TLS/SSL. Untuk informasi selengkapnya, lihat Menyiapkan enkripsi TLS dan SSL .	FALSE
imagePullSecrets	Daftar <code>imagePullSecret</code> nama yang akan digunakan untuk menarik gambar Livy dari repositori pribadi.	[]
sumber daya	Permintaan sumber daya dan batasan untuk kontainer Livy.	{}
NodeSelector	Node untuk menjadwalkan pod Livy.	{}
toleransi	Daftar yang berisi toleransi Pod Livy untuk didefinisikan.	[]
afinitas	Aturan afinitas pod Livy.	{}

Properti	Deskripsi	Default
<code>persistence.enabled</code>	Jika benar, aktifkan persistensi untuk direktori sesi.	SALAH
<code>Persistence.subpath</code>	Subpath PVC untuk dipasang ke direktori sesi.	""
<code>Persistence.ExistingClaim</code>	PVC untuk digunakan alih-alih membuat yang baru.	{}
<code>Persistence.storageClass</code>	Kelas penyimpanan yang akan digunakan. Untuk menentukan parameter ini, gunakan formatnya <code>storageClassName: <storageClass></code> . Menyetel parameter ini untuk "-" menonaktifkan penyediaan dinamis. Jika Anda menyetel parameter ini ke null atau tidak menentukan apa pun, Amazon EMR di EKS tidak menyetel <code>storageClassName</code> dan menggunakan penyedia default.	""
<code>Persistence.accessMode</code>	Mode akses PVC.	<code>ReadWriteOnce</code>
kegigihan. Ukuran	Ukuran PVC.	20Gi
<code>persistence.annotations</code>	Anotasi tambahan untuk PVC.	{}
<code>env. *</code>	Env tambahan untuk disetel ke wadah Livy. Untuk informasi selengkapnya, lihat Memasukkan konfigurasi Livy dan Spark Anda sendiri saat menginstal Livy.	{}

Properti	Deskripsi	Default
EnvFrom. *	Env tambahan untuk disetel ke Livy dari peta konfigurasi Kubernetes atau rahasia.	[]
LivyConf. *	Entri livy.conf tambahan untuk disetel dari peta konfigurasi Kubernetes atau rahasia yang dipasang.	{}
sparkDefaultsConf.*	spark-defaults.conf Entri tambahan untuk disetel dari peta konfigurasi Kubernetes atau rahasia yang dipasang.	{}

Memecahkan masalah kesalahan format variabel lingkungan yang umum

Saat Anda memasukkan konfigurasi Livy dan Spark, ada format variabel lingkungan yang tidak didukung dan dapat menyebabkan kesalahan. Prosedur ini membawa Anda melalui serangkaian langkah untuk membantu memastikan bahwa Anda menggunakan format yang benar.

Memasukkan konfigurasi Livy dan Spark Anda sendiri saat menginstal Livy

Anda dapat mengonfigurasi variabel lingkungan Apache Livy atau Apache Spark dengan properti Helm. `env.*` Ikuti langkah-langkah di bawah ini untuk mengonversi konfigurasi contoh `example.config.with-dash.withUppercase` ke format variabel lingkungan yang didukung.

1. Ganti huruf besar dengan huruf 1 dan huruf kecil. Misalnya, `example.config.with-dash.withUppercase` menjadi `example.config.with-dash.with1uppercase`.
2. Ganti tanda hubung (-) dengan 0. Sebagai contoh, `example.config.with-dash.with1uppercase` menjadi `example.config.with0dash.with1uppercase`
3. Ganti titik (.) dengan garis bawah (_). Misalnya, `example.config.with0dash.with1uppercase` menjadi `example_config_with0dash_with1uppercase`.
4. Ganti semua huruf kecil dengan huruf besar.

5. Tambahkan awalan LIVY_ ke nama variabel.
6. Gunakan variabel saat menginstal Livy melalui bagan helm menggunakan format `--set env.YOUR_VARIABLE_NAME.nilai = yourvalue`

Misalnya, untuk mengatur konfigurasi Livy dan Spark `livy.server.recovery.state-store = filesystem` dan `spark.kubernetes.executor.podNamePrefix = my-prefix`, gunakan properti Helm ini:

```
--set env.LIVY_LIVY_SERVER_RECOVERY_STATESTORE.value=filesystem
--set env.LIVY_SPARK_KUBERNETES_EXECUTOR_PODNAMEPREFIX.value=myprefix
```

Mengelola Amazon EMR pada pekerjaan EKS

Bagian berikut mencakup topik yang membantu Anda mengelola EMR Amazon Anda di pekerjaan EKS. Ini termasuk mengonfigurasi parameter job run saat Anda menggunakan AWS CLI, mengonfigurasi bagaimana data log Anda disimpan, menjalankan skrip Spark SQL untuk menjalankan kueri, memahami status job run, dan mengetahui cara memantau pekerjaan. Anda dapat mengerjakan topik-topik ini, umumnya secara berurutan, jika Anda ingin mengatur dan menyelesaikan pekerjaan untuk memproses data.

Topik

- [Mengelola pekerjaan berjalan dengan AWS CLI](#)
- [Menjalankan skrip Spark SQL melalui API StartJobRun](#)
- [Status tugas berjalan](#)
- [Melihat tugas di konsol Amazon EMR](#)
- [Kesalahan umum saat menjalankan tugas](#)

Mengelola pekerjaan berjalan dengan AWS CLI

Topik ini mencakup cara mengelola pekerjaan berjalan dengan AWS Command Line Interface (AWS CLI). Ini masuk ke detail mengenai properti, seperti parameter keamanan, driver, dan berbagai pengaturan penggantian. Ini juga mencakup subtopik yang mencakup berbagai cara untuk mengonfigurasi logging.

Topik

- [Pilihan untuk mengonfigurasi tugas berjalan](#)
- [Konfigurasi pekerjaan yang dijalankan untuk menggunakan log Amazon S3](#)
- [Mengonfigurasi pekerjaan yang dijalankan untuk menggunakan Amazon CloudWatch Logs](#)
- [Daftar tugas berjalan](#)
- [Jelaskan tugas berjalan](#)
- [Membatalkan tugas berjalan](#)

Pilihan untuk mengonfigurasi tugas berjalan

Gunakan opsi berikut untuk mengonfigurasi parameter tugas berjalan:

- `--execution-role-arn`: Anda harus menyediakan IAM role yang digunakan untuk menjalankan tugas. Untuk informasi selengkapnya, lihat [Menggunakan peran eksekusi tugas dengan Amazon EMR di EKS](#).
- `--release-label`: Anda dapat menyebarkan Amazon EMR di EKS dengan Amazon EMR versi 5.32.0 dan 6.2.0 dan lebih baru. Amazon EMR di EKS tidak didukung dalam versi rilis Amazon EMR sebelumnya. Untuk informasi selengkapnya, lihat [Amazon EMR pada rilis EKS](#).
- `--job-driver`: Driver tugas digunakan untuk memberikan input pada tugas utama. Ini adalah bidang jenis serikat di mana Anda hanya dapat meloloskan salah satu nilai untuk jenis tugas yang ingin Anda jalankan. Jenis tugas yang didukung meliputi:
 - Tugas Spark submit - Digunakan untuk menjalankan perintah melalui Spark submit. Anda dapat menggunakan jenis pekerjaan ini untuk menjalankan Scala, SparkR PySpark, SparkSQL, dan pekerjaan lain yang didukung melalui Spark Submit. Tugas ini memiliki parameter berikut:
 - `Entrypoint` - Ini adalah referensi HCFS (sistem file yang kompatibel dengan Hadoop) ke file utama jar/py yang ingin Anda jalankan.
 - `EntryPointArguments` - Ini adalah array argumen yang ingin Anda lewatkan ke jar/py file utama Anda. Anda harus menangani membaca parameter ini menggunakan kode entrypoint Anda. Setiap argumen dalam array harus dipisahkan dengan koma. `EntryPointArguments` tidak dapat berisi tanda kurung atau tanda kurung, seperti `()`, `{}`, atau `[]`.
 - `SparkSubmitParameters` - Ini adalah parameter percikan tambahan yang ingin Anda kirim ke pekerjaan. Gunakan parameter ini untuk menimpa properti default Spark seperti memori driver atau jumlah pelaksana seperti `—conf` atau `—class`. Untuk informasi tambahan, lihat [Peluncuran Aplikasi dengan spark-submit](#).

- Spark SQL jobs - Digunakan untuk menjalankan file query SQL melalui Spark SQL. Anda dapat menggunakan jenis pekerjaan ini untuk menjalankan pekerjaan SparkSQL. Tugas ini memiliki parameter berikut:
 - Entrypoint - Ini adalah referensi HCFS (sistem file yang kompatibel dengan Hadoop) ke file query SQL yang ingin Anda jalankan.

Untuk daftar parameter Spark tambahan yang dapat Anda gunakan untuk pekerjaan Spark SQL, lihat [Menjalankan skrip Spark SQL melalui API StartJobRun](#)

- `--configuration-overrides`: Anda dapat menimpa konfigurasi default untuk aplikasi dengan menyediakan objek konfigurasi. Anda dapat menggunakan sintaks singkatan untuk menyediakan konfigurasi atau Anda dapat mereferensikan objek konfigurasi dalam file JSON. Objek konfigurasi terdiri dari klasifikasi, properti, dan konfigurasi bersarang opsional. Properti terdiri dari pengaturan yang ingin Anda timpa dalam file tersebut. Anda dapat menentukan beberapa klasifikasi untuk beberapa aplikasi dalam objek JSON tunggal. Klasifikasi konfigurasi yang tersedia bervariasi berdasarkan versi rilis Amazon EMR. Untuk daftar klasifikasi konfigurasi yang tersedia untuk setiap versi rilis Amazon EMR, lihat [Amazon EMR pada rilis EKS](#).

Jika Anda melewati konfigurasi yang sama dalam penimpanan aplikasi dan di parameter kirim Spark, parameter kirim Spark diutamakan. Daftar prioritas konfigurasi lengkap mengikuti, dalam urutan prioritas tertinggi ke prioritas terendah.

- Konfigurasi disediakan saat membuat `SparkSession`.
- Konfigurasi disediakan sebagai bagian dari `sparkSubmitParameters` menggunakan `-conf`.
- Konfigurasi disediakan sebagai bagian dari penimpanan aplikasi.
- Konfigurasi yang dioptimalkan dipilih oleh Amazon EMR untuk rilis.
- Konfigurasi sumber terbuka default untuk aplikasi.

Untuk memantau pekerjaan berjalan menggunakan Amazon CloudWatch atau Amazon S3, Anda harus memberikan detail konfigurasi untuk. CloudWatch Untuk informasi selengkapnya, lihat [Konfigurasi pekerjaan yang dijalankan untuk menggunakan log Amazon S3](#) dan [Mengonfigurasi pekerjaan yang dijalankan untuk menggunakan Amazon CloudWatch Logs](#). Jika bucket S3 atau grup CloudWatch log tidak ada, Amazon EMR membuatnya sebelum mengunggah log ke bucket.

- Untuk daftar tambahan opsi konfigurasi Kubernetes, lihat [Properti Spark di Kubernetes](#).

Konfigurasi Spark berikut tidak didukung.

- `spark.kubernetes.authenticate.driver.serviceAccountName`
- `spark.kubernetes.authenticate.executor.serviceAccountName`

- `spark.kubernetes.namespace`
- `spark.kubernetes.driver.pod.name`
- `spark.kubernetes.container.image.pullPolicy`
- `spark.kubernetes.container.image`

Note

Anda dapat menggunakan `spark.kubernetes.container.image` untuk gambar Docker yang disesuaikan. Untuk informasi selengkapnya, lihat [Menyesuaikan gambar Docker untuk Amazon EMR di EKS](#).

Konfigurasi pekerjaan yang dijalankan untuk menggunakan log Amazon S3

Untuk dapat memantau kemajuan pekerjaan dan memecahkan masalah kegagalan, Anda harus mengonfigurasi pekerjaan Anda untuk mengirim informasi log ke Amazon S3, Amazon Log, CloudWatch atau keduanya. Topik ini membantu Anda mulai menerbitkan log aplikasi ke Amazon S3 pada pekerjaan Anda yang diluncurkan dengan Amazon EMR di EKS.

S3 log kebijakan IAM

Sebelum tugas Anda dapat mengirim data log ke Amazon S3, izin berikut harus disertakan dalam kebijakan perizinan untuk peran eksekusi tugas. Ganti `amzn-s3-demo-logging-bucket` dengan nama bucket logging Anda.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket",

```

```

    "arn:aws:s3:::amzn-s3-demo-bucket/*"
  ],
  "Sid": "AllowS3Putobject"
}
]
}

```

Note

Amazon EMR di EKS juga dapat membuat bucket Amazon S3. Jika bucket Amazon S3 tidak tersedia, sertakan “s3:CreateBucket” izin tersebut dalam kebijakan IAM.

Setelah Anda memberi peran eksekusi izin yang tepat untuk mengirim log ke Amazon S3, data log Anda dikirim ke lokasi Amazon S3 berikut s3MonitoringConfiguration saat diteruskan di monitoringConfiguration bagian permintaan, seperti start-job-run yang ditunjukkan pada.

[Mengelola pekerjaan berjalan dengan AWS CLI](#)

- Log Pengirim `--virtual-cluster-id/jobs/ /containers/logUri/(job-id)stderr.gz/stdout.gz pod-name`
- Log Driver `--virtual-cluster-id/jobs/ logUri job-id /containers/ /spark- -driver/ (spark-application-id)stderr.gz/stdout.gz job-id`
- Log Pelaksana `--virtual-cluster-id/jobs/ logUri job-id /containers///(spark-application-id)stderr.gz/stdout.gz executor-pod-name`

Mengonfigurasi pekerjaan yang dijalankan untuk menggunakan Amazon CloudWatch Logs

Untuk memantau kemajuan pekerjaan dan memecahkan masalah kegagalan, Anda harus mengonfigurasi pekerjaan Anda untuk mengirim informasi log ke Amazon S3, Amazon Log, CloudWatch atau keduanya. Topik ini membantu Anda mulai menggunakan CloudWatch Log pada pekerjaan Anda yang diluncurkan dengan Amazon EMR di EKS. Untuk informasi selengkapnya tentang CloudWatch Log, lihat [Memantau File Log](#) di Panduan CloudWatch Pengguna Amazon.

CloudWatch Log kebijakan IAM

Agar pekerjaan Anda dapat mengirim data CloudWatch log ke Log, izin berikut harus disertakan dalam kebijakan izin untuk peran eksekusi pekerjaan. Ganti *my_log_group_name* dan *my_log_stream_prefix* dengan nama grup CloudWatch log dan nama aliran log Anda, masing-masing. Amazon EMR di EKS menciptakan grup log dan stream log jika mereka tidak ada selama peran eksekusi ARN memiliki izin yang sesuai.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogStream",
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams"
      ],
      "Resource": [
        "arn:aws:logs:*:*:*"
      ],
      "Sid": "AllowLOGSCreatelogstream"
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:PutLogEvents"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:my_log_group_name:log-
stream:my_log_stream_prefix/*"
      ],
      "Sid": "AllowLOGSPutlogevents"
    }
  ]
}
```

Note

Amazon EMR di EKS juga dapat membuat aliran log. Jika aliran log tidak ada, kebijakan IAM harus menyertakan "logs:CreateLogGroup" izin.

Setelah Anda memberikan izin yang tepat pada peran eksekusi, aplikasi Anda mengirimkan data lognya ke CloudWatch Log saat `cloudWatchMonitoringConfiguration` diteruskan di `monitoringConfiguration` bagian `start-job-run` permintaan, seperti yang ditunjukkan pada [Mengelola pekerjaan berjalan dengan AWS CLI](#).

Di `StartJobRun` API, `log_group_name` adalah nama grup log untuk CloudWatch, dan `log_stream_prefix` merupakan awalan nama aliran log untuk CloudWatch. Anda dapat melihat dan mencari log ini di Konsol Manajemen AWS.

- Log pengirim `--virtual-cluster-id/jobs/ logGroup /containers/logStreamPrefix/(stderr/ stdout job-id) pod-name`
- Log driver `-logGroup//virtual-cluster-id/jobs/ logStreamPrefix job-id /containers/ / spark- -driver/ spark-application-id (stderrstdout) job-id`
- Log pelaksana `-logGroup//virtual-cluster-id/jobs/ logStreamPrefix job-id / containers///(stderr/stdoutspark-application-id) executor-pod-name`

Daftar tugas berjalan

Anda dapat menjalankan `list-job-run` untuk menunjukkan keadaan tugas berjalan, seperti yang ditunjukkan contoh berikut.

```
aws emr-containers list-job-runs --virtual-cluster-id <cluster-id>
```

Jelaskan tugas berjalan

Anda dapat menjalankan `describe-job-run` untuk mendapatkan detail lebih lanjut tentang tugas, seperti status tugas, detail tugas, dan nama tugas, seperti yang ditunjukkan contoh berikut.

```
aws emr-containers describe-job-run --virtual-cluster-id cluster-id --id job-run-id
```

Membatalkan tugas berjalan

Anda dapat menjalankan `cancel-job-run` untuk membatalkan tugas berjalan, seperti yang ditunjukkan contoh berikut.

```
aws emr-containers cancel-job-run --virtual-cluster-id cluster-id --id job-run-id
```

Menjalankan skrip Spark SQL melalui API StartJobRun

Amazon EMR di EKS merilis 6.7.0 dan yang lebih tinggi menyertakan driver pekerjaan Spark SQL sehingga Anda dapat menjalankan skrip Spark SQL melalui API. `StartJobRun` Anda dapat menyediakan file entry-point SQL untuk langsung menjalankan kueri Spark SQL di Amazon EMR di EKS `StartJobRun` dengan API, tanpa modifikasi apa pun pada skrip Spark SQL yang ada. Tabel berikut mencantumkan parameter Spark yang didukung untuk pekerjaan Spark SQL melalui API. `StartJobRun`

Anda dapat memilih dari parameter Spark berikut untuk dikirim ke pekerjaan Spark SQL. Gunakan parameter ini untuk mengganti properti Spark default.

Opsi	Deskripsi
<code>--nama NAMA</code>	Nama Aplikasi
<code>--guci toples</code>	Daftar stoples dipisahkan koma untuk disertakan dengan driver dan jalankan classpath.
<code>--paket</code>	Daftar koordinat maven stoples yang dipisahkan koma untuk disertakan pada classpath driver dan eksekutor.
<code>--exclude-paket</code>	Daftar <code>groupId:ArtifactID</code> yang dipisahkan koma, untuk dikecualikan saat menyelesaikan dependensi yang disediakan di <code>—packages</code> untuk menghindari konflik ketergantungan.
<code>--repositori</code>	Daftar repositori jarak jauh tambahan yang dipisahkan koma untuk mencari koordinat maven yang diberikan dengan <code>—packages</code> .

Opsi	Deskripsi
--file BERKAS	Daftar file yang dipisahkan koma untuk ditempatkan di direktori kerja masing-masing pelaksana.
--conf PROP=NILAI	Properti konfigurasi percikan.
--properties-file BERKAS	Jalur ke file untuk memuat properti tambahan.
--driver-memori MEM	Memori untuk pengemudi. Standar 1024MB.
--driver-java-options	Opsi Java ekstra untuk diteruskan ke driver.
--driver-library-path	Entri jalur perpustakaan tambahan untuk diteruskan ke pengemudi.
--driver-class-path	Entri classpath ekstra untuk diteruskan ke pengemudi.
--executor-memori MEM	Memori per eksekutor. Standar 1GB.
--driver-core NUM	Jumlah core yang digunakan oleh driver.
-- total-executor-cores JUMLAH	Total core untuk semua pelaksana.
--executor-core NUM	Jumlah core yang digunakan oleh masing-masing eksekutor.
--num-pelaksana NUM	Jumlah pelaksana yang akan diluncurkan.
-hivevar <kunci=nilai>	Substitusi variabel untuk diterapkan pada perintah Hive, misalnya, -hivevar A=B
-hiveconf <property=nilai>	Nilai yang akan digunakan untuk properti yang diberikan.

Untuk pekerjaan Spark SQL, buat start-job-run-request file.json dan tentukan parameter yang diperlukan untuk menjalankan pekerjaan Anda, seperti pada contoh berikut:

```

{
  "name": "myjob",
  "virtualClusterId": "123456",
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "emr-6.7.0-latest",
  "jobDriver": {
    "sparkSqlJobDriver": {
      "entryPoint": "entryPoint_location",
      "sparkSqlParameters": "--conf spark.executor.instances=2 --conf
spark.executor.memory=2G --conf spark.executor.cores=2 --conf spark.driver.cores=1"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.driver.memory": "2G"
        }
      }
    ],
    "monitoringConfiguration": {
      "persistentAppUI": "ENABLED",
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "my_log_group",
        "logStreamNamePrefix": "log_stream_prefix"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://my_s3_log_location"
      }
    }
  }
}

```

Status tugas berjalan

Ketika Anda mengirimkan tugas berjalan ke antrean tugas Amazon EMR di EKS, tugas berjalan memasuki status PENDING. Kemudian melewati status-status berikut sampai berhasil (keluar dengan kode 0) atau gagal (keluar dengan kode bukan nol).

Tugas berjalan dapat memiliki status berikut:

- **PENDING** - Status tugas awal saat tugas berjalan dikirimkan ke Amazon EMR di EKS. Tugas sedang menunggu untuk dikirimkan ke klaster virtual, dan Amazon EMR di EKS sedang bekerja untuk mengirimkan tugas ini.
- **SUBMITTED** - Tugas berjalan yang telah berhasil dikirimkan ke klaster virtual. Penjadwal klaster kemudian mencoba untuk menjalankan tugas ini di cluster.
- **RUNNING** - Tugas berjalan yang berjalan di klaster virtual. Dalam aplikasi Spark, ini berarti bahwa proses driver Spark ada di status `running`.
- **FAILED** - Tugas berjalan yang gagal untuk dikirimkan ke klaster virtual atau yang gagal diselesaikan. Lihatlah `StateDetails` dan `FailureReason` untuk menemukan informasi tambahan tentang kegagalan pekerjaan ini.
- **COMPLETED** - Tugas berjalan yang telah berhasil diselesaikan.
- **CANCEL_PENDING** - Tugas berjalan telah diminta untuk pembatalan. Amazon EMR di EKS sedang mencoba untuk membatalkan tugas pada klaster virtual.
- **CANCELLED** - Tugas berjalan yang berhasil dibatalkan.

Melihat tugas di konsol Amazon EMR

Data Job run tersedia untuk dilihat, sehingga Anda dapat memantau setiap pekerjaan saat melewati status. Untuk melihat pekerjaan di konsol EMR Amazon, lakukan langkah-langkah berikut.

1. Di menu kiri konsol EMR Amazon, di bawah Amazon EMR di EKS, pilih Cluster virtual.
2. Dari daftar cluster virtual, pilih cluster virtual yang ingin Anda lihat pekerjaannya.
3. Pada tabel Tugas berjalan, pilih Lihat log untuk melihat detail tugas.

Note

Dukungan untuk pengalaman satu klik diaktifkan secara default. Ini dapat dimatikan dengan mengatur `persistentAppUI` ke `DISABLED` dalam `monitoringConfiguration` selama pengiriman tugas. Untuk informasi selengkapnya, lihat [Melihat Antarmuka Pengguna Aplikasi Persisten](#).

Kesalahan umum saat menjalankan tugas

Kesalahan berikut dapat terjadi ketika Anda menjalankan API `StartJobRun`. Tabel mencantumkan setiap kesalahan dan menyediakan langkah-langkah mitigasi sehingga Anda dapat mengatasi masalah dengan cepat.

Pesan Kesalahan	Kondisi Kesalahan	Langkah Selanjutnya yang Disarankan
<code>error: argumen -- <i>argument</i> diperlukan</code>	Parameter yang diperlukan hilang.	Tambahkan argumen yang hilang ke permintaan API.
Terjadi kesalahan (<code>AccessDeniedException</code>) saat memanggil <code>StartJobRun</code> operasi: Pengguna: <i>ARN</i> tidak berwenang untuk melakukan: <code>emr-container: StartJobRun</code>	Peran eksekusi hilang.	Lihat Menggunakan peran eksekusi tugas dengan Amazon EMR di EKS .
Terjadi kesalahan (<code>AccessDeniedException</code>) saat memanggil <code>StartJobRun</code> operasi: Pengguna: <i>ARN</i> tidak berwenang untuk melakukan: <code>emr-container: StartJobRun</code>	Pemanggil tidak memiliki izin untuk peran eksekusi [format valid / tidak valid] melalui kunci kondisi.	Lihat Menggunakan peran eksekusi tugas dengan Amazon EMR di EKS .
Terjadi kesalahan (<code>AccessDeniedException</code>) saat memanggil <code>StartJobRun</code> operasi: Pengguna: <i>ARN</i> tidak berwenang untuk melakukan: <code>emr-container: StartJobRun</code>	Pengirim tugas dan Peran eksekusi ARN berasal dari akun yang berbeda.	Pastikan bahwa pengirim tugas dan peran eksekusi ARN adalah dari akun AWS yang sama.
1 kesalahan validasi terdeteksi: Nilai <i>Role</i> pada 'executionRoleArn' gagal memenuhi pola ekspresi reguler ARN: <code>^arn : (aws [A-Za-z0-9-]</code>	Pemanggil memiliki izin untuk peran eksekusi melalui kunci kondisi, tetapi peran tidak memenuhi batasan format ARN.	Berikan peran eksekusi mengikuti format ARN berikut. Lihat Menggunakan peran eksekusi tugas dengan Amazon EMR di EKS .

Pesan Kesalahan	Kondisi Kesalahan	Langkah Selanjutnya yang Disarankan
*) :iam: :(\ d {12})? : (peran ((\ u002f) (\ u002f [\ u0021-\ u007f] +\ u002f)) [\ w+=, .@-] +)		
Terjadi kesalahan (Resource NotFoundException) saat memanggil StartJobRun operasi: Kluster virtual <i>Virtual Cluster ID</i> tidak ada.	ID kluster virtual tidak ditemukan.	Menyediakan kluster virtual ID terdaftar dengan Amazon EMR di EKS.
Terjadi kesalahan (ValidatonException) saat memanggil StartJobRun operasi: Status kluster virtual <i>state</i> tidak valid untuk membuat sumber daya JobRun.	Kluster virtual tidak siap untuk melaksanakan tugas.	Lihat Status kluster virtual .
Terjadi kesalahan (Resource NotFoundException) saat memanggil StartJobRun operasi: Rilis <i>RELEASE</i> tidak ada.	Rilis yang ditentukan dalam pengiriman tugas tidak benar.	Lihat Amazon EMR pada rilis EKS .

Pesan Kesalahan	Kondisi Kesalahan	Langkah Selanjutnya yang Disarankan
<p>Terjadi kesalahan (AccessDeniedException) saat memanggil StartJobRun operasi: Pengguna: tidak ARN diizinkan untuk melakukan : emr-container: StartJobRun on resource: ARN dengan penolakan eksplisit.</p> <p>Terjadi kesalahan (AccessDeniedException) saat memanggil StartJobRun operasi: Pengguna: ARN tidak berwenang untuk melakukan: emr-container: StartJobRun on resource: ARN</p>	<p>Pengguna tidak berwenang untuk menelepon StartJobRun.</p>	<p>Lihat Menggunakan peran eksekusi tugas dengan Amazon EMR di EKS.</p>
<p>Terjadi galat (ValidationException) saat memanggil StartJobRun operasi: ConfigurationOverrides.MonitoringConfiguration.s3 .loguri gagal memenuhi batasanMonitoringConfiguration: %s</p>	<p>Jalur S3 sintaks URI tidak valid.</p>	<p>logURI harus dalam format s3://...</p>

Kesalahan berikut dapat terjadi ketika Anda menjalankan API DescribeJobRun sebelum tugas berjalan.

Pesan Kesalahan	Kondisi Kesalahan	Langkah Selanjutnya yang Disarankan
<p>StateDetails: JobRun pengiriman gagal.</p>	<p>Parameter StartJobRun dalam tidak valid.</p>	<p>Lihat Amazon EMR pada rilis EKS.</p>

Pesan Kesalahan	Kondisi Kesalahan	Langkah Selanjutnya yang Disarankan
<p>Klasifikasi <i>classification</i> tidak didukung.</p> <p>failureReason: VALIDATION_ERROR</p> <p>status: GAGAL.</p>		
<p>StateDetails: Cluster <i>EKS Cluster ID</i> tidak ada.</p> <p>failureReason: CLUSTER_UNAVAILABLE</p> <p>status: GAGAL</p>	Klaster EKS tidak tersedia.	Periksa apakah klaster EKS ada dan memiliki izin yang tepat. Untuk informasi selengkapnya, lihat Menyiapkan Amazon EMR di EKS .
<p>StateDetails: Cluster <i>EKS Cluster ID</i> tidak memiliki izin yang memadai.</p> <p>failureReason: CLUSTER_UNAVAILABLE</p> <p>status: GAGAL</p>	Amazon EMR tidak memiliki izin untuk mengakses klaster EKS.	Verifikasi bahwa izin diatur untuk Amazon EMR pada namespace terdaftar. Untuk informasi selengkapnya, lihat Menyiapkan Amazon EMR di EKS .
<p>StateDetails: Cluster saat ini <i>EKS Cluster ID</i> tidak dapat dijangkau.</p> <p>failureReason: CLUSTER_UNAVAILABLE</p> <p>status: GAGAL</p>	Klaster EKS tidak dapat dijangkau.	Periksa apakah Klaster EKS ada dan memiliki izin yang tepat. Untuk informasi selengkapnya, lihat Menyiapkan Amazon EMR di EKS .

Pesan Kesalahan	Kondisi Kesalahan	Langkah Selanjutnya yang Disarankan
<p>StateDetails: JobRun pengiriman gagal karena kesalahan internal.</p> <p>failureReason: INTERNAL_ERROR</p> <p>status: GAGAL</p>	<p>Kesalahan internal telah terjadi dengan klaster EKS.</p>	<p>N/A</p>
<p>StateDetails: Cluster <i>EKS Cluster ID</i> tidak memiliki sumber daya yang cukup.</p> <p>failureReason: USER_ERROR</p> <p>status: GAGAL</p>	<p>Ada sumber daya yang tidak mencukupi di klaster EKS untuk menjalankan tugas.</p>	<p>Tambahkan lebih banyak kapasitas ke grup simpul EKS atau atur EKS Autoscaler. Untuk informasi lebih lanjut, lihat Klaster Autoscaler.</p>

Kesalahan berikut dapat terjadi ketika Anda menjalankan API DescribeJobRun setelah tugas berjalan.

Pesan Kesalahan	Kondisi Kesalahan	Langkah Selanjutnya yang Disarankan
<p>StateDetails: Kesulitan memantau Anda. JobRun</p> <p>Cluster <i>EKS Cluster ID</i> tidak ada.</p> <p>failureReason: CLUSTER_UNAVAILABLE</p> <p>status: GAGAL</p>	<p>Klaster EKS tidak ada.</p>	<p>Periksa apakah Klaster EKS ada dan memiliki izin yang tepat. Untuk informasi selengkapnya, lihat Menyiapkan Amazon EMR di EKS.</p>

Pesan Kesalahan	Kondisi Kesalahan	Langkah Selanjutnya yang Disarankan
<p>StateDetails: Kesulitan memantau Anda. JobRun</p> <p>Cluster <i>EKS Cluster ID</i> tidak memiliki izin yang memadai.</p> <p>failureReason: CLUSTER_UNAVAILABLE</p> <p>status: GAGAL</p>	<p>Amazon EMR tidak memiliki izin untuk mengakses kluster EKS.</p>	<p>Verifikasi bahwa izin diatur untuk Amazon EMR pada namespace terdaftar. Untuk informasi selengkapnya, lihat Menyiapkan Amazon EMR di EKS.</p>
<p>StateDetails: Kesulitan memantau Anda. JobRun</p> <p>Cluster saat <i>EKS Cluster ID</i> ini tidak dapat dijangkau.</p> <p>failureReason: CLUSTER_UNAVAILABLE</p> <p>status: GAGAL</p>	<p>Kluster EKS tidak dapat dijangkau.</p>	<p>Periksa apakah Kluster EKS ada dan memiliki izin yang tepat. Untuk informasi selengkapnya, lihat Menyiapkan Amazon EMR di EKS.</p>
<p>StateDetails: Kesulitan memantau Anda JobRun karena kesalahan internal</p> <p>failureReason: INTERNAL_ERROR</p> <p>status: GAGAL</p>	<p>Kesalahan internal telah terjadi dan mencegah JobRun pemantauan.</p>	<p>N/A</p>

Kesalahan berikut dapat terjadi ketika pekerjaan tidak dapat dimulai dan pekerjaan menunggu dalam keadaan DIKIRIM selama 15 menit. Hal ini dapat disebabkan oleh kurangnya sumber daya cluster.

Pesan Kesalahan	Kondisi Kesalahan	Langkah Selanjutnya yang Disarankan
batas waktu cluster	Pekerjaan telah dalam keadaan DIKIRIM selama 15 menit atau lebih.	Anda dapat mengganti pengaturan default 15 menit untuk parameter ini dengan penggantian konfigurasi yang ditunjukkan di bawah ini.

Gunakan konfigurasi berikut untuk mengubah pengaturan batas waktu cluster menjadi 30 menit. Perhatikan bahwa Anda memberikan `job-start-timeout` nilai baru dalam hitungan detik:

```
{
  "configurationOverrides": {
    "applicationConfiguration": [{
      "classification": "emr-containers-defaults",
      "properties": {
        "job-start-timeout": "1800"
      }
    }]
  }
}
```

Menggunakan template pekerjaan

Template pekerjaan menyimpan nilai yang dapat dibagikan di seluruh pemanggilan `StartJobRun` API saat memulai menjalankan pekerjaan. Ini mendukung dua kasus penggunaan:

- Untuk mencegah nilai permintaan `StartJobRun` API berulang berulang.
- Untuk menegakkan aturan bahwa nilai tertentu harus disediakan melalui permintaan `StartJobRun` API.

Job templates memungkinkan Anda untuk menentukan template yang dapat digunakan kembali untuk menjalankan pekerjaan untuk menerapkan kustomisasi tambahan, misalnya:

- Mengkonfigurasi kapasitas komputasi eksekutor dan driver
- Menetapkan properti keamanan dan tata kelola seperti peran IAM

- Menyesuaikan image docker untuk digunakan di beberapa aplikasi dan pipeline data

Topik berikut memberikan informasi terperinci tentang penggunaan templat, termasuk cara menggunakannya untuk memulai pekerjaan dan cara mengubah parameter templat.

Topik

- [Membuat dan menggunakan template pekerjaan untuk memulai pekerjaan](#)
- [Mendefinisikan parameter template pekerjaan](#)
- [Mengontrol akses ke templat pekerjaan](#)

Membuat dan menggunakan template pekerjaan untuk memulai pekerjaan

Bagian ini menjelaskan pembuatan template pekerjaan dan menggunakan template untuk memulai pekerjaan dengan AWS Command Line Interface (AWS CLI).

Untuk membuat template pekerjaan

1. Buat `create-job-template-request.json` file dan tentukan parameter yang diperlukan untuk template pekerjaan Anda, seperti yang ditunjukkan dalam contoh file JSON berikut. Untuk informasi tentang semua parameter yang tersedia, lihat [CreateJobTemplateAPI](#).

Sebagian besar nilai yang diperlukan untuk `StartJobRun` API juga diperlukan untuk `jobTemplateData`. Jika Anda ingin menggunakan placeholder untuk parameter apa pun dan memberikan nilai saat memanggil `StartJobRun` menggunakan templat pekerjaan, silakan lihat bagian selanjutnya pada parameter templat pekerjaan.

```
{
  "name": "mytemplate",
  "jobTemplateData": {
    "executionRoleArn": "iam_role_arn_for_job_execution",
    "releaseLabel": "emr-6.7.0-latest",
    "jobDriver": {
      "sparkSubmitJobDriver": {
        "entryPoint": "entryPoint_location",
        "entryPointArguments": [ "argument1", "argument2", ... ],
        "sparkSubmitParameters": "--class <main_class> --conf
spark.executor.instances=2 --conf spark.executor.memory=2G --conf
spark.executor.cores=2 --conf spark.driver.cores=1"
      }
    }
  }
}
```

```
    },
    "configurationOverrides": {
      "applicationConfiguration": [
        {
          "classification": "spark-defaults",
          "properties": {
            "spark.driver.memory": "2G"
          }
        }
      ],
      "monitoringConfiguration": {
        "persistentAppUI": "ENABLED",
        "cloudWatchMonitoringConfiguration": {
          "logGroupName": "my_log_group",
          "logStreamNamePrefix": "log_stream_prefix"
        },
        "s3MonitoringConfiguration": {
          "logUri": "s3://my_s3_log_location/"
        }
      }
    }
  }
}
```

2. Gunakan `create-job-template` perintah dengan path ke `create-job-template-request.json` file yang disimpan secara lokal.

```
aws emr-containers create-job-template \  
--cli-input-json file://./create-job-template-request.json
```

Untuk memulai pekerjaan dengan menggunakan template pekerjaan

Berikan id cluster virtual, id template pekerjaan, dan nama pekerjaan dalam `StartJobRun` perintah, seperti yang ditunjukkan pada contoh berikut.

```
aws emr-containers start-job-run \  
--virtual-cluster-id 123456 \  
--name myjob \  
--job-template-id 1234abcd
```

Mendefinisikan parameter template pekerjaan

Parameter template Job memungkinkan Anda untuk menentukan variabel dalam template pekerjaan. Nilai untuk variabel parameter ini perlu ditentukan saat memulai pekerjaan yang dijalankan menggunakan template pekerjaan itu. Parameter template Job ditentukan dalam `${parameterName}` format. Anda dapat memilih untuk menentukan nilai apa pun di `jobTemplateData` bidang sebagai parameter template pekerjaan. Untuk setiap variabel parameter template pekerjaan, tentukan tipe datanya (STRING atau NUMBER) dan secara opsional nilai default. Contoh di bawah ini menunjukkan bagaimana Anda dapat menentukan parameter template pekerjaan untuk lokasi titik masuk, kelas utama, dan nilai lokasi log S3.

Untuk menentukan lokasi titik masuk, kelas utama, dan lokasi log Amazon S3 sebagai parameter templat pekerjaan

1. Buat `create-job-template-request.json` file dan tentukan parameter yang diperlukan untuk template pekerjaan Anda, seperti yang ditunjukkan dalam contoh file JSON berikut. Untuk informasi selengkapnya tentang parameter, lihat [CreateJobTemplateAPI](#).

```
{
  "name": "mytemplate",
  "jobTemplateData": {
    "executionRoleArn": "iam_role_arn_for_job_execution",
    "releaseLabel": "emr-6.7.0-latest",
    "jobDriver": {
      "sparkSubmitJobDriver": {
        "entryPoint": "${EntryPointLocation}",
        "entryPointArguments": [ "argument1", "argument2", ... ],
        "sparkSubmitParameters": "--class ${MainClass} --conf
spark.executor.instances=2 --conf spark.executor.memory=2G --conf
spark.executor.cores=2 --conf spark.driver.cores=1"
      }
    },
    "configurationOverrides": {
      "applicationConfiguration": [
        {
          "classification": "spark-defaults",
          "properties": {
            "spark.driver.memory": "2G"
          }
        }
      ]
    }
  },
}
```

```

    "monitoringConfiguration": {
      "persistentAppUI": "ENABLED",
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "my_log_group",
        "logStreamNamePrefix": "log_stream_prefix"
      },
      "s3MonitoringConfiguration": {
        "logUri": "${LogS3BucketUri}"
      }
    },
    "parameterConfiguration": {
      "EntryPointLocation": {
        "type": "STRING"
      },
      "MainClass": {
        "type": "STRING",
        "defaultValue": "Main"
      },
      "LogS3BucketUri": {
        "type": "STRING",
        "defaultValue": "s3://my_s3_log_location/"
      }
    }
  }
}

```

- Gunakan perintah `create-job-template` dengan jalur ke file `create-job-template-request.json` yang disimpan secara lokal atau di Amazon S3.

```

aws emr-containers create-job-template \
--cli-input-json file://./create-job-template-request.json

```

Untuk memulai pekerjaan dengan menggunakan template pekerjaan dengan parameter template pekerjaan

Untuk memulai pekerjaan dengan template pekerjaan yang berisi parameter template pekerjaan, tentukan id templat pekerjaan serta nilai untuk parameter template pekerjaan dalam permintaan `StartJobRun` API seperti yang ditunjukkan di bawah ini.

```

aws emr-containers start-job-run \
--virtual-cluster-id 123456 \

```

```
--name myjob \  
--job-template-id 1234abcd \  
--job-template-parameters '{"EntryPointLocation": "entry_point_location", "MainClass": "ExampleMainClass", "LogS3BucketUri": "s3://example_s3_bucket/"}'
```

Mengontrol akses ke templat pekerjaan

StartJobRunpolicy memungkinkan Anda menegakkan bahwa pengguna atau peran hanya dapat menjalankan pekerjaan menggunakan templat pekerjaan yang Anda tentukan dan tidak dapat menjalankan StartJobRun operasi tanpa menggunakan templat pekerjaan yang ditentukan. Untuk mencapai hal ini, pertama-tama pastikan bahwa Anda memberi pengguna atau peran izin baca ke templat pekerjaan yang ditentukan seperti yang ditunjukkan di bawah ini.

JSON

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "emr-containers:DescribeJobRun"  
      ],  
      "Resource": [  
        "arn:aws:emr-containers:*:*:jobtemplate/job_template_1_id",  
        "arn:aws:emr-containers:*:*:jobtemplate/job_template_2_id"  
      ],  
      "Sid": "AllowEMRCONTAINERSDescribejobtemplate"  
    }  
  ]  
}
```

Untuk menegakkan bahwa pengguna atau peran hanya dapat menjalankan StartJobRun operasi saat menggunakan templat pekerjaan tertentu, Anda dapat menetapkan izin StartJobRun kebijakan berikut ke pengguna atau peran tertentu.

JSON

```
{
```

```

"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "emr-containers:StartJobRun"
    ],
    "Resource": [
      "arn:aws:emr-containers:*:*:/virtualclusters/virtual_cluster_id"
    ],
    "Condition": {
      "ArnLike": {
        "emr-containers:JobTemplateArn": [
          "arn:aws:emr-containers:*:*:jobtemplate/job_template_1_id",
          "arn:aws:emr-containers:*:*:jobtemplate/job_template_2_id"
        ]
      }
    },
    "Sid": "AllowEMRCONTAINERSStartjobrun"
  }
]
}

```

Jika template pekerjaan menentukan parameter template pekerjaan di dalam bidang ARN peran eksekusi, maka pengguna akan dapat memberikan nilai untuk parameter ini dan dengan demikian dapat StartJobRun memanggil menggunakan peran eksekusi arbitrer. Untuk membatasi peran eksekusi yang dapat diberikan pengguna, lihat Mengontrol akses ke peran eksekusi. [Menggunakan peran eksekusi tugas dengan Amazon EMR di EKS](#)

Jika tidak ada kondisi yang ditentukan dalam kebijakan StartJobRun tindakan di atas untuk pengguna atau peran tertentu, pengguna atau peran tersebut akan diizinkan untuk memanggil StartJobRun tindakan pada kluster virtual yang ditentukan menggunakan templat pekerjaan arbitrer yang telah mereka baca akses atau menggunakan peran eksekusi arbitrer.

Menggunakan templat pod

Dimulai dengan Amazon EMR versi 5.33.0 atau 6.3.0, Amazon EMR di EKS mendukung fitur templat pod Spark. Sebuah pod adalah sekelompok satu atau lebih kontainer, dengan penyimpanan bersama dan sumber daya jaringan, dan spesifikasi cara menjalankan kontainer. Templat pod adalah spesifikasi yang menentukan cara menjalankan setiap pod. Anda dapat menggunakan file templat

pod untuk menentukan konfigurasi driver atau pelaksana pod yang tidak didukung konfigurasi Spark. Untuk informasi selengkapnya tentang fitur templat pod Spark, lihat [Templat Pod](#).

Note

Fitur templat pod hanya bekerja dengan driver dan pelaksana pod. Anda tidak dapat mengonfigurasi pod pengirim pekerjaan menggunakan templat pod.

Skenario umum

Anda dapat menentukan cara menjalankan tugas Spark di kluster EKS yang dibagikan dengan menggunakan templat pod dengan Amazon EMR di EKS dan menghemat biaya dan meningkatkan pemanfaatan sumber daya dan performa.

- Untuk mengurangi biaya, Anda dapat menjadwalkan tugas driver Spark untuk dijalankan di Instans Sesuai Permintaan Amazon EC2 sambil menjadwalkan tugas pelaksana Spark untuk dijalankan di Instans Spot Amazon EC2.
- Untuk meningkatkan pemanfaatan sumber daya, Anda dapat mendukung beberapa tim yang menjalankan beban kerja mereka pada kluster EKS yang sama. Setiap tim akan mendapatkan grup simpul Amazon EC2 yang ditunjuk untuk tempat menjalankan beban kerja mereka. Anda dapat menggunakan templat pod untuk menerapkan toleransi yang sesuai untuk beban kerja mereka.
- Untuk meningkatkan pemantauan, Anda dapat menjalankan kontainer pencatatan terpisah untuk meneruskan log pada aplikasi pemantauan yang ada.

Sebagai contoh, file templat pod berikut menunjukkan skenario penggunaan umum.

```
apiVersion: v1
kind: Pod
spec:
  volumes:
    - name: source-data-volume
      emptyDir: {}
    - name: metrics-files-volume
      emptyDir: {}
  nodeSelector:
    eks.amazonaws.com/nodegroup: emr-containers-nodegroup
  containers:
```

```

- name: spark-kubernetes-driver # This will be interpreted as driver Spark main
container
  env:
    - name: RANDOM
      value: "random"
  volumeMounts:
    - name: shared-volume
      mountPath: /var/data
    - name: metrics-files-volume
      mountPath: /var/metrics/data
- name: custom-side-car-container # Sidecar container
  image: <side_car_container_image>
  env:
    - name: RANDOM_SIDE CAR
      value: random
  volumeMounts:
    - name: metrics-files-volume
      mountPath: /var/metrics/data
  command:
    - /bin/sh
    - '-c'
    - <command-to-upload-metrics-files>
initContainers:
- name: spark-init-container-driver # Init container
  image: <spark-pre-step-image>
  volumeMounts:
    - name: source-data-volume # Use EMR predefined volumes
      mountPath: /var/data
  command:
    - /bin/sh
    - '-c'
    - <command-to-download-dependency-jars>

```

Templat pod menyelesaikan tugas berikut:

- Tambahkan [kontainer init](#) baru yang dieksekusi sebelum kontainer utama Spark dimulai. Wadah init berbagi [EmptyDir volume](#) yang disebut `source-data-volume` dengan wadah utama Spark. Anda dapat meminta kontainer init Anda menjalankan langkah inisialisasi, seperti mengunduh dependensi atau menghasilkan data input. Kemudian kontainer utama Spark mengkonsumsi data.
- Tambahkan [kontainer sidecar](#) lain yang dieksekusi bersama dengan kontainer utama Spark. Kedua kontainer berbagi volume EmptyDir lain yang disebut `metrics-files-volume`. Tugas

Spark Anda dapat menghasilkan metrik, seperti metrik Prometheus. Kemudian tugas Spark dapat menempatkan metrik ke dalam file dan meminta kontainer sidecar mengunggah file ke sistem BI Anda sendiri untuk analisis di masa mendatang.

- Tambahkan variabel lingkungan baru ke kontainer utama Spark. Anda dapat meminta tugas Anda mengkonsumsi variabel lingkungan.
- Definisikan sebuah [simpul pemilih](#), sehingga pod hanya dijadwalkan pada grup simpul `emr-containers-nodegroup`. Hal ini membantu untuk mengisolasi sumber daya komputasi di seluruh tugas dan tim.

Mengaktifkan templat pod dengan Amazon EMR di EKS

Untuk mengaktifkan fitur templat pod dengan Amazon EMR pada EKS, konfigurasi properti Spark `spark.kubernetes.driver.podTemplateFile` dan `spark.kubernetes.executor.podTemplateFile` untuk menunjuk ke file templat pod di Amazon S3. Spark kemudian mengunduh file templat pod dan menggunakannya untuk membangun driver dan pod pelaksana.

Note

Spark menggunakan peran eksekusi tugas untuk memuat templat pod, sehingga peran eksekusi pekerjaan harus memiliki izin untuk mengakses Amazon S3 untuk memuat templat pod. Untuk informasi selengkapnya, lihat [Untuk membuat peran eksekusi tugas](#).

Anda dapat menggunakan `SparkSubmitParameters` untuk menentukan jalur Amazon S3 ke templat pod, seperti yang ditunjukkan file tugas berjalan JSON berikut.

```
{
  "name": "myjob",
  "virtualClusterId": "123456",
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "release_label",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "entryPoint_location",
      "entryPointArguments": [argument1, "argument2", ...],
      "sparkSubmitParameters": "--class <main_class> \
        --conf
spark.kubernetes.driver.podTemplateFile=s3://path_to_driver_pod_template \
```

```

    --conf
    spark.kubernetes.executor.podTemplateFile=s3://path_to_executor_pod_template \
    --conf spark.executor.instances=2 \
    --conf spark.executor.memory=2G \
    --conf spark.executor.cores=2 \
    --conf spark.driver.cores=1"
  }
}
}

```

Atau, Anda dapat menggunakan `configurationOverrides` untuk menentukan jalur Amazon S3 ke templat pod, seperti yang ditunjukkan file tugas berjalan JSON berikut.

```

{
  "name": "myjob",
  "virtualClusterId": "123456",
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "release_label",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "entryPoint_location",
      "entryPointArguments": ["argument1", "argument2", ...],
      "sparkSubmitParameters": "--class <main_class> \
        --conf spark.executor.instances=2 \
        --conf spark.executor.memory=2G \
        --conf spark.executor.cores=2 \
        --conf spark.driver.cores=1"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.driver.memory": "2G",
          "spark.kubernetes.driver.podTemplateFile": "s3://path_to_driver_pod_template",
          "spark.kubernetes.executor.podTemplateFile": "s3://path_to_executor_pod_template"
        }
      }
    ]
  }
}

```

Note

1. Anda harus mengikuti pedoman keamanan saat menggunakan fitur templat pod dengan Amazon EMR di EKS, seperti mengisolasi kode aplikasi yang tidak dipercaya. Untuk informasi selengkapnya, lihat [Amazon EMR pada praktik terbaik keamanan EKS](#).
2. Anda tidak dapat mengubah nama kontainer utama Spark dengan menggunakan `spark.kubernetes.driver.podTemplateContainerName` dan `spark.kubernetes.executor.podTemplateContainerName`, karena nama-nama ini di-hardcode sebagai `spark-kubernetes-driver` dan `spark-kubernetes-executors`. Jika Anda ingin menyesuaikan kontainer utama Spark, Anda harus menentukan kontainer dalam templat pod dengan nama-nama di-hardcode ini.

Bidang templat pod

Pertimbangkan pembatasan bidang berikut ketika mengonfigurasi templat pod dengan Amazon EMR di EKS.

- Amazon EMR di EKS hanya mengizinkan bidang-bidang berikut dalam templat pod untuk mengaktifkan penjadwalan tugas yang tepat.

Ini adalah bidang tingkat pod yang diizinkan:

- `apiVersion`
- `kind`
- `metadata`
- `spec.activeDeadlineSeconds`
- `spec.affinity`
- `spec.containers`
- `spec.enableServiceLinks`
- `spec.ephemeralContainers`
- `spec.hostAliases`
- `spec.hostname`
- `spec.imagePullSecrets`

- `spec.nodeName`
- `spec.nodeSelector`
- `spec.overhead`
- `spec.preemptionPolicy`
- `spec.priority`
- `spec.priorityClassName`
- `spec.readinessGates`
- `spec.runtimeClassName`
- `spec.schedulerName`
- `spec.subdomain`
- `spec.terminationGracePeriodSeconds`
- `spec.tolerations`
- `spec.topologySpreadConstraints`
- `spec.volumes`

Ini adalah bidang tingkat kontainer utama Spark yang diizinkan:

- `env`
- `envFrom`
- `name`
- `lifecycle`
- `livenessProbe`
- `readinessProbe`
- `resources`
- `startupProbe`
- `stdin`
- `stdinOnce`
- `terminationMessagePath`
- `terminationMessagePolicy`
- `tty`

- `workingDir`

Saat Anda menggunakan bidang yang tidak diizinkan dalam templat pod, Spark melempar pengecualian dan tugas gagal. Contoh berikut menunjukkan pesan kesalahan dalam log pengendali Spark karena bidang tidak diizinkan.

```
Executor pod template validation failed.  
Field container.command in Spark main container not allowed but specified.
```

- Amazon EMR di EKS menentukan terlebih dahulu parameter berikut dalam templat pod. Bidang yang Anda tentukan dalam templat pod tidak boleh tumpang tindih dengan bidang ini.

Ini adalah nama volume yang telah ditetapkan:

- `emr-container-communicate`
- `config-volume`
- `emr-container-application-log-dir`
- `emr-container-event-log-dir`
- `temp-data-dir`
- `mnt-dir`
- `home-dir`
- `emr-container-s3`

Ini adalah pemasangan volume yang telah ditetapkan yang hanya berlaku untuk kontainer utama Spark:

- Nama:`emr-container-communicate`; MountPath: `/var/log/fluentd`
- Nama:`emr-container-application-log-dir`; MountPath: `/var/log/spark/user`
- Nama:`emr-container-event-log-dir`; MountPath: `/var/log/spark/apps`
- Nama:`mnt-dir`; MountPath: `/mnt`
- Nama:`temp-data-dir`; MountPath: `/tmp`
- Nama:`home-dir`; MountPath: `/home/hadoop`

Ini adalah variabel lingkungan yang telah ditetapkan yang hanya berlaku untuk kontainer utama Spark:

- `SPARK_CONTAINER_ID`

- K8S_SPARK_LOG_URL_STDOUT
- SIDECAR_SIGNAL_FILE

Note

Anda masih dapat menggunakan volume standar yang telah ditentukan ini dan memasangnya ke kontainer sidecar tambahan Anda. Misalnya, Anda dapat menggunakan `emr-container-application-log-dir` dan memasangnya ke kontainer sidecar Anda sendiri yang didefinisikan dalam templat pod.

Jika bidang yang Anda tentukan bertentangan dengan salah satu bidang yang telah ditetapkan dalam templat pod, Spark melempar pengecualian dan tugas gagal. Contoh berikut menunjukkan pesan kesalahan dalam log aplikasi Spark karena bertentangan dengan bidang yang telah ditentukan.

```
Defined volume mount path on main container must not overlap with reserved mount paths: [<reserved-paths>]
```

Pertimbangan kontainer sidecar

Amazon EMR mengendalikan siklus hidup pod yang disediakan oleh Amazon EMR di EKS. Kontainer sidecar harus mengikuti siklus hidup yang sama dengan kontainer utama Spark. Jika Anda menyuntikkan kontainer sidecar tambahan ke pod Anda, kami sarankan Anda mengintegrasikan dengan manajemen siklus hidup pod yang didefinisikan Amazon EMR sehingga kontainer sidecar dapat berhenti sendiri ketika kontainer utama Spark keluar.

Untuk mengurangi biaya, sebaiknya Anda menerapkan proses yang mencegah driver pod dengan kontainer sidecar terus berjalan setelah tugas Anda selesai. Driver Spark menghapus pod pelaksana saat pelaksana selesai. Namun, ketika program driver selesai, kontainer sidecar tambahan terus berjalan. Pod ditagihkan sampai Amazon EMR di EKS membersihkan pod driver, biasanya kurang dari satu menit setelah kontainer utama driver Spark selesai. Untuk mengurangi biaya, Anda dapat mengintegrasikan kontainer sidecar tambahan Anda dengan mekanisme manajemen siklus hidup yang didefinisikan Amazon EMR di EKS untuk driver dan pelaksana pod, seperti yang dijelaskan pada bagian berikut.

Kontainer utama Spark dalam driver dan pelaksana pod mengirimkan heartbeat ke `/var/log/fluentd/main-container-terminated` file setiap dua detik. Dengan menambahkan Amazon EMR yang telah ditetapkan `emr-container-communicate` pemasangan volume ke kontainer sidecar Anda, Anda dapat menentukan sub-proses kontainer sidecar Anda untuk secara berkala melacak waktu terakhir diubah untuk file ini. Sub-proses kemudian berhenti sendiri jika menemukan bahwa kontainer utama Spark menghentikan heartbeat untuk durasi yang lebih lama.

Contoh berikut menunjukkan sub-proses yang melacak file detak jantung dan berhenti sendiri. Ganti `your_volume_mount` dengan jalur tempat Anda memasang volume yang telah ditentukan. Script dipaketkan di dalam gambar yang digunakan oleh kontainer sidecar. Dalam file templat pod, Anda dapat menentukan kontainer sidecar dengan perintah `sub_process_script.sh` dan `main_command` berikut.

```
MOUNT_PATH="your_volume_mount"
FILE_TO_WATCH="$MOUNT_PATH/main-container-terminated"
INITIAL_HEARTBEAT_TIMEOUT_THRESHOLD=60
HEARTBEAT_TIMEOUT_THRESHOLD=15
SLEEP_DURATION=10

function terminate_main_process() {
    # Stop main process
}

# Waiting for the first heartbeat sent by Spark main container
echo "Waiting for file $FILE_TO_WATCH to appear..."
start_wait=$(date +%s)
while ! [[ -f "$FILE_TO_WATCH" ]]; do
    elapsed_wait=$(expr $(date +%s) - $start_wait)
    if [ "$elapsed_wait" -gt "$INITIAL_HEARTBEAT_TIMEOUT_THRESHOLD" ]; then
        echo "File $FILE_TO_WATCH not found after $INITIAL_HEARTBEAT_TIMEOUT_THRESHOLD
seconds; aborting"
        terminate_main_process
        exit 1
    fi
    sleep $SLEEP_DURATION;
done;
echo "Found file $FILE_TO_WATCH; watching for heartbeats..."

while [[ -f "$FILE_TO_WATCH" ]]; do
    LAST_HEARTBEAT=$(stat -c %Y $FILE_TO_WATCH)
    ELAPSED_TIME_SINCE_AFTER_HEARTBEAT=$(expr $(date +%s) - $LAST_HEARTBEAT)
```

```

    if [ "$ELAPSED_TIME_SINCE_AFTER_HEARTBEAT" -gt "$HEARTBEAT_TIMEOUT_THRESHOLD" ];
then
    echo "Last heartbeat to file $FILE_TO_WATCH was more than
$HEARTBEAT_TIMEOUT_THRESHOLD seconds ago at $LAST_HEARTBEAT; terminating"
    terminate_main_process
    exit 0
fi
sleep $SLEEP_DURATION;
done;
echo "Outside of loop, main-container-terminated file no longer exists"

# The file will be deleted once the fluentd container is terminated

echo "The file $FILE_TO_WATCH doesn't exist any more;"
terminate_main_process
exit 0

```

Menggunakan kebijakan coba lagi pekerjaan

Di Amazon EMR di EKS versi 6.9.0 dan yang lebih baru, Anda dapat menetapkan kebijakan coba lagi untuk menjalankan pekerjaan Anda. Kebijakan coba lagi menyebabkan pod driver pekerjaan dimulai ulang secara otomatis jika gagal atau dihapus. Ini membuat pekerjaan streaming Spark yang berjalan lama lebih tahan terhadap kegagalan.

Menetapkan kebijakan coba lagi untuk suatu pekerjaan

Untuk mengonfigurasi kebijakan coba lagi, Anda menyediakan `RetryPolicyConfiguration` bidang menggunakan [StartJobRunAPI](#). Contoh `retryPolicyConfiguration` ditampilkan di sini:

```

aws emr-containers start-job-run \
--virtual-cluster-id cluster_id \
--name sample-job-name \
--execution-role-arn execution-role-arn \
--release-label emr-6.9.0-latest \
--job-driver '{
  "sparkSubmitJobDriver": {
    "entryPoint": "local:///usr/lib/spark/examples/src/main/python/pi.py",
    "entryPointArguments": [ "2" ],
    "sparkSubmitParameters": "--conf spark.executor.instances=2 --conf
spark.executor.memory=2G --conf spark.executor.cores=2 --conf spark.driver.cores=1"
  }
}

```

```
}' \  
--retry-policy-configuration '{  
  "maxAttempts": 5  
}' \  
--configuration-overrides '{  
  "monitoringConfiguration": {  
    "cloudWatchMonitoringConfiguration": {  
      "logGroupName": "my_log_group_name",  
      "logStreamNamePrefix": "my_log_stream_prefix"  
    },  
    "s3MonitoringConfiguration": {  
      "logUri": "s3://amzn-s3-demo-logging-bucket"  
    }  
  }  
}'
```

Note

`retryPolicyConfiguration` hanya tersedia dari versi AWS CLI 1.27.68 dan seterusnya. Untuk memperbarui AWS CLI ke versi terbaru, lihat [Menginstal atau memperbarui versi terbaru AWS CLI](#)

Konfigurasi `maxAttempts` bidang dengan jumlah maksimum kali Anda ingin pod driver pekerjaan dimulai ulang jika gagal atau dihapus. [Interval eksekusi antara dua upaya coba lagi pengemudi pekerjaan adalah interval percobaan ulang eksponensial \(10 detik, 20 detik, 40 detik...\) yang dibatasi pada 6 menit, seperti yang dijelaskan dalam dokumentasi Kubernetes.](#)

Note

Setiap eksekusi driver pekerjaan tambahan akan ditagih sebagai pekerjaan lain, dan akan dikenakan [EMR Amazon pada](#) harga EKS.

Coba lagi nilai konfigurasi kebijakan

- Kebijakan coba ulang default untuk pekerjaan: `StartJobRun` menyertakan kebijakan coba ulang yang disetel ke 1 upaya maksimum secara default. Anda dapat mengonfigurasi kebijakan coba lagi sesuai keinginan.

Note

Jika `maxAttempts` dari set ke 1, itu berarti tidak ada percobaan ulang yang akan dilakukan untuk memunculkan pod driver pada kegagalan. `retryPolicyConfiguration`

- Menonaktifkan kebijakan coba lagi untuk pekerjaan: Untuk menonaktifkan kebijakan coba lagi, tetapkan nilai upaya maksimal ke 1. `retryPolicyConfiguration`

```
"retryPolicyConfiguration": {
  "maxAttempts": 1
}
```

- Setel `MaxAttempt` untuk pekerjaan dalam rentang yang valid: `StartJobRun` panggilan akan gagal jika `maxAttempts` nilainya berada di luar rentang yang valid. `maxAttempts` Rentang yang valid adalah dari 1 hingga 2.147.483.647 (32-bit integer), rentang yang didukung untuk pengaturan konfigurasi Kubernetes. `backOffLimit` Untuk informasi selengkapnya, lihat [Kebijakan kegagalan backoff Pod di dokumentasi](#) Kubernetes. Jika `maxAttempts` nilainya tidak valid, pesan galat berikut dikembalikan:

```
{
  "message": "Retry policy configuration's parameter value of maxAttempts is invalid"
}
```

Mengambil status kebijakan coba lagi untuk suatu pekerjaan

Anda dapat melihat status upaya coba lagi untuk pekerjaan dengan [ListJobRuns](#) dan [DescribeJobRun](#) APIs. Setelah Anda meminta pekerjaan dengan konfigurasi kebijakan coba ulang yang diaktifkan, `DescribeJobRun` respons `ListJobRun` dan akan berisi status kebijakan coba lagi di `RetryPolicyExecution` bidang. Selain itu, `DescribeJobRun` respon akan berisi `RetryPolicyConfiguration` yang dimasukkan dalam `StartJobRun` permintaan untuk pekerjaan itu.

Sampel tanggapan

ListJobRuns response

```
{
```

```

"jobRuns": [
  ...
  ...
  "retryPolicyExecution" : {
    "currentAttemptCount": 2
  }
  ...
  ...
]
}

```

DescribeJobRun response

```

{
  ...
  ...
  "retryPolicyConfiguration": {
    "maxAttempts": 5
  },
  "retryPolicyExecution" : {
    "currentAttemptCount": 2
  },
  ...
  ...
}

```

Kolom ini tidak akan terlihat saat kebijakan coba ulang dinonaktifkan dalam pekerjaan, seperti yang dijelaskan di [Coba lagi nilai konfigurasi kebijakan](#) bawah ini.

Memantau pekerjaan dengan kebijakan coba lagi

Saat Anda mengaktifkan kebijakan coba lagi, CloudWatch peristiwa akan dibuat untuk setiap driver pekerjaan yang dibuat. Untuk berlangganan acara ini, siapkan aturan CloudWatch acara menggunakan perintah berikut:

```

aws events put-rule \
--name cwe-test \
--event-pattern '{"detail-type": ["EMR Job Run New Driver Attempt"]}'

```

Acara ini akan mengembalikan informasi tentang `newDriverPodName`, `newDriverCreatedAt` stempel waktu `previousDriverFailureMessage`, dan `currentAttemptCount` driver pekerjaan. Peristiwa ini tidak akan dibuat jika kebijakan coba lagi dinonaktifkan.

Untuk informasi selengkapnya tentang cara memantau pekerjaan Anda dengan CloudWatch acara, lihat [Pantau pekerjaan dengan Amazon CloudWatch Events](#).

Menemukan log untuk driver dan pelaksana

Nama pod driver mengikuti formatnya `spark-<job id>-driver-<random-suffix>`. `random-suffix` Hal yang sama ditambahkan ke nama pod eksekutor yang muncul driver. Saat Anda menggunakan ini `random-suffix`, Anda dapat menemukan log untuk driver dan pelaksana terkait. Hanya `random-suffix` ada jika [kebijakan coba lagi diaktifkan](#) untuk pekerjaan; jika tidak, tidak `random-suffix` ada.

Untuk informasi selengkapnya tentang cara mengonfigurasi pekerjaan dengan konfigurasi pemantauan untuk pencatatan, lihat [Jalankan aplikasi Spark](#).

Menggunakan rotasi log peristiwa Spark

Dengan Amazon EMR 6.3.0 dan yang lebih baru, Anda dapat mengaktifkan fitur rotasi log peristiwa Spark untuk Amazon EMR di EKS. Alih-alih menghasilkan file log peristiwa tunggal, fitur ini merotasikan file berdasarkan interval waktu terkonfigurasi Anda dan menghapus file log peristiwa terlama.

Merotasi log peristiwa Spark dapat membantu Anda menghindari potensi masalah dengan file log peristiwa Spark besar yang dihasilkan untuk tugas yang dijalankan atau di-stream dalam jangka panjang. Misalnya, Anda memulai tugas Spark berjangka panjang dengan log peristiwa yang diaktifkan dengan parameter `persistantAppUI`. Driver Spark menghasilkan file log peristiwa. Jika tugas berjalan selama berjam-jam atau sehari-hari dan ada ruang disk terbatas pada simpul Kubernetes, file log peristiwa dapat mengkonsumsi semua ruang disk yang tersedia. Mengaktifkan fitur rotasi log peristiwa Spark memecahkan masalah dengan membelah file log ke beberapa file dan menghapus file terlama.

Note

Fitur ini hanya berfungsi dengan Amazon EMR di EKS. Amazon EMR yang berjalan di Amazon EC2 tidak mendukung rotasi log peristiwa Spark.

Untuk mengaktifkan fitur rotasi log peristiwa Spark, konfigurasi parameter Spark berikut:

- `spark.eventLog.rotation.enabled` - menyalakan rotasi log. Ini dinonaktifkan secara default dalam file konfigurasi Spark. Atur ke `true` untuk mengaktifkan fitur ini.
- `spark.eventLog.rotation.interval` - menentukan interval waktu untuk rotasi log. Nilai minimum adalah 60 detik. Nilai default adalah 300 detik.
- `spark.eventLog.rotation.minFileSize` - menentukan ukuran file minimum untuk merotasikan file log. Nilai minimum dan default adalah 1 MB.
- `spark.eventLog.rotation.maxFilesToRetain` - menentukan berapa banyak file log yang dirotasikan untuk disimpan selama pembersihan. Rentang validnya adalah 1 hingga 10. Nilai default adalah 2.

Anda dapat menentukan parameter ini dalam bagian `sparkSubmitParameters` dari API [StartJobRun](#), seperti yang ditunjukkan contoh berikut.

```
"sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi --conf  
spark.eventLog.rotation.enabled=true --conf spark.eventLog.rotation.interval=300 --  
conf spark.eventLog.rotation.minFileSize=1m --conf  
spark.eventLog.rotation.maxFilesToRetain=2"
```

Menggunakan rotasi log kontainer Spark

Dengan Amazon EMR 6.11.0 dan yang lebih baru, Anda dapat mengaktifkan fitur rotasi log kontainer Spark untuk Amazon EMR di EKS. Alih-alih menghasilkan file tunggal `stdout` atau `stderr` log, fitur ini memutar file berdasarkan ukuran rotasi yang dikonfigurasi dan menghapus file log tertua dari wadah.

Memutar log kontainer Spark dapat membantu Anda menghindari potensi masalah dengan file log Spark besar yang dihasilkan untuk pekerjaan yang berjalan lama atau streaming. Misalnya, Anda mungkin memulai pekerjaan Spark yang berjalan lama, dan driver Spark menghasilkan file log kontainer. Jika pekerjaan berjalan selama berjam-jam atau sehari-hari dan ruang disk terbatas pada node Kubernetes, file log kontainer dapat menggunakan semua ruang disk yang tersedia. Saat Anda mengaktifkan rotasi log kontainer Spark, Anda membagi file log menjadi beberapa file, dan menghapus file tertua.

Untuk mengaktifkan fitur rotasi log kontainer Spark, konfigurasi parameter Spark berikut:

containerLogRotationConfiguration

Sertakan parameter ini `monitoringConfiguration` untuk mengaktifkan rotasi log. Ini dinonaktifkan secara default. Anda harus menggunakan `containerLogRotationConfiguration` selain `s3MonitoringConfiguration`.

rotationSize

`rotationSizeParameter` menentukan ukuran file untuk rotasi log. Kisaran nilai yang mungkin adalah dari 2KB ke 2GB. Bagian unit numerik dari `rotationSize` parameter dilewatkan sebagai bilangan bulat. Karena nilai desimal tidak didukung, Anda dapat menentukan ukuran rotasi 1,5GB, misalnya, dengan nilai. `1500MB`

maxFilesToKeep

`maxFilesToKeepParameter` menentukan jumlah maksimum file untuk disimpan dalam wadah setelah rotasi telah terjadi. Nilai minimum adalah 1, dan nilai maksimum adalah 50.

Anda dapat menentukan parameter ini dalam bagian `monitoringConfiguration` dari API `StartJobRun`, seperti yang ditunjukkan contoh berikut. Dalam contoh ini, dengan `rotationSize = "10 MB"` dan `maxFilesToKeep = 3`, Amazon EMR di EKS memutar log Anda pada 10 MB, menghasilkan file log baru, dan kemudian membersihkan file log tertua setelah jumlah file log mencapai 3.

```
{
  "name": "my-long-running-job",
  "virtualClusterId": "123456",
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "emr-6.11.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "entryPoint_location",
      "entryPointArguments": ["argument1", "argument2", ...],
      "sparkSubmitParameters": "--class main_class --conf spark.executor.instances=2
--conf spark.executor.memory=2G --conf spark.executor.cores=2 --conf
spark.driver.cores=1"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
```

```
    "properties": {
      "spark.driver.memory": "2G"
    }
  ],
  "monitoringConfiguration": {
    "persistentAppUI": "ENABLED",
    "cloudWatchMonitoringConfiguration": {
      "logGroupName": "my_log_group",
      "logStreamNamePrefix": "log_stream_prefix"
    },
    "s3MonitoringConfiguration": {
      "logUri": "s3://my_s3_log_location"
    },
    "containerLogRotationConfiguration": {
      "rotationSize": "10MB",
      "maxFilesToKeep": "3"
    }
  }
}
```

Untuk memulai pekerjaan dengan rotasi log kontainer Spark, sertakan jalur ke file json yang Anda konfigurasi dengan parameter ini dalam perintah. [StartJobRun](#)

```
aws emr-containers start-job-run \  
--cli-input-json file://path-to-json-request-file
```

Menggunakan penskalaan otomatis vertikal dengan pekerjaan Amazon EMR Spark

Amazon EMR pada penskalaan otomatis vertikal EKS secara otomatis menyetel memori dan sumber daya CPU untuk beradaptasi dengan kebutuhan beban kerja yang Anda sediakan untuk aplikasi Amazon EMR Spark. Ini menyederhanakan manajemen sumber daya.

[Untuk melacak pemanfaatan sumber daya real-time dan historis dari aplikasi Amazon EMR Spark Anda, penskalaan otomatis vertikal memanfaatkan Kubernetes Vertical Pod Autoscaler \(VPA\).](#)

Kemampuan penskalaan otomatis vertikal menggunakan data yang dikumpulkan VPA untuk secara otomatis menyetel memori dan sumber daya CPU yang ditetapkan ke aplikasi Spark Anda. Proses yang disederhanakan ini meningkatkan keandalan dan mengoptimalkan biaya.

Topik

- [Menyiapkan penskalaan otomatis vertikal untuk Amazon EMR di EKS](#)
- [Memulai dengan penskalaan otomatis vertikal untuk Amazon EMR di EKS](#)
- [Mengkonfigurasi penskalaan otomatis vertikal untuk Amazon EMR di EKS](#)
- [Memantau penskalaan otomatis vertikal untuk Amazon EMR di EKS](#)
- [Copot pemasangan EMR Amazon pada operator penskalaan otomatis vertikal EKS](#)

Menyiapkan penskalaan otomatis vertikal untuk Amazon EMR di EKS

Topik ini membantu Anda menyiapkan kluster Amazon EKS Anda untuk mengirimkan pekerjaan Amazon EMR Spark dengan penskalaan otomatis vertikal. Proses penyiapan mengharuskan Anda untuk mengonfirmasi atau menyelesaikan tugas di bagian berikut:

Topik

- [Prasyarat](#)
- [Instal Operator Lifecycle Manager \(OLM\) di kluster Amazon EKS](#)
- [Instal EMR Amazon pada operator penskalaan otomatis vertikal EKS](#)

Prasyarat

Selesaikan tugas-tugas berikut sebelum Anda menginstal operator Kubernetes penskalaan otomatis vertikal di kluster Anda. Jika Anda telah menyelesaikan salah satu prasyarat, Anda dapat melewatinya dan melanjutkan ke yang berikutnya.

- [Instal atau perbarui ke versi terbaru AWS CLI](#) - Jika Anda sudah menginstal AWS CLI, konfirmasi bahwa Anda memiliki versi terbaru.
- [Instal kubectl](#) — kubectl adalah alat baris perintah yang Anda gunakan untuk berkomunikasi dengan server API Kubernetes. Anda memerlukan kubectl untuk menginstal dan memantau artefak terkait penskalaan otomatis vertikal di cluster Amazon EKS Anda.
- [Instal Operator SDK](#) — Amazon EMR di EKS menggunakan Operator SDK sebagai pengelola paket untuk masa pakai operator penskalaan otomatis vertikal yang Anda instal di cluster Anda.
- [Instal Docker](#) - Anda memerlukan akses ke CLI Docker untuk mengautentikasi dan mengambil gambar Docker terkait penskalaan otomatis vertikal untuk dipasang di cluster Amazon EKS Anda.
- [Instal server Kubernetes Metrics](#) — [Anda harus menginstal server](#) metrik terlebih dahulu agar autoscaler pod vertikal dapat mengambil metrik dari server API Kubernetes.

- [Memulai Amazon EKS — eksctl](#) (versi 1.24 atau lebih tinggi) — Penskalaan otomatis vertikal didukung dengan Amazon EKS versi 1.24 dan yang lebih tinggi. Setelah Anda membuat cluster, [daftar untuk digunakan dengan Amazon EMR](#).
- [Pilih URI gambar dasar EMR Amazon](#) (rilis 6.10.0 atau lebih tinggi) — Penskalaan otomatis vertikal didukung dengan rilis Amazon EMR 6.10.0 dan yang lebih tinggi.

Instal Operator Lifecycle Manager (OLM) di kluster Amazon EKS

Gunakan Operator SDK CLI untuk menginstal Operator Lifecycle Manager (OLM) di Amazon EMR pada kluster EKS tempat Anda ingin mengatur penskalaan otomatis vertikal, seperti yang ditunjukkan pada contoh berikut. Setelah mengaturnya, Anda dapat menggunakan OLM untuk menginstal dan mengelola siklus hidup operator penskalaan otomatis vertikal Amazon [EMR](#).

```
operator-sdk olm install
```

Untuk memvalidasi instalasi, jalankan `olm status` perintah:

```
operator-sdk olm status
```

Verifikasi bahwa perintah mengembalikan hasil yang sukses, mirip dengan contoh output berikut:

```
INFO[0007] Successfully got OLM status for version X.XX
```

Jika instalasi Anda tidak berhasil, lihat [Memecahkan masalah Amazon EMR pada penskalaan otomatis vertikal EKS](#).

Instal EMR Amazon pada operator penskalaan otomatis vertikal EKS

Gunakan langkah-langkah berikut untuk menginstal operator penskalaan otomatis vertikal di kluster Amazon EKS Anda:

1. Siapkan variabel lingkungan berikut yang akan Anda gunakan untuk menyelesaikan instalasi:
 - **\$REGION** menunjuk ke Wilayah AWS untuk cluster Anda. Misalnya, `us-west-2`.
 - **\$ACCOUNT_ID** menunjuk ke ID akun Amazon ECR untuk Wilayah Anda. Untuk informasi selengkapnya, lihat [Akun registri Amazon ECR berdasarkan Wilayah](#).

- **\$RELEASE** menunjuk ke rilis EMR Amazon yang ingin Anda gunakan untuk cluster Anda. Dengan penskalaan otomatis vertikal, Anda harus menggunakan Amazon EMR rilis 6.10.0 atau lebih tinggi.
2. Selanjutnya, dapatkan token otentikasi ke [registri Amazon ECR](#) untuk operator.

```
aws ecr get-login-password \  
  --region region-id | docker login \  
  --username AWS \  
  --password-stdin $ACCOUNT_ID.dkr.ecr.region-id.amazonaws.com
```

3. Instal EMR Amazon pada operator penskalaan otomatis vertikal EKS dengan perintah berikut:

```
ECR_URL=$ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com && \  
REPO_DEST=dynamic-sizing-k8s-operator-olm-bundle && \  
BUNDLE_IMG=emr-$RELEASE-dynamic-sizing-k8s-operator && \  
operator-sdk run bundle \  
$ECR_URL/$REPO_DEST/$BUNDLE_IMG\:latest
```

Ini akan membuat rilis operator penskalaan otomatis vertikal di namespace default kluster Amazon EKS Anda. Gunakan perintah ini untuk menginstal di namespace yang berbeda:

```
operator-sdk run bundle \  
$ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/dynamic-sizing-k8s-operator-olm-bundle/  
emr-$RELEASE-dynamic-sizing-k8s-operator\:latest \  
-n operator-namespace
```

Note

Jika namespace yang Anda tentukan tidak ada, OLM tidak akan menginstal operator. Untuk informasi selengkapnya, lihat [Namespace Kubernetes tidak ditemukan](#).

4. Verifikasi bahwa Anda berhasil menginstal operator dengan alat baris perintah kubectl Kubernetes.

```
kubectl get csv -n operator-namespace
```

kubect1Perintah harus mengembalikan operator autoscaler vertikal yang baru digunakan dengan status Fase Succeeded. Jika Anda mengalami masalah dengan instalasi atau pengaturan, lihat [Memecahkan masalah Amazon EMR pada penskalaan otomatis vertikal EKS](#).

Memulai dengan penskalaan otomatis vertikal untuk Amazon EMR di EKS

Gunakan penskalaan otomatis vertikal untuk Amazon EMR di EKS saat Anda ingin penyetelan otomatis memori dan sumber daya CPU untuk beradaptasi dengan beban kerja aplikasi Amazon EMR Spark Anda. Untuk informasi selengkapnya, lihat [Menggunakan penskalaan otomatis vertikal dengan pekerjaan Amazon EMR Spark](#).

Mengirimkan pekerjaan Spark dengan autoscaling vertikal

Saat Anda mengirimkan pekerjaan melalui [StartJobRun](#) API, tambahkan dua konfigurasi berikut ke driver untuk pekerjaan Spark Anda untuk mengaktifkan penskalaan otomatis vertikal:

```
"spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/  
dynamic.sizing":"true",  
"spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/  
dynamic.sizing.signature":"YOUR_JOB_SIGNATURE"
```

Pada kode di atas, baris pertama memungkinkan kemampuan autoscaling vertikal. Baris berikutnya adalah konfigurasi tanda tangan wajib yang memungkinkan Anda memilih tanda tangan untuk pekerjaan Anda.

Untuk informasi selengkapnya tentang konfigurasi ini dan nilai parameter yang dapat diterima, lihat [Mengkonfigurasi penskalaan otomatis vertikal untuk Amazon EMR di EKS](#). Secara default, pekerjaan Anda dikirimkan dalam mode Monitoring-Only Off dari penskalaan otomatis vertikal. Status pemantauan ini memungkinkan Anda menghitung dan melihat rekomendasi sumber daya tanpa melakukan penskalaan otomatis. Untuk informasi selengkapnya, lihat [Mode penskalaan otomatis vertikal](#).

Contoh berikut menunjukkan cara menyelesaikan `start-job-run` perintah sampel dengan autoscaling vertikal:

```
aws emr-containers start-job-run \  
--virtual-cluster-id $VIRTUAL_CLUSTER_ID \  
--name $JOB_NAME \  
--execution-role-arn $EMR_ROLE_ARN \  

```

```
--release-label emr-6.10.0-latest \
--job-driver '{
  "sparkSubmitJobDriver": {
    "entryPoint": "local:///usr/lib/spark/examples/src/main/python/pi.py"
  }
}' \
--configuration-overrides '{
  "applicationConfiguration": [{
    "classification": "spark-defaults",
    "properties": {
      "spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/
dynamic.sizing": "true",
      "spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/
dynamic.sizing.signature": "test-signature"
    }
  ]}
}'
```

Memverifikasi fungsionalitas penskalaan otomatis vertikal

Untuk memverifikasi bahwa penskalaan otomatis vertikal berfungsi dengan benar untuk pekerjaan yang dikirimkan, gunakan kubectl untuk mendapatkan sumber daya verticalpodautoscaler kustom dan melihat rekomendasi penskalaan Anda. Misalnya, permintaan perintah berikut untuk rekomendasi pada pekerjaan contoh dari [Mengirimkan pekerjaan Spark dengan autoscaling vertikal](#) bagian:

```
kubectl get verticalpodautoscalers --all-namespaces \
-l=emr-containers.amazonaws.com/dynamic.sizing.signature=test-signature
```

Output dari kueri ini harus menyerupai yang berikut:

NAME	MODE	CPU	MEM
PROVIDED AGE			
ds-jceyefkxnhrvdzw6djum3naf2abm6o63a6dvjkkedqtkhlrf25eq-vpa 87m	Off	3304504865	True

Jika output Anda tidak terlihat serupa atau berisi kode kesalahan, lihat langkah-langkah [Memecahkan masalah Amazon EMR pada penskalaan otomatis vertikal EKS](#) untuk membantu menyelesaikan masalah.

Mengonfigurasi penskalaan otomatis vertikal untuk Amazon EMR di EKS

Anda dapat mengonfigurasi penskalaan otomatis vertikal saat mengirimkan pekerjaan Amazon EMR Spark melalui API. [StartJobRun](#) Atur parameter konfigurasi terkait penskalaan otomatis pada pod driver Spark seperti yang ditunjukkan pada contoh di [Mengirimkan pekerjaan Spark dengan autoscaling vertikal](#)

Amazon EMR pada operator penskalaan otomatis vertikal EKS mendengarkan pod driver yang memiliki autoscaling, kemudian mengatur integrasi dengan Kubernetes Vertical Pod Autoscaler (VPA) dengan pengaturan pada pod driver. Ini memfasilitasi pelacakan sumber daya dan penskalaan otomatis pod pelaksana Spark.

Bagian berikut menjelaskan parameter yang dapat Anda gunakan saat mengonfigurasi penskalaan otomatis vertikal untuk kluster Amazon EKS Anda.

Note

Konfigurasi parameter sakelar fitur sebagai label, dan konfigurasi parameter yang tersisa sebagai anotasi pada pod driver Spark. Parameter penskalaan otomatis milik `emr-containers.amazonaws.com/` domain dan memiliki awalan `dynamic.sizing`

Parameter yang diperlukan

Anda harus menyertakan dua parameter berikut pada driver pekerjaan Spark saat Anda mengirimkan pekerjaan Anda:

Key	Deskripsi	Nilai yang diterima	Nilai default	Tipe	Parameter percikan 1
<code>dynamic.sizing</code>	Fitur toggle	<code>true, false</code>	tidak diatur	label	<code>spark.kubernetes.driver.label.emr-containers.amazonaws.com/dyna</code>

Key	Deskripsi	Nilai yang diterima	Nilai default	Tipe	Parameter percikan 1
					<code>mic.sizing</code>
<code>dynamic.sizing.signature</code>	Tanda tangan Job	tali	tidak diatur	anotasi	<code>spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.signature</code>

¹ Gunakan parameter ini sebagai `SparkSubmitParameter` atau `ConfigurationOverride` di `StartJobRun` API.

- **dynamic.sizing**— Anda dapat menghidupkan dan mematikan penskalaan otomatis vertikal dengan label `dynamic.sizing`. Untuk mengaktifkan penskalaan otomatis vertikal, atur `dynamic.sizing` ke `true` pada pod driver Spark. Jika Anda menghilangkan label ini atau mengaturnya ke nilai apa pun selain `true`, penskalaan otomatis vertikal tidak aktif.
- **dynamic.sizing.signature**— Atur tanda tangan pekerjaan dengan `dynamic.sizing.signature` anotasi pada pod driver. Penskalaan otomatis vertikal mengumpulkan data penggunaan sumber daya Anda di berbagai pekerjaan Amazon EMR Spark untuk mendapatkan rekomendasi sumber daya. Anda memberikan pengenal unik untuk mengikat pekerjaan bersama-sama.

Note

Jika pekerjaan Anda berulang pada interval tetap seperti harian atau mingguan, maka tanda tangan pekerjaan Anda harus tetap sama untuk setiap contoh pekerjaan baru. Ini

memastikan bahwa penskalaan otomatis vertikal dapat menghitung dan mengumpulkan rekomendasi di berbagai pekerjaan.

¹ Gunakan parameter ini sebagai `SparkSubmitParameter` atau `ConfigurationOverride` di `StartJobRun` API.

Parameter opsional

Penskalaan otomatis vertikal juga mendukung parameter opsional berikut. Tetapkan mereka sebagai anotasi pada pod driver.

Key	Deskripsi	Nilai yang diterima	Nilai default	Tipe	Parameter percikan 1
dynamic.sizing.mode	Mode penskalaan otomatis vertikal	Off, Initial, Auto	Off	anotasi	<code>spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.mode</code>
dynamic.sizing.scale.memory	Mengaktifkan penskalaan memori	<i>true, false</i>	true	anotasi	<code>spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.scale.memory</code>

Key	Deskripsi	Nilai yang diterima	Nilai default	Tipe	Parameter percikan 1
dynamic.sizing.scale.cpu	Aktifkan atau nonaktifkan penskalaan CPU	<i>true, false</i>	false	anotasi	spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.scale.cpu
dynamic.sizing.scale.memory.min	Batas minimum untuk penskalaan memori	string, kuantitas sumber daya K8s mis: 1G	tidak diatur	anotasi	spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.scale.memory.min

Key	Deskripsi	Nilai yang diterima	Nilai default	Tipe	Parameter percikan 1
dynamic.sizing.scale.memory.max	Batas maksimum untuk penskalaan memori	string, kuantitas sumber daya K8s mis: 4G	tidak diatur	anotasi	spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.scale.memory.max
dynamic.sizing.scale.cpu.min	Batas minimum untuk penskalaan CPU	string, kuantitas sumber daya K8s mis: 1	tidak diatur	anotasi	spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.scale.cpu.min

Key	Deskripsi	Nilai yang diterima	Nilai default	Tipe	Parameter percikan 1
dynamic.sizing.scale.cpu.max	Batas maksimum untuk penskalaan CPU	string, kuantitas sumber daya K8s mis: 2	tidak diatur	anotasi	spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.scale.cpu.max

Mode penskalaan otomatis vertikal

modeParameter memetakan ke berbagai mode penskalaan otomatis yang didukung VPA. Gunakan `dynamic.sizing.mode` anotasi pada pod driver untuk mengatur mode. Nilai-nilai berikut didukung untuk parameter ini:

- Off — Mode dry-run di mana Anda dapat memantau rekomendasi, tetapi penskalaan otomatis tidak dilakukan. Ini adalah mode default untuk penskalaan otomatis vertikal. Dalam mode ini, sumber daya autoscaler pod vertikal terkait menghitung rekomendasi, dan Anda dapat memantau rekomendasi melalui alat seperti kubectl, Prometheus, dan Grafana.
- Awal — Dalam mode ini, VPA menskalakan otomatis sumber daya saat pekerjaan dimulai jika rekomendasi tersedia berdasarkan riwayat pekerjaan, seperti dalam kasus pekerjaan berulang.
- Otomatis — Dalam mode ini, VPA mengusir pod pelaksana Spark, dan menskalakannya secara otomatis dengan pengaturan sumber daya yang disarankan saat pod driver Spark memulai ulang pod. Terkadang, VPA mengusir pod pelaksana Spark yang sedang berjalan, sehingga dapat menghasilkan latensi tambahan saat mencoba ulang eksekutor yang terputus.

Penskalaan sumber daya

Saat Anda mengatur penskalaan otomatis vertikal, Anda dapat memilih apakah akan menskalakan sumber daya CPU dan memori. Atur `dynamic.sizing.scale.cpu` dan

`dynamic.sizing.scale.memory` anotasi ke `true` atau `false`. Secara default, penskalaan CPU diatur ke `false`, dan penskalaan memori diatur ke `true`

Minimum dan maksimum sumber daya (Batas)

Secara opsional, Anda juga dapat menetapkan batasan pada CPU dan sumber daya memori. Pilih nilai minimum dan maksimum untuk sumber daya ini dengan `dynamic.sizing.[memory/cpu].[min/max]` anotasi saat Anda mengaktifkan penskalaan otomatis. Secara default, sumber daya tidak memiliki batasan. Tetapkan anotasi sebagai nilai string yang mewakili kuantitas sumber daya Kubernetes. Misalnya, atur `dynamic.sizing.memory.max 4G` untuk mewakili 4 GB.

Memantau penskalaan otomatis vertikal untuk Amazon EMR di EKS

Anda dapat menggunakan alat baris perintah `kubectl` Kubernetes untuk membuat daftar rekomendasi terkait penskalaan otomatis vertikal yang aktif di klaster Anda. Anda juga dapat melihat tanda tangan pekerjaan yang dilacak, dan membersihkan sumber daya yang tidak dibutuhkan yang terkait dengan tanda tangan.

Buat daftar rekomendasi penskalaan otomatis vertikal untuk klaster Anda

Gunakan `kubectl` untuk mendapatkan `verticalpodautoscaler` sumber daya, dan lihat status dan rekomendasi saat ini. Contoh kueri berikut mengembalikan semua sumber daya aktif di klaster Amazon EKS Anda.

```
kubectl get verticalpodautoscalers \
-o custom-columns="NAME:.metadata.name, \"
\"SIGNATURE:.metadata.labels.emr-containers\\.amazonaws\\.com/dynamic\\.sizing
\\.signature, \"
\"MODE:.spec.updatePolicy.updateMode, \"
\"MEM:.status.recommendation.containerRecommendations[0].target.memory\" \
--all-namespaces
```

Output dari kueri ini menyerupai yang berikut:

NAME	SIGNATURE	MODE	MEM
ds- <i>example-id-1</i> -vpa	<i>job-signature-1</i>	Off	<i>none</i>
ds- <i>example-id-2</i> -vpa	<i>job-signature-2</i>	Initial	12936384283

Kueri dan hapus rekomendasi penskalaan otomatis vertikal untuk kluster Anda

Saat Anda menghapus resource job-run penskalaan otomatis vertikal Amazon EMR, sumber daya ini secara otomatis menghapus objek VPA terkait yang melacak dan menyimpan rekomendasi.

Contoh berikut menggunakan kubectl untuk membersihkan rekomendasi untuk pekerjaan yang diidentifikasi dengan tanda tangan:

```
kubectl delete jobrun -n emr -l=emr-containers\.amazonaws\.com/dynamic\.sizing
\.signature=integ-test
jobrun.dynamicsizing.emr.services.k8s.aws "ds-job-signature" deleted
```

Jika Anda tidak mengetahui tanda tangan pekerjaan tertentu, atau ingin membersihkan semua sumber daya di kluster, Anda dapat menggunakan `--all` atau `--all-namespaces` dalam perintah Anda alih-alih ID pekerjaan unik, seperti yang ditunjukkan pada contoh berikut:

```
kubectl delete jobruns --all --all-namespaces
jobrun.dynamicsizing.emr.services.k8s.aws "ds-example-id" deleted
```

Copot pemasangan EMR Amazon pada operator penskalaan otomatis vertikal EKS

Jika Anda ingin menghapus operator penskalaan otomatis vertikal dari kluster Amazon EKS Anda, gunakan `cleanup` perintah dengan Operator SDK CLI seperti yang ditunjukkan pada contoh berikut. Ini juga menghapus dependensi upstream yang diinstal dengan operator, seperti Vertical Pod Autoscaler.

```
operator-sdk cleanup emr-dynamic-sizing
```

Jika ada pekerjaan yang berjalan di cluster saat Anda menghapus operator, pekerjaan tersebut terus berjalan tanpa penskalaan otomatis vertikal. [Jika Anda mengirimkan pekerjaan di kluster setelah menghapus operator, Amazon EMR di EKS akan mengabaikan parameter terkait penskalaan otomatis vertikal yang mungkin telah Anda tentukan selama konfigurasi.](#)

Menjalankan beban kerja interaktif di Amazon EMR di EKS

Endpoint interaktif adalah gateway yang menghubungkan Amazon EMR Studio ke Amazon EMR di EKS sehingga Anda dapat menjalankan beban kerja interaktif. [Anda dapat menggunakan endpoint interaktif dengan EMR Studio untuk menjalankan analisis interaktif dengan kumpulan data di penyimpanan data seperti Amazon S3 dan Amazon DynamoDB.](#)

Kasus penggunaan

- Buat skrip ETL dengan pengalaman EMR Studio IDE. IDE menyerap data lokal dan menyimpannya di Amazon S3 setelah transformasi untuk analisis selanjutnya.
- Gunakan buku catatan untuk menjelajahi kumpulan data dan melatih model pembelajaran mesin untuk mendeteksi anomali dalam kumpulan data.
- Buat skrip yang menghasilkan laporan harian untuk aplikasi analitik seperti dasbor bisnis.

Topik


- [Ikhtisar titik akhir interaktif](#)
- [Prasyarat untuk membuat titik akhir interaktif di Amazon EMR di EKS](#)
- [Membuat endpoint interaktif untuk kluster virtual Anda](#)
- [Mengkonfigurasi pengaturan untuk titik akhir interaktif](#)
- [Memantau titik akhir interaktif](#)
- [Menggunakan notebook Jupyter yang dihosting sendiri](#)
- [Mendapatkan informasi tentang titik akhir interaktif dengan perintah CLI](#)

Ikhtisar titik akhir interaktif

Endpoint interaktif menyediakan kemampuan untuk klien interaktif seperti Amazon EMR Studio untuk terhubung ke Amazon EMR pada kluster EKS untuk menjalankan beban kerja interaktif. Endpoint interaktif didukung oleh Jupyter Enterprise Gateway yang menyediakan kemampuan manajemen siklus hidup kernel jarak jauh yang dibutuhkan klien interaktif. Kernel adalah proses khusus bahasa yang berinteraksi dengan klien Amazon EMR Studio berbasis Jupyter untuk menjalankan beban kerja interaktif.

Endpoint interaktif mendukung kernel berikut:

- Python 3
- PySpark di Kubernetes
- Apache Spark dengan Scala

 Note

Amazon EMR pada harga EKS berlaku untuk titik akhir dan kernel interaktif. Untuk informasi lebih lanjut, lihat halaman [harga Amazon EMR di EKS](#).

Entitas berikut diperlukan agar EMR Studio terhubung dengan Amazon EMR di EKS.

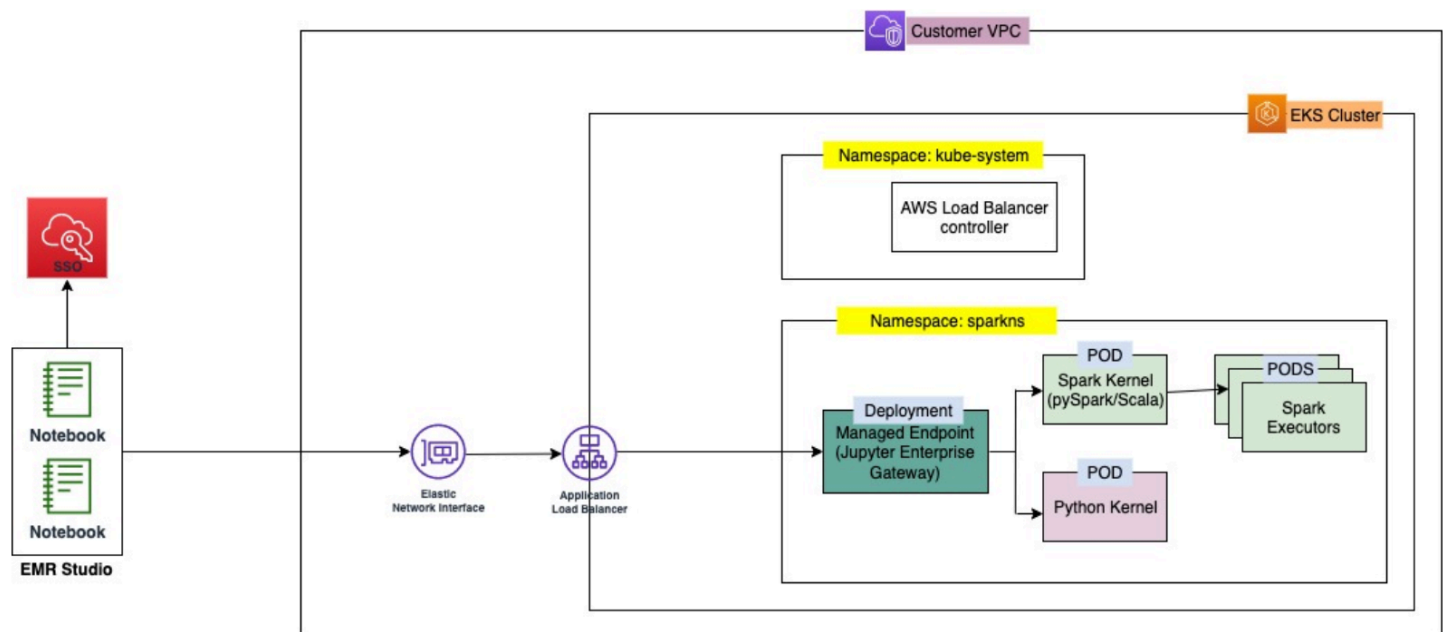
- Amazon EMR di kluster virtual EKS — Cluster virtual adalah namespace Kubernetes tempat Anda mendaftarkan EMR Amazon. Amazon EMR menggunakan kluster virtual untuk menjalankan tugas dan meng-host titik akhir. Anda dapat mendukung beberapa cluster virtual dengan cluster fisik yang sama. Namun, setiap cluster virtual memetakan ke satu namespace di cluster Amazon EKS. Cluster virtual tidak membuat sumber daya aktif apa pun yang berkontribusi pada tagihan Anda atau yang memerlukan manajemen siklus hidup di luar layanan.
- Amazon EMR pada titik akhir interaktif EKS - Titik akhir interaktif adalah titik akhir HTTPS tempat pengguna EMR Studio dapat menghubungkan ruang kerja. Anda hanya dapat mengakses titik akhir HTTPS dari EMR Studio Anda, dan Anda membuatnya di subnet pribadi Amazon Virtual Private Cloud (Amazon VPC) untuk kluster Amazon EKS Anda.

Kernel Python, PySpark, dan Spark Scala menggunakan izin yang ditentukan dalam EMR Amazon Anda pada peran eksekusi pekerjaan EKS untuk memanggil yang lain. Layanan AWS Semua kernel dan pengguna yang terhubung ke endpoint interaktif menggunakan peran yang Anda tentukan saat membuat endpoint. Kami menyarankan Anda membuat titik akhir terpisah untuk pengguna yang berbeda, dan bahwa pengguna memiliki peran AWS Identity and Access Management (IAM) yang berbeda.

- AWS Pengontrol Application Load Balancer — Pengontrol AWS Application Load Balancer mengelola Elastic Load Balancing untuk kluster Amazon EKS Kubernetes. Controller menyediakan Application Load Balancer (ALB) saat Anda membuat resource Kubernetes Ingress. ALB mengekspos layanan Kubernetes, seperti endpoint interaktif, di luar kluster Amazon EKS tetapi dalam VPC Amazon yang sama. Saat Anda membuat titik akhir interaktif, sumber daya Ingress juga digunakan yang mengekspos titik akhir interaktif melalui ALB untuk terhubung dengan klien

interaktif. Anda hanya perlu menginstal satu AWS Application Load Balancer controller untuk setiap cluster Amazon EKS.

Diagram berikut menggambarkan arsitektur endpoint interaktif di Amazon EMR di EKS. Cluster Amazon EKS terdiri dari komputasi untuk menjalankan beban kerja analitik, dan titik akhir interaktif. Pengontrol Application Load Balancer berjalan di kube-system namespace; beban kerja dan endpoint interaktif berjalan di namespace yang Anda tentukan saat Anda membuat cluster virtual. Saat Anda membuat titik akhir interaktif, EMR Amazon pada bidang kontrol EKS membuat penerapan titik akhir interaktif di kluster Amazon EKS. Selain itu, instance dari ingress penyeimbang beban aplikasi dibuat oleh pengontrol penyeimbang AWS beban. Penyeimbang beban aplikasi menyediakan antarmuka eksternal untuk klien seperti EMR Studio untuk terhubung ke cluster EMR Amazon dan menjalankan beban kerja interaktif.



Prasyarat untuk membuat titik akhir interaktif di Amazon EMR di EKS

Bagian ini menjelaskan prasyarat untuk menyiapkan titik akhir interaktif yang dapat digunakan EMR Studio untuk terhubung ke EMR Amazon di kluster EKS dan menjalankan beban kerja interaktif.

AWS CLI

Ikuti langkah-langkah di [Instal atau perbarui ke versi terbaru AWS CLI](#) untuk menginstal versi terbaru dari AWS Command Line Interface (AWS CLI).

Instalasi eksctl

Ikuti langkah-langkah di [Install kubectl](#) untuk menginstal versi terbaru eksctl. Jika Anda menggunakan Kubernetes versi 1.22 atau yang lebih baru untuk kluster Amazon EKS Anda, gunakan versi eksctl yang lebih besar dari 0.117.0.

Kluster Amazon EKS

Buat kluster Amazon EKS. Daftarkan cluster sebagai cluster virtual dengan Amazon EMR di EKS. Berikut ini adalah persyaratan dan pertimbangan untuk cluster ini:

- Cluster harus berada di Amazon Virtual Private Cloud (VPC) yang sama dengan EMR Studio Anda.
- Cluster harus memiliki setidaknya satu subnet pribadi untuk mengaktifkan endpoint interaktif, untuk menautkan repositori berbasis Git, dan untuk meluncurkan Application Load Balancer dalam mode pribadi.
- Harus ada minimal satu subnet privat yang sama antara EMR Studio Anda dan kluster Amazon EKS yang Anda gunakan untuk mendaftarkan kluster virtual Anda. Ini memastikan bahwa endpoint interaktif Anda muncul sebagai opsi di ruang kerja Studio Anda, dan mengaktifkan konektivitas dari Studio ke Application Load Balancer.

Ada dua metode yang dapat Anda pilih untuk menghubungkan Studio dan kluster Amazon EKS Anda:

- Buat kluster Amazon EKS dan kaitkan dengan subnet milik EMR Studio Anda.
- Atau, buat EMR Studio dan tentukan subnet pribadi untuk kluster Amazon EKS Anda.
- Amazon EKS dioptimalkan ARM Amazon Linux tidak AMIs didukung untuk Amazon EMR pada titik akhir interaktif EKS.
- Hanya [grup node terkelola Amazon EKS dan node](#) yang disediakan Karpenter yang didukung.

Berikan akses Cluster untuk Amazon EMR di EKS

Gunakan langkah-langkah dalam [Grant Cluster Access untuk Amazon EMR di EKS](#) untuk memberikan Amazon EMR di EKS akses ke namespace tertentu di cluster Anda.

Aktifkan IRSA di cluster Amazon EKS

Untuk mengaktifkan peran IAM untuk Akun Layanan (IRSA) di kluster Amazon EKS, ikuti langkah-langkah di [Aktifkan Peran IAM untuk Akun Layanan \(IRSA\)](#).

Buat peran eksekusi pekerjaan IAM

Anda harus membuat peran IAM untuk menjalankan beban kerja di Amazon EMR pada titik akhir interaktif EKS. Kami menyebut peran IAM ini sebagai peran eksekusi pekerjaan dalam dokumentasi ini. Peran IAM ini akan ditetapkan ke wadah endpoint interaktif dan kontainer eksekusi aktual yang dibuat saat Anda mengirimkan pekerjaan dengan EMR Studio. Anda memerlukan Nama Sumber Daya Amazon (ARN) dari peran eksekusi pekerjaan Anda untuk Amazon EMR di EKS. Ada dua langkah yang diperlukan untuk ini:

- [Buat peran IAM untuk eksekusi pekerjaan.](#)
- [Perbarui kebijakan kepercayaan dari peran eksekusi pekerjaan.](#)

Memberikan pengguna akses ke Amazon EMR di EKS

Entitas IAM (pengguna atau peran) yang membuat permintaan untuk membuat titik akhir interaktif juga harus memiliki `Amazon emr-containers EC2` dan izin berikut. Ikuti langkah-langkah yang dijelaskan [Memberikan pengguna akses ke Amazon EMR di EKS](#) untuk memberikan izin ini yang memungkinkan Amazon EMR di EKS membuat, mengelola, dan menghapus grup keamanan yang membatasi lalu lintas masuk ke penyeimbang beban titik akhir interaktif Anda.

`emr-containers` izin berikut memungkinkan pengguna untuk melakukan operasi endpoint interaktif dasar:

```
"ec2:CreateSecurityGroup",
"ec2:DeleteSecurityGroup",
"ec2:AuthorizeSecurityGroupEgress",
"ec2:AuthorizeSecurityGroupIngress",
"ec2:RevokeSecurityGroupEgress",
"ec2:RevokeSecurityGroupIngress"

"emr-containers:CreateManagedEndpoint",
"emr-containers:ListManagedEndpoints",
"emr-containers:DescribeManagedEndpoint",
"emr-containers>DeleteManagedEndpoint"
```

Daftarkan kluster Amazon EKS dengan Amazon EMR

Siapkan cluster virtual dan petakan ke namespace di kluster Amazon EKS tempat Anda ingin menjalankan pekerjaan Anda. Untuk cluster AWS Fargate-only, gunakan namespace yang sama untuk EMR Amazon di kluster virtual EKS dan profil Fargate.

Untuk informasi tentang pengaturan EMR Amazon di kluster virtual EKS, lihat [Daftarkan kluster Amazon EKS dengan Amazon EMR](#)

Menerapkan AWS Load Balancer Controller ke kluster Amazon EKS

AWS Application Load Balancer diperlukan untuk kluster Amazon EKS Anda. Anda hanya perlu menyiapkan satu pengontrol Application Load Balancer per kluster Amazon EKS. Untuk informasi tentang cara menyiapkan pengontrol AWS Application Load Balancer, lihat [Menginstal add-on Load AWS Balancer Controller di Panduan](#) Pengguna Amazon EKS.

Membuat endpoint interaktif untuk kluster virtual Anda

Topik ini menjelaskan beberapa cara untuk membuat endpoint interaktif menggunakan AWS Command Line Interface (AWS CLI) dan menyertakan detail tentang parameter konfigurasi yang tersedia.

Buat endpoint interaktif dengan perintah **create-managed-endpoint**

Tentukan parameter dalam `create-managed-endpoint` perintah sebagai berikut. Amazon EMR di EKS mendukung pembuatan endpoint interaktif dengan Amazon EMR rilis 6.7.0 dan yang lebih tinggi.

```
aws emr-containers create-managed-endpoint \  
--type JUPYTER_ENTERPRISE_GATEWAY \  
--virtual-cluster-id 1234567890abcdef0xxxxxxxx \  
--name example-endpoint-name \  
--execution-role-arn arn:aws:iam::444455556666:role/JobExecutionRole \  
--release-label emr-6.9.0-latest \  
--configuration-overrides '{  
  "applicationConfiguration": [{  
    "classification": "spark-defaults",  
    "properties": {  
      "spark.driver.memory": "2G"  
    }  
  }  
}],  
"monitoringConfiguration": {
```

```

    "cloudWatchMonitoringConfiguration": {
      "logGroupName": "log_group_name",
      "logStreamNamePrefix": "log_stream_prefix"
    },
    "persistentAppUI": "ENABLED",
    "s3MonitoringConfiguration": {
      "logUri": "s3://my_s3_log_location"
    }
  }
}'

```

Untuk informasi selengkapnya, lihat [Parameter untuk membuat endpoint interaktif](#).

Buat endpoint interaktif dengan parameter tertentu dalam file JSON

1. Buat `create-managed-endpoint-request.json` file dan tentukan parameter yang diperlukan untuk titik akhir Anda, seperti yang ditunjukkan pada file JSON berikut:

```

{
  "name": "MY_TEST_ENDPOINT",
  "virtualClusterId": "MY_CLUSTER_ID",
  "type": "JUPYTER_ENTERPRISE_GATEWAY",
  "releaseLabel": "emr-6.9.0-latest",
  "executionRoleArn": "arn:aws:iam::444455556666:role/JobExecutionRole",
  "configurationOverrides":
  {
    "applicationConfiguration":
    [
      {
        "classification": "spark-defaults",
        "properties":
        {
          "spark.driver.memory": "8G"
        }
      }
    ],
    "monitoringConfiguration":
    {
      "persistentAppUI": "ENABLED",
      "cloudWatchMonitoringConfiguration":
      {
        "logGroupName": "my_log_group",
        "logStreamNamePrefix": "log_stream_prefix"
      }
    }
  }
}

```

```

    },
    "s3MonitoringConfiguration":
    {
        "logUri": "s3://my_s3_log_location"
    }
}
}
}

```

- Gunakan `create-managed-endpoint` perintah dengan jalur ke `create-managed-endpoint-request.json` file yang disimpan secara lokal atau di Amazon S3.

```
aws emr-containers create-managed-endpoint \
--cli-input-json file:///./create-managed-endpoint-request.json --region AWS-Region
```

Output dari membuat endpoint interaktif

Anda akan melihat output berikut di terminal. Outputnya mencakup nama dan pengenal titik akhir interaktif baru Anda:

```

{
  "id": "1234567890abcdef0",
  "name": "example-endpoint-name",
  "arn": "arn:aws:emr-containers:us-west-2:111122223333:/
virtualclusters/444455556666/endpoints/444455556666",
  "virtualClusterId": "111122223333xxxxxxxxx"
}

```

Running `aws emr-containers create-managed-endpoint` membuat sertifikat yang ditandatangani sendiri yang memungkinkan komunikasi HTTPS antara EMR Studio dan server endpoint interaktif.

Jika Anda menjalankan `create-managed-endpoint` dan belum menyelesaikan prasyarat, Amazon EMR mengembalikan pesan kesalahan dengan tindakan yang harus Anda ambil untuk melanjutkan.

Parameter untuk membuat endpoint interaktif

Topik

- [Parameter yang diperlukan untuk titik akhir interaktif](#)
- [Parameter opsional untuk titik akhir interaktif](#)

Parameter yang diperlukan untuk titik akhir interaktif

Anda harus menentukan parameter berikut saat membuat endpoint interaktif:

--type

Gunakan `JUPYTER_ENTERPRISE_GATEWAY`. Ini adalah satu-satunya jenis yang didukung.

--virtual-cluster-id

Pengidentifikasi cluster virtual yang Anda daftarkan dengan Amazon EMR di EKS.

--name

Nama deskriptif untuk endpoint interaktif yang membantu pengguna EMR Studio memilihnya dari daftar dropdown.

--execution-role-arn

Nama Sumber Daya Amazon (ARN) dari peran eksekusi pekerjaan IAM Anda untuk Amazon EMR di EKS yang dibuat sebagai bagian dari prasyarat.

--release-label

Label rilis EMR Amazon untuk digunakan untuk titik akhir. Misalnya, `emr-6.9.0-latest`. Amazon EMR di EKS mendukung endpoint interaktif dengan Amazon EMR rilis 6.7.0 dan lebih tinggi.

Parameter opsional untuk titik akhir interaktif

Secara opsional, Anda juga dapat menentukan parameter berikut saat membuat endpoint interaktif:

--configuration-overrides

Untuk mengganti konfigurasi default untuk aplikasi, berikan objek konfigurasi. Anda dapat menggunakan sintaks singkatan untuk menyediakan konfigurasi, atau Anda dapat mereferensikan objek konfigurasi dalam file JSON.

Objek konfigurasi terdiri dari klasifikasi, properti, dan konfigurasi bersarang opsional. Properti terdiri dari pengaturan yang ingin Anda timpa dalam file itu. Anda dapat menentukan beberapa klasifikasi untuk beberapa aplikasi dalam objek JSON tunggal. Klasifikasi konfigurasi yang tersedia bervariasi menurut Amazon EMR pada rilis EKS. Untuk daftar klasifikasi konfigurasi yang tersedia untuk setiap

rilis Amazon EMR di EKS, lihat. [Amazon EMR pada rilis EKS](#) Selain klasifikasi konfigurasi yang terdaftar untuk setiap rilis, titik akhir interaktif membawa klasifikasi tambahan. `jeg-config` Untuk informasi selengkapnya, lihat [Opsi konfigurasi Jupyter Enterprise Gateway \(JEG\)](#).

Mengkonfigurasi pengaturan untuk titik akhir interaktif

Bagian ini berisi serangkaian topik yang mencakup berbagai konfigurasi untuk titik akhir interaktif dan pengaturan pod. Ini memberi Anda kemampuan untuk memantau dan memecahkan masalah kegagalan, mengirim informasi log ke Amazon S3 atau ke, atau Amazon CloudWatch Logs untuk membuat titik akhir interaktif tempat Anda menentukan templat pod kustom.

Topik

- [Lowongan kerja Monitoring Spark](#)
- [Menentukan template pod kustom dengan endpoint interaktif](#)
- [Menerapkan pod JEG ke grup node](#)
- [Opsi konfigurasi Jupyter Enterprise Gateway \(JEG\)](#)
- [Memodifikasi PySpark parameter sesi](#)
- [Gambar kernel kustom dengan endpoint interaktif](#)

Lowongan kerja Monitoring Spark

Agar Anda dapat memantau dan memecahkan masalah kegagalan, konfigurasi titik akhir interaktif Anda sehingga pekerjaan yang dimulai dengan titik akhir dapat mengirim informasi log ke Amazon S3, Amazon Log, atau keduanya. CloudWatch Bagian berikut menjelaskan cara mengirim log aplikasi Spark ke Amazon S3 untuk pekerjaan Spark yang Anda luncurkan dengan Amazon EMR di titik akhir interaktif EKS.

Konfigurasi kebijakan IAM untuk log Amazon S3

Sebelum kernel Anda dapat mengirim data log ke Amazon S3, kebijakan izin untuk peran eksekusi pekerjaan harus menyertakan izin berikut. Ganti `amzn-s3-demo-destination-bucket` dengan nama bucket logging Anda.

JSON

```
{
```

```
"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "s3:PutObject",
      "s3:GetObject",
      "s3:ListBucket"
    ],
    "Resource": [
      "arn:aws:s3:::amzn-s3-demo-bucket",
      "arn:aws:s3:::amzn-s3-demo-bucket/*"
    ],
    "Sid": "AllowS3Putobject"
  }
]
```

Note

Amazon EMR di EKS juga dapat membuat bucket S3. Jika bucket S3 tidak tersedia, sertakan `s3:CreateBucket` izin tersebut dalam kebijakan IAM.

Setelah Anda memberikan izin pada peran eksekusi yang diperlukan untuk mengirim log ke bucket S3, data log Anda akan dikirim ke lokasi Amazon S3 berikut. Ini terjadi ketika `s3MonitoringConfiguration` diteruskan di `monitoringConfiguration` bagian `create-managed-endpoint` permintaan.

- Log driver — `logUri/virtual-cluster-id/endpoints/endpoint-id/containers/spark-application-id/spark-application-id-driver/(stderr.gz/stdout.gz)`
- Log pelaksana — `logUri/virtual-cluster-id/endpoints/endpoint-id/containers/spark-application-id/executor-pod-name-exec-<Number>/(stderr.gz/stdout.gz)`

Note

Amazon EMR di EKS tidak mengunggah log titik akhir ke bucket S3 Anda.

Menentukan template pod kustom dengan endpoint interaktif

Anda dapat membuat endpoint interaktif di mana Anda menentukan template pod kustom untuk driver dan pelaksana. Template Pod adalah spesifikasi yang menentukan cara menjalankan setiap pod. Anda dapat menggunakan file template pod untuk menentukan konfigurasi driver atau pod pelaksana yang tidak didukung oleh konfigurasi Spark. Template Pod saat ini didukung di Amazon EMR rilis 6.3.0 dan yang lebih baru.

Untuk informasi selengkapnya tentang template pod, lihat [Menggunakan templat pod](#) di Amazon EMR on EKS Development Guide.

Contoh berikut menunjukkan cara membuat endpoint interaktif dengan template pod:

```
aws emr-containers create-managed-endpoint \  
  --type JUPYTER_ENTERPRISE_GATEWAY \  
  --virtual-cluster-id virtual-cluster-id \  
  --name example-endpoint-name \  
  --execution-role-arn arn:aws:iam::aws-account-id:role/EKSClusterRole \  
  --release-label emr-6.9.0-latest \  
  --configuration-overrides '{  
    "applicationConfiguration": [  
      {  
        "classification": "spark-defaults",  
        "properties": {  
          "spark.kubernetes.driver.podTemplateFile": "path/to/driver/  
template.yaml",  
          "spark.kubernetes.executor.podTemplateFile": "path/to/executor/  
template.yaml"  
        }  
      }  
    ]  
  }'
```

Menerapkan pod JEG ke grup node

Penempatan pod JEG (Jupyter Enterprise Gateway) adalah fitur yang memungkinkan Anda untuk menerapkan endpoint interaktif pada grup node tertentu. Dengan fitur ini, Anda dapat mengonfigurasi pengaturan seperti `instance type` untuk titik akhir interaktif.

Mengaitkan pod JEG ke grup node terkelola

Properti konfigurasi berikut memungkinkan Anda menentukan nama grup node terkelola di kluster Amazon EKS tempat pod JEG akan di-deploy.

```
//payload
--configuration-overrides '{
  "applicationConfiguration": [
    {
      "classification": "endpoint-configuration",
      "properties": {
        "managed-nodegroup-name": NodeGroupName
      }
    }
  ]
}'
```

Sebuah grup node harus memiliki label Kubernetes yang `for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName` melekat pada semua node yang merupakan bagian dari grup node. Untuk membuat daftar semua node dari grup node yang memiliki tag ini, gunakan perintah berikut:

```
kubectl get nodes --show-labels | grep for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName
```

Jika output dari perintah di atas tidak mengembalikan node yang merupakan bagian dari grup node terkelola Anda, maka tidak ada node dalam grup node yang memiliki label `for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName` Kubernetes terpasang. Dalam hal ini, ikuti langkah-langkah di bawah ini untuk melampirkan label tersebut ke node di grup node Anda.

1. Gunakan perintah berikut untuk menambahkan label `for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName` Kubernetes ke semua node dalam grup node terkelola: *NodeGroupName*

```
kubectl label nodes --selector eks:nodegroup-name=NodeGroupName for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName
```

2. Verifikasi bahwa node diberi label dengan benar menggunakan perintah berikut:

```
kubectl get nodes --show-labels | grep for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName
```

Grup node terkelola harus dikaitkan dengan grup keamanan klaster Amazon EKS, yang biasanya terjadi jika Anda membuat cluster dan grup node terkelola menggunakan `eksctl`. Anda dapat memverifikasi ini di AWS konsol menggunakan langkah-langkah berikut.

1. Buka cluster Anda di konsol Amazon EKS.
2. Buka tab jaringan cluster Anda dan catat grup keamanan cluster.
3. Buka tab komputasi klaster Anda dan klik pada nama grup node terkelola.
4. Di bawah tab Detail grup node terkelola, verifikasi bahwa grup keamanan klaster yang Anda catat sebelumnya terdaftar di bawah Grup keamanan.

Jika grup node terkelola tidak dilampirkan ke grup keamanan klaster Amazon EKS, Anda harus melampirkan `for-use-with-emr-containers-managed-endpoint-sg=ClusterName/NodeGroupName` tag ke grup keamanan grup node. Gunakan langkah-langkah di bawah ini untuk melampirkan tag ini.

1. Buka konsol Amazon EC2 dan klik grup keamanan di panel navigasi kiri.
2. Pilih grup keamanan grup node terkelola Anda dengan mengklik kotak centang.
3. Di bawah tab Tag, tambahkan tag `for-use-with-emr-containers-managed-endpoint-sg=ClusterName/NodeGroupName` menggunakan tombol Kelola tag.

Mengaitkan pod JEG ke grup node yang dikelola sendiri

Properti konfigurasi berikut memungkinkan Anda menentukan nama grup node yang dikelola sendiri atau tidak dikelola di klaster Amazon EKS tempat pod JEG akan di-deploy.

```
//payload
--configuration-overrides '{
  "applicationConfiguration": [
    {
      "classification": "endpoint-configuration",
      "properties": {
        "self-managed-nodegroup-name": NodeGroupName
      }
    }
  ]
}'
```

Grup node harus memiliki label `for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName` Kubernetes yang melekat pada semua node yang merupakan bagian dari grup node. Untuk membuat daftar semua node dari grup node yang memiliki tag ini, gunakan perintah berikut:

```
kubectl get nodes --show-labels | grep for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName
```

Jika output dari perintah di atas tidak mengembalikan node yang merupakan bagian dari grup node yang dikelola sendiri, maka tidak ada node di grup node yang memiliki label `for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName` Kubernetes yang terpasang. Dalam hal ini, ikuti langkah-langkah di bawah ini untuk melampirkan label tersebut ke node di grup node Anda.

1. Jika Anda membuat grup node yang dikelola sendiri menggunakan `eksctl`, maka gunakan perintah berikut untuk menambahkan label `for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName` Kubernetes ke semua node dalam grup node yang dikelola sendiri sekaligus. *NodeGroupName*

```
kubectl label nodes --selector alpha.eksctl.io/nodegroup-name=NodeGroupName for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName
```

Jika Anda tidak menggunakannya `eksctl` untuk membuat grup node yang dikelola sendiri, maka Anda perlu mengganti pemilih pada perintah di atas ke label Kubernetes yang berbeda yang dilampirkan ke semua node dari grup node.

2. Gunakan perintah berikut untuk memverifikasi bahwa node diberi label dengan benar:

```
kubectl get nodes --show-labels | grep for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName
```

Grup keamanan untuk grup node yang dikelola sendiri harus memiliki `for-use-with-emr-containers-managed-endpoint-sg=ClusterName/NodeGroupName` tag yang dilampirkan. Gunakan langkah-langkah berikut untuk melampirkan tag ke grup keamanan dari Konsol Manajemen AWS.

1. Arahkan ke konsol Amazon EC2. Pilih Grup keamanan di panel navigasi kiri.
2. Pilih kotak centang di samping grup keamanan untuk grup node yang dikelola sendiri.

- Di bawah tab Tag, gunakan tombol Kelola tag untuk menambahkan tag `for-use-with-emr-containers-managed-endpoint-sg=ClusterName/NodeGroupName`. Ganti *ClusterName* dan *NodeGroupName* dengan nilai yang sesuai.

Mengaitkan pod JEG ke grup node terkelola dengan instans On-Demand

Anda juga dapat menentukan label tambahan, yang dikenal sebagai pemilih label Kubernetes, untuk menentukan batasan atau batasan tambahan untuk menjalankan titik akhir interaktif pada node atau grup node tertentu. Contoh berikut menunjukkan cara menggunakan instans Amazon EC2 On-Demand untuk pod JEG.

```
--configuration-overrides '{
  "applicationConfiguration": [
    {
      "classification": "endpoint-configuration",
      "properties": {
        "managed-nodegroup-name": NodeGroupName,
        "node-labels": "eks.amazonaws.com/capacityType:ON_DEMAND"
      }
    }
  ]
}'
```

Note

Anda hanya dapat menggunakan `node-labels` properti dengan `self-managed-nodegroup-name` properti `managed-nodegroup-name` atau properti.

Opsi konfigurasi Jupyter Enterprise Gateway (JEG)

Amazon EMR di EKS menggunakan Jupyter Enterprise Gateway (JEG) untuk mengaktifkan endpoint interaktif. Anda dapat mengatur nilai berikut untuk konfigurasi JEG yang diizinkan saat Anda membuat titik akhir.

- `RemoteMappingKernelManager.cull_idle_timeout`**— Timeout dalam hitungan detik (integer), setelah itu kernel dianggap idle dan siap untuk dimusnahkan. Nilai 0 atau lebih rendah menonaktifkan pemusnahan. Batas waktu yang singkat dapat mengakibatkan kernel dimusnahkan untuk pengguna dengan koneksi jaringan yang buruk.

- **RemoteMappingKernelManager.cull_interval**— Interval dalam detik (integer) untuk memeriksa kernel idle yang melebihi nilai batas waktu pemusnahan.

Memodifikasi PySpark parameter sesi

Dimulai dengan Amazon EMR pada rilis EKS 6.9.0, di Amazon EMR Studio Anda dapat menyesuaikan konfigurasi Spark yang terkait dengan PySpark sesi dengan menjalankan perintah `%configure` ajaib di sel notebook EMR.

Contoh berikut menunjukkan payload sampel yang dapat Anda gunakan untuk memodifikasi memori, core, dan properti lainnya untuk driver dan eksekutor Spark. Untuk `conf` pengaturan, Anda dapat mengonfigurasi konfigurasi Spark apa pun yang disebutkan dalam dokumentasi konfigurasi [Apache Spark](#).

```
%%configure -f
{
  "driverMemory": "16G",
  "driverCores": 4,
  "executorMemory" : "32G",
  "executorCores": 2,
  "conf": {
    "spark.dynamicAllocation.maxExecutors" : 10,
    "spark.dynamicAllocation.minExecutors": 1
  }
}
```

Contoh berikut menunjukkan contoh payload yang dapat Anda gunakan untuk menambahkan file, PyFiles, dan dependensi jar ke runtime Spark.

```
%%configure -f
{
  "files": "s3://amzn-s3-demo-bucket-emr-eks/sample_file.txt",
  "pyFiles": : "path-to-python-files",
  "jars" : "path-to-jars"
}
```

Gambar kernel kustom dengan endpoint interaktif

Untuk memastikan bahwa Anda memiliki dependensi yang benar untuk aplikasi saat menjalankan beban kerja interaktif dari Amazon EMR Studio, Anda dapat menyesuaikan gambar Docker untuk

titik akhir interaktif dan menjalankan image kernel dasar yang disesuaikan. Untuk membuat endpoint interaktif dan menghubungkannya dengan image Docker kustom, lakukan langkah-langkah berikut.

Note

Anda hanya dapat mengganti gambar dasar. Anda tidak dapat menambahkan jenis gambar kernel baru.

1. Buat dan publikasikan gambar Docker yang disesuaikan. Gambar dasar berisi runtime Spark dan kernel notebook yang berjalan dengannya. Untuk membuat gambar, Anda dapat mengikuti langkah 1 hingga 4 in [Cara menyesuaikan gambar Docker](#). Pada langkah 1, URI gambar dasar dalam file Docker Anda harus digunakan notebook-spark sebagai pengganti. spark

```
ECR-registry-account.dkr.ecr.Region.amazonaws.com/notebook-spark/container-image-tag
```

Untuk informasi selengkapnya tentang cara memilih Wilayah AWS dan menampung tag gambar, lihat [Detail untuk memilih URI gambar dasar](#).

2. Buat endpoint interaktif yang dapat digunakan dengan gambar kustom.
 - a. Buat file JSON `custom-image-managed-endpoint.json` dengan konten berikut. Contoh ini menggunakan Amazon EMR rilis 6.9.0.

Example

```
{
  "name": "endpoint-name",
  "virtualClusterId": "virtual-cluster-id",
  "type": "JUPYTER_ENTERPRISE_GATEWAY",
  "releaseLabel": "emr-6.9.0-latest",
  "executionRoleArn": "execution-role-arn",
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "jupyter-kernel-overrides",
        "configurations": [
          {
            "classification": "python3",
            "properties": {
```

```

        "container-image": "123456789012.dkr.ecr.us-
west-2.amazonaws.com/custom-notebook-python:latest"
    },
    {
        "classification": "spark-python-kubernetes",
        "properties": {
            "container-image": "123456789012.dkr.ecr.us-
west-2.amazonaws.com/custom-notebook-spark:latest"
        }
    }
]
}

```

- b. Buat endpoint interaktif dengan konfigurasi yang ditentukan dalam file JSON seperti yang ditunjukkan pada contoh berikut. Untuk informasi selengkapnya, lihat [Buat endpoint interaktif dengan perintah create-managed-endpoint](#).

```
aws emr-containers create-managed-endpoint --cli-input-json custom-image-
managed-endpoint.json
```

3. Connect ke endpoint interaktif melalui EMR Studio. Untuk informasi selengkapnya dan langkah-langkah yang harus diselesaikan, lihat [Menghubungkan dari Studio](#) di bagian Amazon EMR di EKS pada dokumen AWS Workshop Studio.

Memantau titik akhir interaktif

Dengan Amazon EMR di EKS versi 6.10 dan yang lebih baru, titik akhir interaktif memancarkan metrik CloudWatch Amazon untuk memantau dan memecahkan masalah operasi siklus hidup kernel. Metrik dipicu oleh klien interaktif, seperti EMR Studio atau notebook Jupyter yang dihosting sendiri. Setiap operasi yang didukung oleh endpoint interaktif memiliki metrik yang terkait dengannya. Operasi dimodelkan sebagai dimensi untuk setiap metrik, seperti yang ditunjukkan pada tabel di bawah ini. Metrik yang dipancarkan oleh titik akhir interaktif terlihat di bawah namespace khusus, di akun Anda. EMRContainers

Metrik	Deskripsi	Unit
RequestCount	Jumlah kumulatif permintaan operasi yang diproses oleh endpoint interaktif.	Hitungan
RequestLatency	Waktu dari ketika permintaan tiba di titik akhir interaktif dan respons dikirim oleh titik akhir interaktif.	Milidetik
4 XXError	Dipancarkan ketika permintaan untuk operasi menghasilkan kesalahan 4xx selama pemrosesan.	Hitungan
5 XXError	Dipancarkan saat permintaan operasi menghasilkan kesalahan sisi server 5Xxx.	Hitungan
KernelLaunchSuccess	Hanya berlaku untuk CreateKernel operasi. Ini menunjukkan jumlah kumulatif peluncuran kernel yang berhasil hingga dan termasuk permintaan ini.	Hitungan
KernelLaunchFailure	Hanya berlaku untuk CreateKernel operasi. Ini menunjukkan jumlah kumulatif kegagalan peluncuran kernel hingga dan termasuk permintaan ini.	Hitungan

Setiap metrik endpoint interaktif memiliki dimensi berikut yang melekat padanya:

- **ManagedEndpointId**— Pengidentifikasi untuk titik akhir interaktif

- **OperationName**— Operasi yang dipicu oleh klien interaktif

Nilai yang mungkin untuk **OperationName** dimensi ditunjukkan pada tabel berikut:

operationName	Deskripsi operasi
CreateKernel	Minta endpoint interaktif memulai kernel.
ListKernels	Minta agar titik akhir interaktif mencantumkan kernel yang sebelumnya telah dimulai menggunakan token sesi yang sama.
GetKernel	Minta agar endpoint interaktif mendapatkan detail tentang kernel tertentu yang telah dimulai sebelumnya.
ConnectKernel	Minta endpoint interaktif membangun konektivitas antara klien notebook dan kernel.
ConfigureKernel	Publikasikan <code>%%configure magic request</code> pada kernel pyspark.
ListKernelSpecs	Minta agar titik akhir interaktif mencantumkan spesifikasi kernel yang tersedia.
GetKernelSpec	Minta agar endpoint interaktif mendapatkan spesifikasi kernel dari kernel yang telah diluncurkan sebelumnya.
GetKernelSpecResource	Minta agar endpoint interaktif mendapatkan sumber daya spesifik yang terkait dengan spesifikasi kernel yang telah diluncurkan sebelumnya.

Contoh

Untuk mengakses jumlah kernel yang diluncurkan untuk titik akhir interaktif pada hari tertentu:

1. Pilih namespace kustom: `EMRContainers`
2. Pilih `AndaManagedEndpointId`, `OperationName - CreateKernel`
3. `RequestCount` metrik dengan statistik SUM dan periode 1 day akan memberikan semua permintaan peluncuran kernel yang dibuat dalam 24 jam terakhir.
4. `KernelLaunchSuccess` metrik dengan statistik SUM dan periode 1 day akan memberikan semua permintaan peluncuran kernel yang berhasil dibuat dalam 24 jam terakhir.

Untuk mengakses jumlah kegagalan kernel untuk endpoint interaktif pada hari tertentu:

1. Pilih namespace kustom: `EMRContainers`
2. Pilih `AndaManagedEndpointId`, `OperationName - CreateKernel`
3. `KernelLaunchFailure` metrik dengan statistik SUM dan periode 1 day akan memberikan semua permintaan peluncuran kernel gagal yang dibuat dalam 24 jam terakhir. Anda juga dapat memilih `5XXError` metrik `4XXError` dan untuk mengetahui jenis kegagalan peluncuran kernel yang terjadi.

Menggunakan notebook Jupyter yang dihosting sendiri

Anda dapat meng-host dan mengelola Jupyter atau JupyterLab notebook di instans Amazon EC2 atau di cluster Amazon EKS Anda sendiri sebagai notebook Jupyter yang dihosting sendiri. Anda kemudian dapat menjalankan beban kerja interaktif dengan notebook Jupyter yang dihosting sendiri. Bagian berikut berjalan melalui proses untuk menyiapkan dan menerapkan notebook Jupyter yang dihosting sendiri di cluster Amazon EKS.

Membuat notebook Jupyter yang dihosting sendiri di cluster EKS

- [Membuat grup keamanan](#)
- [Buat EMR Amazon di titik akhir interaktif EKS](#)
- [Ambil URL server gateway dari titik akhir interaktif Anda](#)
- [Ambil token autentikasi untuk terhubung ke titik akhir interaktif](#)

- [Contoh: Menyebarkan buku catatan JupyterLab](#)
- [Menghapus buku catatan Jupyter yang dihosting sendiri](#)

Membuat grup keamanan

Sebelum Anda dapat membuat endpoint interaktif dan menjalankan Jupyter atau JupyterLab notebook yang dihosting sendiri, Anda harus membuat grup keamanan untuk mengontrol lalu lintas antara buku catatan Anda dan titik akhir interaktif. Untuk menggunakan konsol Amazon EC2 atau Amazon EC2 SDK untuk membuat grup keamanan, lihat langkah-langkah [dalam Membuat grup keamanan di Panduan Pengguna](#) Amazon EC2. Anda harus membuat grup keamanan di VPC tempat Anda ingin menyebarkan server notebook Anda.

Untuk mengikuti contoh dalam panduan ini, gunakan VPC yang sama dengan cluster Amazon EKS Anda. Jika Anda ingin meng-host notebook Anda di VPC yang berbeda dari VPC untuk cluster Amazon EKS Anda, Anda mungkin perlu membuat koneksi peering di antara keduanya. VPCs Untuk langkah-langkah untuk membuat koneksi peering di antara keduanya VPCs, lihat [Membuat koneksi peering VPC](#) di Panduan Memulai VPC Amazon.

Anda memerlukan ID untuk grup keamanan untuk [membuat EMR Amazon di titik akhir interaktif EKS di langkah](#) berikutnya.

Buat EMR Amazon di titik akhir interaktif EKS

Setelah Anda membuat grup keamanan untuk buku catatan Anda, gunakan langkah-langkah yang disediakan [Membuat endpoint interaktif untuk klaster virtual Anda](#) untuk membuat titik akhir interaktif. Anda harus memberikan ID grup keamanan yang Anda buat untuk buku catatan Anda [Membuat grup keamanan](#).

Masukkan ID keamanan sebagai pengganti pengaturan penggantian konfigurasi berikut: *your-notebook-security-group-id*

```
--configuration-overrides '{
  "applicationConfiguration": [
    {
      "classification": "endpoint-configuration",
      "properties": {
        "notebook-security-group-id": "your-notebook-security-group-id"
      }
    }
  ]
}
```

```

    }
  ],
  "monitoringConfiguration": {
    ...'

```

Ambil URL server gateway dari titik akhir interaktif Anda

Setelah Anda membuat endpoint interaktif, ambil URL server gateway dengan `describe-managed-endpoint` perintah di AWS CLI Anda memerlukan URL ini untuk menghubungkan buku catatan Anda ke titik akhir. URL server gateway adalah titik akhir pribadi.

```

aws emr-containers describe-managed-endpoint \
--region region \
--virtual-cluster-id virtualClusterId \
--id endpointId

```

Awalnya, titik akhir Anda ada di CREATING negara bagian. Setelah beberapa menit, ia beralih ke ACTIVE negara. Ketika titik akhir ACTIVE, siap digunakan.

Perhatikan `serverUrl` atribut yang dikembalikan `aws emr-containers describe-managed-endpoint` perintah dari endpoint aktif. Anda memerlukan URL ini untuk menghubungkan buku catatan Anda ke titik akhir saat [Anda menerapkan Jupyter atau notebook yang dihosting sendiri](#).
JupyterLab

Ambil token autentikasi untuk terhubung ke titik akhir interaktif

Untuk terhubung ke endpoint interaktif dari Jupyter atau JupyterLab notebook, Anda harus membuat token sesi dengan API. `GetManagedEndpointSessionCredentials` Token bertindak sebagai bukti otentikasi untuk terhubung ke server endpoint interaktif.

Perintah berikut dijelaskan secara lebih rinci dengan contoh output di bawah ini.

```

aws emr-containers get-managed-endpoint-session-credentials \
--endpoint-identifier endpointArn \
--virtual-cluster-identifier virtualClusterArn \
--execution-role-arn executionRoleArn \
--credential-type "TOKEN" \
--duration-in-seconds durationInSeconds \
--region region

```

endpointArn

ARN dari titik akhir Anda. Anda dapat menemukan ARN dalam hasil panggilan `describe-managed-endpoint`.

virtualClusterArn

ARN dari cluster virtual.

executionRoleArn

ARN dari peran eksekusi.

durationInSeconds

Durasi dalam detik dimana token valid. Durasi default adalah 15 menit (900), dan maksimum adalah 12 jam (43200).

region

Wilayah yang sama dengan titik akhir Anda.

Output Anda harus menyerupai contoh berikut. Catat *session-token* nilai yang akan Anda gunakan saat [menerapkan Jupyter atau notebook yang dihosting sendiri](#). JupyterLab

```
{
  "id": "credentialsId",
  "credentials": {
    "token": "session-token"
  },
  "expiresAt": "2022-07-05T17:49:38Z"
}
```

Contoh: Menyebarkan buku catatan JupyterLab

Setelah Anda menyelesaikan langkah-langkah di atas, Anda dapat mencoba prosedur contoh ini untuk menyebarkan JupyterLab notebook ke cluster Amazon EKS dengan endpoint interaktif Anda.

1. Buat namespace untuk menjalankan server notebook.
2. Buat file secara lokal, `notebook.yaml`, dengan konten berikut. Isi file dijelaskan di bawah ini.

```
apiVersion: v1
kind: Pod
metadata:
```

```

name: jupyter-notebook
namespace: namespace
spec:
  containers:
  - name: minimal-notebook
    image: jupyter/all-spark-notebook:lab-3.1.4 # open source image
    ports:
    - containerPort: 8888
    command: ["start-notebook.sh"]
    args: ["--LabApp.token='']
    env:
    - name: JUPYTER_ENABLE_LAB
      value: "yes"
    - name: KERNEL_LAUNCH_TIMEOUT
      value: "400"
    - name: JUPYTER_GATEWAY_URL
      value: "serverUrl"
    - name: JUPYTER_GATEWAY_VALIDATE_CERT
      value: "false"
    - name: JUPYTER_GATEWAY_AUTH_TOKEN
      value: "session-token"

```

Jika Anda menerapkan notebook Jupyter ke cluster khusus Fargate, beri label pada pod Jupyter dengan label seperti yang ditunjukkan pada contoh berikut: `role`

```

...
metadata:
  name: jupyter-notebook
  namespace: default
  labels:
    role: example-role-name-label
spec:
  ...

```

namespace

Namespace Kubernetes yang digunakan notebook.

serverUrl

`serverUrl` Atribut yang dikembalikan `describe-managed-endpoint` perintah [Ambil URL server gateway dari titik akhir interaktif Anda](#).

session-token

session-tokenAtribut yang dikembalikan `get-managed-endpoint-session-credentials` perintah [Ambil token autentikasi untuk terhubung ke titik akhir interaktif](#).

KERNEL_LAUNCH_TIMEOUT

Jumlah waktu dalam hitungan detik titik akhir interaktif menunggu kernel muncul. RUNNING Pastikan waktu yang cukup untuk menyelesaikan peluncuran kernel dengan mengatur batas waktu peluncuran kernel ke nilai yang sesuai (maksimum 400 detik).

KERNEL_EXTRA_SPARK_OPTS

Secara opsional, Anda dapat meneruskan konfigurasi Spark tambahan untuk kernel Spark. Tetapkan variabel lingkungan ini dengan nilai-nilai sebagai properti konfigurasi Spark seperti yang ditunjukkan pada contoh berikut:

```
- name: KERNEL_EXTRA_SPARK_OPTS
  value: "--conf spark.driver.cores=2
         --conf spark.driver.memory=2G
         --conf spark.executor.instances=2
         --conf spark.executor.cores=2
         --conf spark.executor.memory=2G
         --conf spark.dynamicAllocation.enabled=true
         --conf spark.dynamicAllocation.shuffleTracking.enabled=true
         --conf spark.dynamicAllocation.minExecutors=1
         --conf spark.dynamicAllocation.maxExecutors=5
         --conf spark.dynamicAllocation.initialExecutors=1
         "
```

3. Menerapkan spesifikasi pod ke cluster Amazon EKS Anda:

```
kubectl apply -f notebook.yaml -n namespace
```

Ini akan memulai JupyterLab notebook minimal yang terhubung ke EMR Amazon Anda di titik akhir interaktif EKS. Tunggu sampai pod itu RUNNING. Anda dapat memeriksa statusnya dengan perintah berikut:

```
kubectl get pod jupyter-notebook -n namespace
```

Ketika pod siap, `get pod` perintah mengembalikan output yang mirip dengan ini:

NAME	READY	STATUS	RESTARTS	AGE
jupyter-notebook	1/1	Running	0	46s

4. Lampirkan grup keamanan notebook ke node tempat notebook dijadwalkan.
 - a. Pertama, identifikasi node tempat jupyter-notebook pod dijadwalkan dengan describe pod perintah.

```
kubectl describe pod jupyter-notebook -n namespace
```

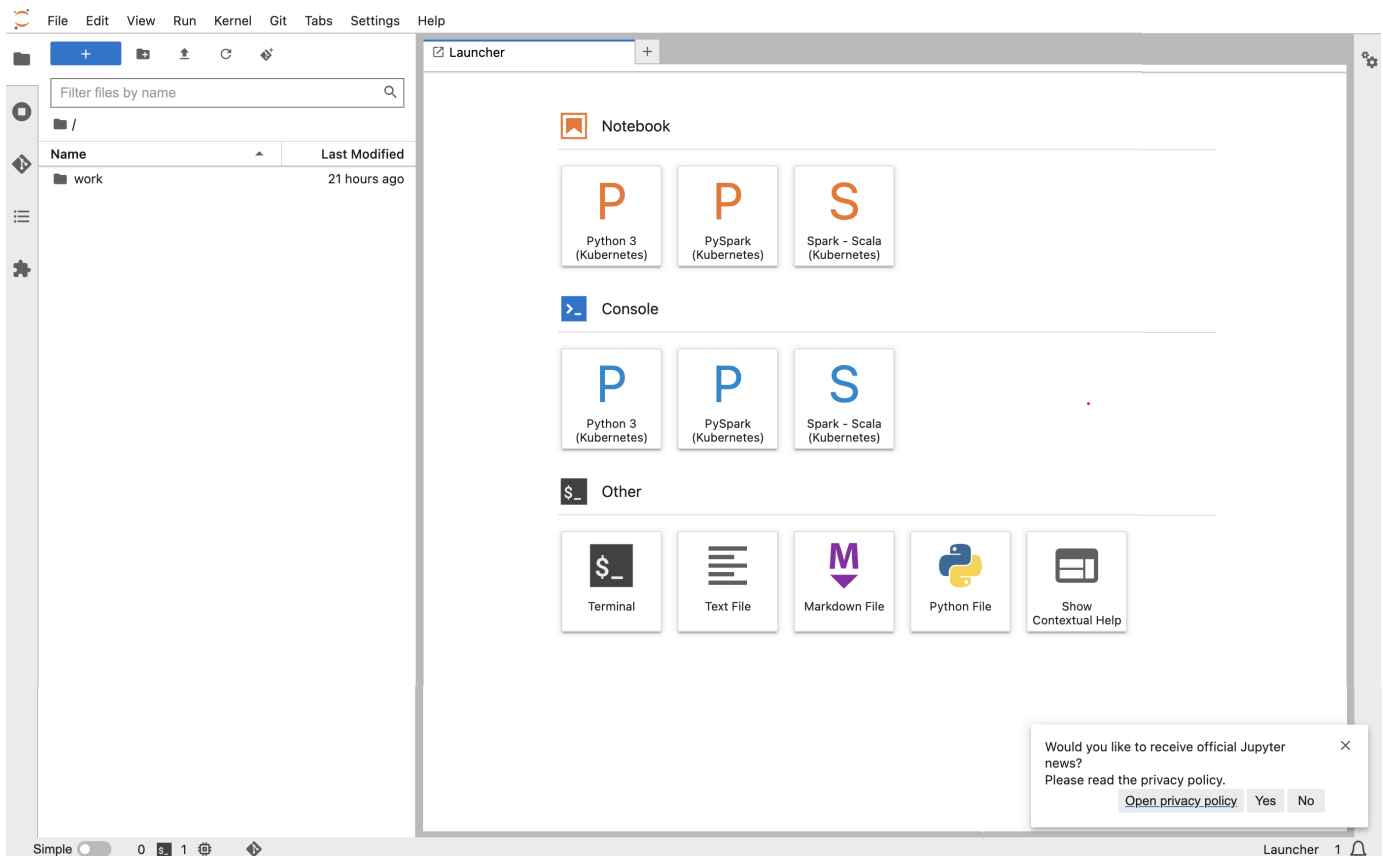
- b. Buka konsol Amazon EKS di <https://console.aws.amazon.com/eks/rumah#/cluster>.
- c. Arahkan ke tab Compute untuk cluster Amazon EKS Anda dan pilih node yang diidentifikasi oleh describe pod perintah. Pilih ID instance untuk node.
- d. Dari menu Tindakan, pilih Keamanan > Ubah grup keamanan untuk melampirkan grup keamanan yang Anda buat [Membuat grup keamanan](#).
- e. Jika Anda menerapkan pod notebook Jupyter AWS Fargate, buat a [SecurityGroupPolicy](#) to apply ke pod notebook Jupyter dengan label peran:

```
cat >my-security-group-policy.yaml <<EOF
apiVersion: vpcresources.k8s.aws/v1beta1
kind: SecurityGroupPolicy
metadata:
  name: example-security-group-policy-name
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: example-role-name-label
  securityGroups:
    groupIds:
      - your-notebook-security-group-id
EOF
```

5. Sekarang, port-forward sehingga Anda dapat mengakses antarmuka secara lokal: JupyterLab

```
kubectl port-forward jupyter-notebook 8888:8888 -n namespace
```

Setelah berjalan, navigasikan ke browser lokal Anda dan kunjungi `localhost:8888` untuk melihat JupyterLab antarmuka:



6. Dari JupyterLab, buat notebook Scala baru. Berikut adalah contoh cuplikan kode yang dapat Anda jalankan untuk memperkirakan nilai Pi:

```
import scala.math.random
import org.apache.spark.sql.SparkSession

/** Computes an approximation to pi */
val session = SparkSession
  .builder
  .appName("Spark Pi")
  .getOrCreate()

val slices = 2
// avoid overflow
val n = math.min(100000L * slices, Int.MaxValue).toInt

val count = session.sparkContext
  .parallelize(1 until n, slices)
  .map { i =>
    val x = random * 2 - 1
    val y = random * 2 - 1
```

```

    if (x*x + y*y <= 1) 1 else 0
  }.reduce(_ + _)

println(s"Pi is roughly ${4.0 * count / (n - 1)}")
session.stop()

```

```

File Edit View Run Kernel Git Tabs Settings Help
+
Filter files by name
/
Name Last Modified
work 21 hours ago
Untitled.ipynb 4 minutes ago
Untitled.ipynb
[3]: import scala.math.random
import org.apache.spark.sql.SparkSession

/** Computes an approximation to pi */
val session = SparkSession
  .builder
  .appName("Spark Pi")
  .getOrCreate()

val slices = 2
// avoid overflow
val n = math.min(100000L * slices, Int.MaxValue).toInt

val count = session.sparkContext
  .parallelize(1 until n, slices)
  .map { i =>
    val x = random * 2 - 1
    val y = random * 2 - 1
    if (x*x + y*y <= 1) 1 else 0
  }.reduce(_ + _)

println(s"Pi is roughly ${4.0 * count / (n - 1)}")
session.stop()

Pi is roughly 3.140955704778524
session = org.apache.spark.sql.SparkSession@722cd3ee
slices = 2
n = 200000
count = 157047

[3]: 157047
[ ]:

```

Menghapus buku catatan Jupyter yang dihosting sendiri

Saat Anda siap untuk menghapus buku catatan yang dihosting sendiri, Anda juga dapat menghapus titik akhir interaktif dan grup keamanan juga. Lakukan tindakan dengan urutan sebagai berikut:

1. Gunakan perintah berikut untuk menghapus jupyter-notebook pod:

```
kubectl delete pod jupyter-notebook -n namespace
```

2. Kemudian, hapus endpoint interaktif Anda dengan delete-managed-endpoint perintah. Untuk langkah-langkah menghapus titik akhir interaktif, lihat [Hapus titik akhir interaktif](#). Awalnya, titik akhir Anda akan berada di TERMINATING negara bagian. Setelah semua sumber daya dibersihkan, ia beralih ke TERMINATED negara bagian.

3. Jika Anda tidak berencana untuk menggunakan grup keamanan notebook yang Anda buat [Membuat grup keamanan](#) untuk penerapan notebook Jupyter lainnya, Anda dapat menghapusnya. Lihat [Menghapus grup keamanan](#) di Panduan Pengguna Amazon EC2 untuk informasi selengkapnya.

Mendapatkan informasi tentang titik akhir interaktif dengan perintah CLI

Topik ini mencakup operasi yang didukung pada titik akhir interaktif selain [create-managed-endpoint](#).

Ambil detail titik akhir interaktif

Setelah Anda membuat endpoint interaktif, Anda dapat mengambil detailnya menggunakan perintah `describe-managed-endpoint` AWS CLI Masukkan nilai Anda sendiri untuk *managed-endpoint-id*, *virtual-cluster-id*, dan *region*:

```
aws emr-containers describe-managed-endpoint --id managed-endpoint-id \
--virtual-cluster-id virtual-cluster-id --region region
```

Outputnya terlihat mirip dengan berikut ini, dengan titik akhir yang ditentukan, seperti ARN, ID, dan nama.

```
{
  "id": "as3ys2xxxxxxxx",
  "name": "endpoint-name",
  "arn": "arn:aws:emr-containers:us-east-1:1828xxxxxxxx:/virtualclusters/
lbhl6kwwyoxxxxxxxxxxxxxxxxx/endpoints/as3ysxxxxxxxx",
  "virtualClusterId": "lbhl6kwwyoxxxxxxxxxxxxxxxxx",
  "type": "JUPYTER_ENTERPRISE_GATEWAY",
  "state": "ACTIVE",
  "releaseLabel": "emr-6.9.0-latest",
  "executionRoleArn": "arn:aws:iam::1828xxxxxxxx:role/RoleName",
  "certificateAuthority": {
    "certificateArn": "arn:aws:acm:us-east-1:1828xxxxxxxx:certificate/zzzzzzzz-
e59b-4ed0-aaaa-bbbbbbbbbbbb",
    "certificateData": "certificate-data"
  },
  "configurationOverrides": {
```

```

    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.driver.memory": "8G"
        }
      }
    ],
    "monitoringConfiguration": {
      "persistentAppUI": "ENABLED",
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "log-group-name",
        "logStreamNamePrefix": "log-stream-name-prefix"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3-bucket-name"
      }
    }
  },
  "serverUrl": "https://internal-k8s-namespace-ingressa-aaaaaaaaaa-
zzzzzzzzzz.us-east-1.elb.amazonaws.com:18888 (https://internal-k8s-nspluto-
ingressa-51e860abbd-1620715833.us-east-1.elb.amazonaws.com:18888/)",
  "createdAt": "2022-09-19T12:37:49+00:00",
  "securityGroup": "sg-aaaaaaaaaaaaaa",
  "subnetIds": [
    "subnet-111111111111",
    "subnet-222222222222",
    "subnet-333333333333"
  ],
  "stateDetails": "Endpoint created successfully. It took 3 Minutes 15 Seconds",
  "tags": {}
}

```

Buat daftar semua titik akhir interaktif yang terkait dengan cluster virtual

Gunakan `list-managed-endpoints` AWS CLI perintah untuk mengambil daftar semua endpoint interaktif yang terkait dengan cluster virtual tertentu. Ganti `virtual-cluster-id` dengan ID cluster virtual Anda.

```
aws emr-containers list-managed-endpoints --virtual-cluster-id virtual-cluster-id
```

Output dari `list-managed-endpoint` perintah ditunjukkan di bawah ini:

```

{
  "endpoints": [{
    "id": "as3ys2xxxxxxxx",
    "name": "endpoint-name",
    "arn": "arn:aws:emr-containers:us-east-1:1828xxxxxxxx:/virtualclusters/
lbhl6kwwyoxxxxxxxxxxxxxxxxx/endpoints/as3ysxxxxxxxx",
    "virtualClusterId": "lbhl6kwwyoxxxxxxxxxxxxxxxxx",
    "type": "JUPYTER_ENTERPRISE_GATEWAY",
    "state": "ACTIVE",
    "releaseLabel": "emr-6.9.0-latest",
    "executionRoleArn": "arn:aws:iam::1828xxxxxxxx:role/RoleName",
    "certificateAuthority": {
      "certificateArn": "arn:aws:acm:us-east-1:1828xxxxxxxx:certificate/zzzzzzzz-
e59b-4ed0-aaaa-bbbbbbbbbbbb",
      "certificateData": "certificate-data"
    },
    "configurationOverrides": {
      "applicationConfiguration": [{
        "classification": "spark-defaults",
        "properties": {
          "spark.driver.memory": "8G"
        }
      }
    ],
    "monitoringConfiguration": {
      "persistentAppUI": "ENABLED",
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "log-group-name",
        "logStreamNamePrefix": "log-stream-name-prefix"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3-bucket-name"
      }
    }
  }
},
  "serverUrl": "https://internal-k8s-namespace-ingressa-aaaaaaaaa-
zzzzzzzzzz.us-east-1.elb.amazonaws.com:18888 (https://internal-k8s-nspluto-
ingressa-51e860abbd-1620715833.us-east-1.elb.amazonaws.com:18888/)",
  "createdAt": "2022-09-19T12:37:49+00:00",
  "securityGroup": "sg-aaaaaaaaaaaaa",
  "subnetIds": [
    "subnet-11111111111",
    "subnet-22222222222",
    "subnet-33333333333"
  ]
}

```

```
    ],
    "stateDetails": "Endpoint created successfully. It took 3 Minutes 15 Seconds",
    "tags": {}
  ]}
}
```

Hapus titik akhir interaktif

Untuk menghapus titik akhir interaktif yang terkait dengan EMR Amazon di kluster virtual EKS, gunakan `delete-managed-endpoint` AWS CLI perintah. Saat Anda menghapus titik akhir interaktif, Amazon EMR di EKS menghapus grup keamanan default yang dibuat untuk titik akhir tersebut.

Tentukan nilai untuk parameter berikut ke perintah:

- `--id`: Pengenal titik akhir interaktif yang ingin Anda hapus.
- `--virtual-cluster-id` — Pengidentifikasi cluster virtual yang terkait dengan titik akhir interaktif yang ingin Anda hapus. Ini adalah ID cluster virtual yang sama yang ditentukan ketika endpoint interaktif dibuat.

```
aws emr-containers delete-managed-endpoint --id managed-endpoint-id --virtual-cluster-id virtual-cluster-id
```

Perintah mengembalikan output yang mirip dengan berikut ini untuk mengonfirmasi bahwa Anda menghapus titik akhir interaktif:

```
{
  "id": "8gai4l4exxxxx",
  "virtualClusterId": "0b0qvauoy3ch1nqodxxxxxxxx"
}
```

Baca, Tulis, dan Unggah data ke Amazon S3 Express One Zone dengan Amazon EMR di EKS

Dengan Amazon EMR merilis 7.2.0 dan yang lebih tinggi, Anda dapat menggunakan Amazon EMR di EKS dengan kelas penyimpanan [Amazon S3 Express One Zone](#) untuk meningkatkan kinerja saat menjalankan pekerjaan dan beban kerja. S3 Express One Zone adalah kelas penyimpanan Amazon S3 zona tunggal berkinerja tinggi yang memberikan akses data milidetik satu digit yang konsisten untuk sebagian besar aplikasi yang sensitif terhadap latensi. Pada saat rilis, S3 Express One Zone memberikan latensi terendah dan penyimpanan objek cloud kinerja tertinggi di Amazon S3.

Prasyarat

Sebelum Anda dapat menggunakan S3 Express One Zone dengan Amazon EMR di EKS, Anda harus memiliki prasyarat berikut:

- [Selesai menyiapkan Amazon EMR di EKS](#).
- Setelah Anda mengatur Amazon EMR di EKS, [buat cluster virtual](#).

Memulai dengan S3 Express One Zone

Ikuti langkah-langkah ini untuk memulai dengan S3 Express One Zone

1. Tambahkan `CreateSession` izin ke peran eksekusi pekerjaan Anda. Ketika S3 Express One Zone awalnya melakukan tindakan seperti `GET`, `LIST`, atau `PUT` pada objek S3, kelas penyimpanan memanggil `CreateSession` atas nama Anda. Berikut ini adalah contoh cara memberikan `CreateSession` izin.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3express:*:*:bucket/DOC-EXAMPLE-BUCKET"
      ]
    }
  ]
}
```

```

    ],
    "Action": [
      "s3express:CreateSession"
    ],
    "Sid": "AllowS3EXPRESSCreatesession"
  }
]
}

```

2. Anda harus menggunakan konektor Apache Hadoop S3A untuk mengakses bucket S3 Express, jadi ubah Amazon S3 Anda untuk menggunakan skema untuk menggunakan konektor. URIs s3a Jika mereka tidak menggunakan skema, Anda dapat mengubah implementasi sistem file yang Anda gunakan untuk s3 dan skema. s3n

Untuk mengubah s3 skema, tentukan konfigurasi cluster berikut:

```

[
  {
    "Classification": "core-site",
    "Properties": {
      "fs.s3.impl": "org.apache.hadoop.fs.s3a.S3AFileSystem",
      "fs.AbstractFileSystem.s3.impl": "org.apache.hadoop.fs.s3a.S3A"
    }
  }
]

```

Untuk mengubah skema s3n, tentukan konfigurasi cluster berikut:

```

[
  {
    "Classification": "core-site",
    "Properties": {
      "fs.s3n.impl": "org.apache.hadoop.fs.s3a.S3AFileSystem",
      "fs.AbstractFileSystem.s3n.impl": "org.apache.hadoop.fs.s3a.S3A",
      "fs.s3a.endpoint.region": "us-west-2",
      "fs.s3a.change.detection.mode": "none",
      "fs.s3a.select.enabled": "false"
    }
  },
  {
    "Classification": "spark-defaults",
    "Properties": {

```

```
"spark.hadoop.fs.s3a.aws.credentials.provider":  
  "software.amazon.awssdk.auth.credentials.WebIdentityTokenFileCredentialsProvider",  
  "spark.sql.sources.fastS3PartitionDiscovery.enabled": "false"  
  }  
}  
]
```

3. Dalam konfigurasi spark-submit Anda, gunakan penyedia kredensi identitas web.

```
"spark.hadoop.fs.s3a.aws.credentials.provider=com.amazonaws.auth.WebIdentityTokenCredential
```

Pemantauan tugas

Anda dapat menggunakan Amazon CloudWatch Events untuk melacak pekerjaan yang berjalan di Amazon EMR di kluster virtual EKS. Anda dapat menggunakan peristiwa untuk melacak aktivitas dan kesehatan tugas yang Anda jalankan pada kluster virtual. Topik berikut menunjukkan cara untuk mengonfigurasi pemantauan secara efektif untuk menjaga kesehatan sumber daya Anda.

Topik

- [Pantau pekerjaan dengan Amazon CloudWatch Events](#)
- [Otomatiskan Amazon EMR di EKS dengan Acara CloudWatch](#)
- [Contoh: Mengatur aturan yang memanggil Lambda](#)
- [Pantau pod driver job dengan kebijakan coba lagi menggunakan Amazon CloudWatch Events](#)

Pantau pekerjaan dengan Amazon CloudWatch Events

Amazon EMR di EKS menyiarkan peristiwa ketika keadaan tugas berjalan berubah. Setiap peristiwa memberikan informasi, seperti tanggal dan waktu ketika peristiwa terjadi, bersama dengan detail lebih lanjut tentang peristiwa, seperti ID kluster virtual dan ID tugas berjalan yang terpengaruh.

Anda dapat menggunakan peristiwa untuk melacak aktivitas dan kesehatan tugas yang Anda jalankan pada kluster virtual. Anda juga dapat menggunakan Amazon CloudWatch Events untuk menentukan tindakan yang akan diambil saat menjalankan pekerjaan menghasilkan peristiwa yang cocok dengan pola yang Anda tentukan. Peristiwa berguna untuk memantau kejadian tertentu selama siklus hidup dari tugas berjalan. Misalnya, Anda dapat memantau saat status tugas berjalan berubah dari `submitted` ke `running`. Untuk informasi selengkapnya tentang CloudWatch Acara, lihat [Panduan EventBridge Pengguna Amazon](#).

Tabel berikut mencantumkan peristiwa Amazon EMR di EKS, bersama dengan status atau perubahan status yang ditunjukkan peristiwa, tingkat kepelikan peristiwa, dan pesan peristiwa. Setiap peristiwa direpresentasikan sebagai objek JSON yang dikirim secara otomatis ke alur kejadian. Objek JSON mencakup detail lebih lanjut tentang peristiwa tersebut. Objek JSON sangat penting ketika Anda mengatur aturan untuk pemrosesan acara menggunakan CloudWatch Acara karena aturan berusaha untuk mencocokkan pola dalam objek JSON. Untuk informasi selengkapnya, lihat [pola EventBridge acara Amazon](#) dan EMR Amazon di Acara EKS di [EventBridge Panduan Pengguna Amazon](#).

Peristiwa perubahan status tugas berjalan

Status	Kepelikan	Pesan
DIKIRIMKAN	INFO	Job Run <i>JobRunId</i> (<i>JobRunName</i>) berhasil dikirimkan ke cluster virtual <i>VirtualClusterId</i> di <i>Time</i> UTC.
BERJALAN	INFO	Job Run <i>JobRunId</i> (<i>JobRunName</i>) di klaster virtual <i>VirtualClusterId</i> mulai berjalan di <i>Time</i> .
DISELESAIKAN	INFO	Job Run <i>jobRunId</i> (<i>JobRunName</i>) di klaster virtual <i>VirtualClusterId</i> selesai di <i>Time</i> . Job Run mulai berjalan <i>Time</i> dan membutuhkan waktu <i>Num</i> beberapa menit untuk menyelesaikannya.
DIBATALKAN	PERINGATAN	Permintaan pembatalan telah berhasil untuk Job Run <i>JobRunId</i> (<i>JobRunName</i>) di klaster virtual <i>VirtualClusterId</i> di <i>Time</i> dan Job Run sekarang dibatalkan.
FAILED	ERROR	Job Run <i>JobRunId</i> (<i>JobRunName</i>) di klaster virtual <i>VirtualClusterId</i> gagal di <i>Time</i> .

Otomatiskan Amazon EMR di EKS dengan Acara CloudWatch

Anda dapat menggunakan Amazon CloudWatch Events untuk mengotomatiskan AWS layanan Anda guna merespons peristiwa sistem seperti masalah ketersediaan aplikasi atau perubahan sumber daya. Acara dari AWS layanan dikirim ke CloudWatch Acara dalam waktu dekat. Anda dapat menuliskan aturan sederhana untuk menunjukkan peristiwa mana yang sesuai kepentingan Anda, dan tindakan otomatis yang diambil ketika suatu peristiwa sesuai dengan suatu aturan. Tindakan yang dapat dipicu secara otomatis meliputi hal berikut:

- Memanggil fungsi AWS Lambda
- Meminta Perintah Amazon EC2 Run

- Mengirim peristiwa ke Amazon Kinesis Data Streams
- Mengaktifkan mesin AWS Step Functions negara
- Memberitahu topik Amazon Simple Notification Service (SNS) atau antrian Amazon Simple Queue Service (SQS)

Beberapa contoh penggunaan CloudWatch Acara dengan Amazon EMR di EKS meliputi yang berikut:

- Mengaktifkan fungsi Lambda ketika tugas berjalan berhasil
- Memberi tahu topik Amazon SNS saat pekerjaan berjalan gagal

CloudWatch Acara untuk "detail-type:" "EMR Job Run State Change" dihasilkan oleh Amazon EMR di EKS untuk SUBMITTED,, RUNNINGCANCELLED, FAILED dan perubahan COMPLETED status.

Contoh: Mengatur aturan yang memanggil Lambda


Gunakan langkah-langkah berikut untuk menyiapkan aturan CloudWatch Acara yang memanggil Lambda ketika ada acara "EMR Job Run State Change".

```
aws events put-rule \  
--name cwe-test \  
--event-pattern '{"detail-type": ["EMR Job Run State Change"]}'
```

Tambahkan fungsi Lambda yang Anda miliki sebagai target baru dan berikan izin CloudWatch Acara untuk menjalankan fungsi Lambda sebagai berikut. Ganti **123456789012** dengan ID akun Anda.

```
aws events put-targets \  
--rule cwe-test \  
--targets Id=1,Arn=arn:aws:lambda:us-east-1:123456789012:function:MyFunction
```

```
aws lambda add-permission \  
--function-name MyFunction \  
--statement-id MyId \  
--action 'lambda:InvokeFunction' \  
--principal events.amazonaws.com
```

 Note

Anda tidak dapat menulis program yang tergantung pada urutan keberadaan atau notifikasi peristiwa, karena program tersebut mungkin tidak berurutan atau hilang. Kejadian dipancarkan atas dasar upaya terbaik.

Pantau pod driver job dengan kebijakan coba lagi menggunakan Amazon CloudWatch Events

Dengan menggunakan CloudWatch peristiwa, Anda dapat memantau pod driver yang telah dibuat dalam pekerjaan yang memiliki kebijakan coba ulang. Untuk informasi selengkapnya, lihat [Memantau pekerjaan dengan kebijakan coba lagi](#) dalam panduan ini.

Mengelola klaster virtual

Sebuah klaster virtual adalah namespace Kubernetes tempat Amazon EMR terdaftar. Anda dapat menciptakan, menggambarkan, membuat daftar, dan menghapus klaster virtual. Mereka tidak mengonsumsi sumber daya tambahan apa pun dalam sistem Anda. Sebuah klaster virtual tunggal memetakan ke satu namespace Kubernetes. Mengingat hubungan ini, Anda dapat memodelkan klaster virtual dengan cara yang sama Anda memodelkan namespace Kubernetes untuk memenuhi persyaratan Anda. Lihat kemungkinan kasus penggunaan di dokumentasi [Gambaran Umum Konsep Kubernetes](#).

Untuk mendaftar Amazon EMR dengan namespace Kubernetes pada klaster Amazon EKS, Anda perlu nama klaster EKS dan namespace yang telah diatur untuk menjalankan beban kerja Anda. Klaster terdaftar tersebut di Amazon EMR disebut klaster virtual karena mereka tidak mengelola komputasi atau penyimpanan fisik tetapi menunjuk ke namespace Kubernetes di mana beban kerja Anda dijadwalkan.

Note

Sebelum membuat klaster virtual, Anda harus terlebih dahulu menyelesaikan langkah 1-8 di [Menyiapkan Amazon EMR di EKS](#).

Topik

- [Membuat klaster virtual](#)
- [Daftar klaster virtual](#)
- [Gambarkan klaster virtual](#)
- [Menghapus klaster virtual](#)
- [Status klaster virtual](#)

Membuat klaster virtual

Jalankan perintah berikut untuk membuat klaster virtual dengan mendaftarkan Amazon EMR dengan namespace pada klaster EKS. Ganti *virtual_cluster_name* dengan nama yang Anda berikan untuk cluster virtual Anda. Ganti *eks_cluster_name* dengan nama cluster EKS. Ganti *namespace_name* dengan namespace yang ingin Anda daftarkan Amazon EMR.

```
aws emr-containers create-virtual-cluster \  
--name virtual_cluster_name \  
--container-provider '{  
  "id": "eks_cluster_name",  
  "type": "EKS",  
  "info": {  
    "eksInfo": {  
      "namespace": "namespace_name"  
    }  
  }  
'
```

Atau, Anda dapat membuat file JSON yang mencakup parameter yang diperlukan untuk kluster virtual, seperti yang ditunjukkan contoh berikut.

```
{  
  "name": "virtual_cluster_name",  
  "containerProvider": {  
    "type": "EKS",  
    "id": "eks_cluster_name",  
    "info": {  
      "eksInfo": {  
        "namespace": "namespace_name"  
      }  
    }  
  }  
}
```

Kemudian jalankan perintah `create-virtual-cluster` berikut dengan jalur ke file JSON.

```
aws emr-containers create-virtual-cluster \  
--cli-input-json file:///./create-virtual-cluster-request.json
```

Note

Untuk memvalidasi keberhasilan pembuatan kluster virtual, lihat status kluster virtual dengan menjalankan perintah `list-virtual-clusters` atau dengan masuk ke halaman Kluster virtual di konsol Amazon EMR.

Daftar klaster virtual

Jalankan perintah berikut untuk menampilkan status klaster virtual.

```
aws emr-containers list-virtual-clusters
```

Gambarkan klaster virtual

Jalankan perintah berikut untuk mendapatkan detail lebih lanjut tentang klaster virtual, seperti namespace, status, dan tanggal terdaftar. Ganti **123456** dengan ID cluster virtual Anda.

```
aws emr-containers describe-virtual-cluster --id 123456
```

Menghapus klaster virtual

Jalankan perintah berikut untuk menghapus klaster virtual. Ganti **123456** dengan ID cluster virtual Anda.

```
aws emr-containers delete-virtual-cluster --id 123456
```

Status klaster virtual

Tabel berikut menjelaskan empat kemungkinan status klaster virtual.

State	Deskripsi
RUNNING	Klaster virtual berada dalam status RUNNING.
TERMINATING	Penghentian klaster virtual yang diminta sedang berlangsung.
TERMINATED	Penghentian yang diminta selesai.
ARRESTED	Penghentian yang diminta gagal karena izin yang tidak mencukupi.

Tutorial untuk Amazon EMR di EKS

Bagian ini menjelaskan kasus penggunaan umum ketika Anda bekerja dengan Amazon EMR pada aplikasi EKS. Setiap aplikasi khusus dan dapat mengambil langkah-langkah unik untuk mengkonfigurasi. Topik-topik ini memberikan instruksi untuk menggunakan setiap aplikasi.

Topik

- [Menggunakan Delta Lake dengan Amazon EMR di EKS](#)
- [Menggunakan Apache Iceberg dengan Amazon EMR di EKS](#)
- [Menggunakan PyFlink](#)
- [Menggunakan AWS Glue dengan Flink](#)
- [Menggunakan Apache Hudi dengan Apache Flink](#)
- [Menggunakan RAPIDS Accelerator untuk Apache Spark dengan Amazon EMR di EKS](#)
- [Menggunakan integrasi Amazon Redshift untuk Apache Spark di Amazon EMR di EKS](#)
- [Menggunakan Volcano sebagai penjadwal khusus untuk Apache Spark di Amazon EMR di EKS](#)
- [Menggunakan YuniKorn sebagai penjadwal khusus untuk Apache Spark di Amazon EMR di EKS](#)

Menggunakan Delta Lake dengan Amazon EMR di EKS

Delta Lake adalah kerangka penyimpanan sumber terbuka untuk membangun arsitektur Lakehouse. Berikut ini menunjukkan cara mengaturnya untuk digunakan.

Untuk menggunakan [Delta Lake](#) dengan Amazon EMR pada aplikasi EKS

1. Saat Anda memulai pekerjaan untuk mengirimkan pekerjaan Spark dalam konfigurasi aplikasi, sertakan file JAR Delta Lake:

```
--job-driver '{"sparkSubmitJobDriver" : {  
  "sparkSubmitParameters" : "--jars local:///usr/share/aws/delta/lib/delta-  
core.jar,local:///usr/share/aws/delta/lib/delta-storage.jar,local:///usr/share/aws/  
delta/lib/delta-storage-s3-dynamodb.jar"}}'
```

Note

Amazon EMR merilis 7.0.0 dan yang lebih tinggi menggunakan Delta Lake 3.0, yang berganti nama menjadi `delta-core.jar` dan `delta-spark.jar`. Jika Anda menggunakan Amazon EMR rilis 7.0.0 atau lebih tinggi, pastikan untuk menggunakan nama file yang benar, seperti dalam contoh berikut:

```
--jars local:///usr/share/aws/delta/lib/delta-spark.jar
```

2. Sertakan konfigurasi tambahan Delta Lake dan gunakan AWS Glue Data Catalog sebagai metastore Anda.

```
--configuration-overrides '{
  "applicationConfiguration": [
    {
      "classification" : "spark-defaults",
      "properties" : {
        "spark.sql.extensions" : "io.delta.sql.DeltaSparkSessionExtension",
        "spark.sql.catalog.spark_catalog":"org.apache.spark.sql.delta.catalog.DeltaCatalog",
        "spark.hadoop.hive.metastore.client.factory.class":"com.amazonaws.glue.catalog.metastore.AW
      }
    }
  ]
}'
```

Menggunakan Apache Iceberg dengan Amazon EMR di EKS

JAR runtime untuk Iceberg berisi kelas Iceberg yang diperlukan untuk dukungan runtime Spark.

Prosedur berikut menunjukkan cara memulai pekerjaan dengan menggunakan runtime Iceberg spark.

Untuk menggunakan Apache Iceberg dengan Amazon EMR pada aplikasi EKS

1. Saat Anda memulai pekerjaan untuk mengirimkan pekerjaan Spark dalam konfigurasi aplikasi, sertakan file JAR runtime Iceberg spark:

```
--job-driver '{"sparkSubmitJobDriver" : {"sparkSubmitParameters" : "--jars
local:///usr/share/aws/iceberg/lib/iceberg-spark3-runtime.jar"}}'
```

2. Sertakan konfigurasi tambahan Iceberg:

```
--configuration-overrides '{
  "applicationConfiguration": [
    "classification" : "spark-defaults",
    "properties" : {
      "spark.sql.catalog.dev.warehouse" : "s3://amzn-s3-demo-bucket/EXAMPLE-
PREFIX/ ",
      "spark.sql.extensions ":"
org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions ",
      "spark.sql.catalog.dev" : "org.apache.iceberg.spark.SparkCatalog",
      "spark.sql.catalog.dev.catalog-impl" :
"org.apache.iceberg.aws.glue.GlueCatalog",
      "spark.sql.catalog.dev.io-impl": "org.apache.iceberg.aws.s3.S3FileIO"
    }
  ]
}'
```

[Untuk mempelajari lebih lanjut tentang EMR versi rilis Apache Iceberg, lihat Riwayat rilis Iceberg.](#)

Konfigurasi sesi percikan untuk integrasi katalog

Konfigurasi sesi percikan untuk integrasi katalog Iceberg AWS Glue

Contoh ini menunjukkan cara mengintegrasikan Iceberg dengan: Perayap AWS Glue

```
spark-sql \
--conf spark.sql.catalog.rms = org.apache.iceberg.spark.SparkCatalog \
--conf spark.sql.catalog.rms.type = glue \
--conf spark.sql.catalog.rms.glue.id = glue RMS catalog ID \
--conf spark.sql.catalog.rms.glue.account-id = AWS account ID \

--conf spark.sql.extensions=
org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions
```

Berikut ini menunjukkan contoh query:

```
SELECT * FROM rms.rmsdb.table1
```

Konfigurasi sesi percikan untuk integrasi katalog Iceberg REST AWS Glue

Contoh ini menunjukkan cara mengintegrasikan Iceberg REST dengan: Perayap AWS Glue

```
spark-sql \  
  --conf spark.sql.catalog.rms = org.apache.iceberg.spark.SparkCatalog \  
  --conf spark.sql.catalog.rms.type = rest \  
  --conf spark.sql.catalog.rms.warehouse = glue RMS catalog ID \  
  --conf spark.sql.catalog.rms.uri = glue endpoint URI/iceberg \  
  --conf spark.sql.catalog.rms.rest.sigv4-enabled = true \  
  --conf spark.sql.catalog.rms.rest.signing-name = glue \  
  
  --conf spark.sql.extensions=  
    org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions
```

Berikut ini menunjukkan contoh query:

```
SELECT * FROM rms.rmsdb.table1
```

Konfigurasi ini hanya berfungsi untuk Redshift Managed Storage. FGAC untuk Amazon S3 tidak didukung.

Menggunakan PyFlink

Amazon EMR di EKS merilis 6.15.0 dan dukungan yang lebih tinggi. PyFlink Jika Anda sudah memiliki PyFlink skrip, Anda dapat melakukan salah satu hal berikut:

- Buat gambar kustom dengan PyFlink skrip Anda disertakan.
- Unggah skrip Anda ke lokasi Amazon S3

Jika Anda belum memiliki skrip, Anda dapat menggunakan contoh berikut untuk meluncurkan PyFlink pekerjaan. Contoh ini mengambil skrip dari S3. Jika Anda menggunakan gambar kustom dengan skrip yang sudah disertakan dalam gambar, Anda harus memperbarui jalur skrip ke lokasi tempat Anda menyimpan skrip Anda. Jika skrip berada di lokasi S3, Amazon EMR di EKS akan mengambil skrip dan menempatkannya di bawah `/opt/flink/usrlib/` direktori dalam wadah Flink.

```
apiVersion: flink.apache.org/v1beta1  
kind: FlinkDeployment  
metadata:  
  name: python-example  
spec:  
  flinkVersion: v1_17  
  flinkConfiguration:
```

```

taskmanager.numberOfTaskSlots: "1"
executionRoleArn: job-execution-role
emrReleaseLabel: "emr-6.15.0-flink-latest"
jobManager:
  highAvailabilityEnabled: false
  replicas: 1
  resource:
    memory: "2048m"
    cpu: 1
taskManager:
  resource:
    memory: "2048m"
    cpu: 1
job:
  jarURI: s3://S3 bucket with your script/pyflink-script.py
  entryClass: "org.apache.flink.client.python.PythonDriver"
  args: ["-py", "/opt/flink/usrlib/pyflink-script.py"]
  parallelism: 1
  upgradeMode: stateless

```

Menggunakan AWS Glue dengan Flink

Amazon EMR di EKS dengan Apache Flink merilis 6.15.0 dan dukungan yang lebih tinggi menggunakan Katalog Data AWS Glue sebagai penyimpanan metadata untuk streaming dan alur kerja SQL batch.

Anda harus terlebih dahulu membuat database AWS Glue bernama default yang berfungsi sebagai Katalog SQL Flink Anda. Katalog Flink ini menyimpan metadata seperti database, tabel, parisi, tampilan, fungsi, dan informasi lain yang diperlukan untuk mengakses data di sistem eksternal lainnya.

```

aws glue create-database \
  --database-input "{\"Name\":\"default\"}"

```

Untuk mengaktifkan dukungan AWS Glue, gunakan FlinkDeployment spesifikasi. Contoh spesifikasi ini menggunakan skrip Python untuk dengan cepat mengeluarkan beberapa pernyataan SQL Flink untuk berinteraksi dengan katalog Glue. AWS

```

apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment

```

```

metadata:
  name: python-example
spec:
  flinkVersion: v1_17
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "1"
    aws.glue.enabled: "true"
  executionRoleArn: job-execution-role-arn;
  emrReleaseLabel: "emr-6.15.0-flink-latest"
  jobManager:
    highAvailabilityEnabled: false
    replicas: 1
    resource:
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
  job:
    jarURI: s3://<S3_bucket_with_your_script/pyflink-glue-script.py
    entryClass: "org.apache.flink.client.python.PythonDriver"
    args: ["-py", "/opt/flink/usrlib/pyflink-glue-script.py"]
    parallelism: 1
    upgradeMode: stateless

```

Berikut ini adalah contoh dari apa PyFlink script Anda mungkin terlihat seperti.

```

import logging
import sys
from pyflink.datastream import StreamExecutionEnvironment
from pyflink.table import StreamTableEnvironment

def glue_demo():
    env = StreamExecutionEnvironment.get_execution_environment()
    t_env = StreamTableEnvironment.create(stream_execution_environment=env)
    t_env.execute_sql("""
        CREATE CATALOG glue_catalog WITH (
            'type' = 'hive',
            'default-database' = 'default',
            'hive-conf-dir' = '/glue/confs/hive/conf',
            'hadoop-conf-dir' = '/glue/confs/hadoop/conf'
        )
    """)

```

```

        """)
t_env.execute_sql("""
    USE CATALOG glue_catalog;
    """)
t_env.execute_sql("""
    DROP DATABASE IF EXISTS eks_flink_db CASCADE;
    """)
t_env.execute_sql("""
    CREATE DATABASE IF NOT EXISTS eks_flink_db WITH ('hive.database.location-
uri'= 's3a://S3-bucket-to-store-metadata/flink/flink-glue-for-hive/warehouse/');
    """)
t_env.execute_sql("""
    USE eks_flink_db;
    """)
t_env.execute_sql("""
    CREATE TABLE IF NOT EXISTS eksglueorders (
        order_number BIGINT,
        price          DECIMAL(32,2),
        buyer          ROW first_name STRING, last_name STRING,
        order_time     TIMESTAMP(3)
    ) WITH (
        'connector' = 'datagen'
    );
    """)
t_env.execute_sql("""
    CREATE TABLE IF NOT EXISTS eksdestglueorders (
        order_number BIGINT,
        price          DECIMAL(32,2),
        buyer          ROW first_name STRING, last_name STRING,
        order_time     TIMESTAMP(3)
    ) WITH (
        'connector' = 'filesystem',
        'path' = 's3://S3-bucket-to-store-metadata/flink/flink-glue-for-hive/
warehouse/eksdestglueorders',
        'format' = 'json'
    );
    """)
t_env.execute_sql("""
    CREATE TABLE IF NOT EXISTS print_table (
        order_number BIGINT,
        price          DECIMAL(32,2),
        buyer          ROW first_name STRING, last_name STRING,
        order_time     TIMESTAMP(3)
    ) WITH (

```

```

        'connector' = 'print'
    );
        """
t_env.execute_sql("""
EXECUTE STATEMENT SET
BEGIN
INSERT INTO eksdestglueorders SELECT * FROM eksglueorders LIMIT 10;
INSERT INTO print_table SELECT * FROM eksdestglueorders;
END;
""")

if __name__ == '__main__':
    logging.basicConfig(stream=sys.stdout, level=logging.INFO, format="%(message)s")
    glue_demo()

```

Menggunakan Apache Hudi dengan Apache Flink

Apache Hudi adalah kerangka kerja manajemen data sumber terbuka dengan operasi tingkat rekaman seperti menyisipkan, memperbaiki, meningkatkan, dan menghapus yang dapat Anda gunakan untuk menyederhanakan manajemen data dan pengembangan pipa data. Ketika dikombinasikan dengan manajemen data yang efisien di Amazon S3, Hudi memungkinkan Anda menelan dan memperbaiki data secara real time. Hudi mempertahankan metadata dari semua operasi yang Anda jalankan pada dataset, sehingga semua tindakan tetap atom dan konsisten.

Apache Hudi tersedia di Amazon EMR di EKS dengan Apache Flink dengan Amazon EMR rilis 7.2.0 dan lebih tinggi. Lihat langkah-langkah berikut untuk mempelajari cara memulai dan mengirimkan pekerjaan Apache Hudi.

Kirim pekerjaan Apache Hudi

Lihat langkah-langkah berikut untuk mempelajari cara mengirimkan pekerjaan Apache Hudi.

1. Buat database AWS Glue bernama default.

```
aws glue create-database --database-input "{\"Name\":\"default\"}"
```

2. Ikuti [Contoh SQL Operator Flink Kubernetes](#) untuk membangun file. `flink-sql-runner.jar`
3. Buat skrip Hudi SQL seperti berikut ini.

```
CREATE CATALOG hudi_glue_catalog WITH (
```

```
'type' = 'hudi',
'mode' = 'hms',
'table.external' = 'true',
'default-database' = 'default',
'hive.conf.dir' = '/glue/confs/hive/conf/',
'catalog.path' = 's3://<hudi-example-bucket>/FLINK_HUDI/warehouse/'
);

USE CATALOG hudi_glue_catalog;
CREATE DATABASE IF NOT EXISTS hudi_db;
use hudi_db;

CREATE TABLE IF NOT EXISTS hudi-flink-example-table(
    uuid VARCHAR(20),
    name VARCHAR(10),
    age INT,
    ts TIMESTAMP(3),
    `partition` VARCHAR(20)
)
PARTITIONED BY (`partition`)
WITH (
    'connector' = 'hudi',
    'path' = 's3://<hudi-example-bucket>/hudi-flink-example-table',
    'hive_sync.enable' = 'true',
    'hive_sync.mode' = 'glue',
    'hive_sync.table' = 'hudi-flink-example-table',
    'hive_sync.db' = 'hudi_db',
    'compaction.delta_commits' = '1',
    'hive_sync.partition_fields' = 'partition',
    'hive_sync.partition_extractor_class' =
'org.apache.hudi.hive.MultiPartKeyValueExtractor',
    'table.type' = 'COPY_ON_WRITE'
);

EXECUTE STATEMENT SET
BEGIN

INSERT INTO hudi-flink-example-table VALUES
    ('id1', 'Alex', 23, TIMESTAMP '1970-01-01 00:00:01', 'par1'),
    ('id2', 'Stephen', 33, TIMESTAMP '1970-01-01 00:00:02', 'par1'),
    ('id3', 'Julian', 53, TIMESTAMP '1970-01-01 00:00:03', 'par2'),
    ('id4', 'Fabian', 31, TIMESTAMP '1970-01-01 00:00:04', 'par2'),
    ('id5', 'Sophia', 18, TIMESTAMP '1970-01-01 00:00:05', 'par3'),
    ('id6', 'Emma', 20, TIMESTAMP '1970-01-01 00:00:06', 'par3'),
```

```

('id7', 'Bob', 44, TIMESTAMP '1970-01-01 00:00:07', 'par4'),
('id8', 'Han', 56, TIMESTAMP '1970-01-01 00:00:08', 'par4');

```

```
END;
```

4. Unggah skrip Hudi SQL Anda dan `flink-sql-runner.jar` file ke lokasi S3.
5. Di file `FlinkDeployments` YAMAL Anda, setel `hudi.enabled` ke `true`.

```

spec:
  flinkConfiguration:
    hudi.enabled: "true"

```

6. Buat file YAMAL untuk menjalankan konfigurasi Anda. File contoh ini diberi nama `hudi-write.yaml`.

```

apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: hudi-write-example
spec:
  flinkVersion: v1_18
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
    hudi.enabled: "true"
  executionRoleArn: "<JobExecutionRole>"
  emrReleaseLabel: "emr-7.13.0-flink-latest"
  jobManager:
    highAvailabilityEnabled: false
    replicas: 1
    resource:
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
  job:
    jarURI: local:///opt/flink/usrlib/flink-sql-runner.jar
    args: ["/opt/flink/scripts/hudi-write.sql"]
    parallelism: 1
    upgradeMode: stateless
  podTemplate:
    spec:

```

```
initContainers:
  - name: flink-sql-script-download
    args:
      - s3
      - cp
      - s3://<s3_location>/hudi-write.sql
      - /flink-scripts
    image: amazon/aws-cli:latest
    imagePullPolicy: Always
    resources: {}
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
      - mountPath: /flink-scripts
        name: flink-scripts
  - name: flink-sql-runner-download
    args:
      - s3
      - cp
      - s3://<s3_location>/flink-sql-runner.jar
      - /flink-artifacts
    image: amazon/aws-cli:latest
    imagePullPolicy: Always
    resources: {}
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
      - mountPath: /flink-artifacts
        name: flink-artifact
containers:
  - name: flink-main-container
    volumeMounts:
      - mountPath: /opt/flink/scripts
        name: flink-scripts
      - mountPath: /opt/flink/usrlib
        name: flink-artifact
volumes:
  - emptyDir: {}
    name: flink-scripts
  - emptyDir: {}
    name: flink-artifact
```

7. Kirim pekerjaan Flink Hudi ke operator [Flink Kubernetes](#).

```
kubectl apply -f hudi-write.yaml
```

Menggunakan RAPIDS Accelerator untuk Apache Spark dengan Amazon EMR di EKS

Dengan Amazon EMR di EKS, Anda dapat menjalankan pekerjaan untuk Nvidia RAPIDS Accelerator untuk Apache Spark. Tutorial ini mencakup cara menjalankan pekerjaan Spark menggunakan RAPIDS pada jenis instans unit pemrosesan grafis (GPU) EC2. Tutorial menggunakan versi berikut:

- Amazon EMR pada versi rilis EKS 6.9.0 dan yang lebih baru
- Apache Spark 3.x

Anda dapat mempercepat Spark dengan jenis instans GPU Amazon EC2 dengan menggunakan plugin [Nvidia RAPIDS Accelerator](#) for Apache Spark. Ketika Anda menggunakan teknologi ini bersama-sama, Anda mempercepat jalur ilmu data Anda tanpa harus membuat perubahan kode apa pun. Ini mengurangi waktu berjalan yang diperlukan untuk pemrosesan data dan pelatihan model. Dengan menyelesaikan lebih banyak dalam waktu yang lebih singkat, Anda menghabiskan lebih sedikit untuk biaya infrastruktur.

Sebelum Anda mulai, pastikan Anda memiliki sumber daya berikut.

- Amazon EMR pada klaster virtual EKS
- Cluster Amazon EKS dengan grup node berkemampuan GPU

Cluster virtual Amazon EKS adalah pegangan terdaftar ke namespace Kubernetes di cluster Amazon EKS, dan dikelola oleh Amazon EMR di EKS. Pegangan ini memungkinkan Amazon EMR untuk menggunakan namespace Kubernetes sebagai tujuan untuk menjalankan pekerjaan. Untuk informasi selengkapnya tentang cara menyiapkan klaster virtual, lihat [Menyiapkan Amazon EMR di EKS](#) di panduan ini.

Anda harus mengonfigurasi klaster virtual Amazon EKS dengan grup node yang memiliki instance GPU. Anda harus mengkonfigurasi node dengan plugin perangkat Nvidia. Lihat [grup node dikelola](#) untuk mempelajari lebih lanjut.

Untuk mengonfigurasi kluster Amazon EKS Anda untuk menambahkan grup node berkemampuan GPU, lakukan prosedur berikut:

Untuk menambahkan grup node yang diaktifkan GPU

1. Buat grup node berkemampuan GPU dengan perintah [create-nodegroup](#) berikut. Pastikan untuk mengganti parameter yang benar untuk cluster Amazon EKS Anda. Gunakan jenis instans yang mendukung Spark RAPIDS, seperti P4, P3, G5 atau G4dn.

```
aws eks create-nodegroup \  
  --cluster-name EKS_CLUSTER_NAME \  
  --nodegroup-name NODEGROUP_NAME \  
  --scaling-config minSize=0,maxSize=5,desiredSize=2 CHOOSE_APPROPRIATELY \  
  --ami-type AL2_x86_64_GPU \  
  --node-role NODE_ROLE \  
  --subnets SUBNETS_SPACE_DELIMITED \  
  --remote-access ec2SshKey= SSH_KEY \  
  --instance-types GPU_INSTANCE_TYPE \  
  --disk-size DISK_SIZE \  
  --region AWS_REGION
```

2. Instal plugin perangkat Nvidia di cluster Anda untuk memancarkan jumlah GPUs pada setiap node cluster Anda dan untuk menjalankan kontainer berkemampuan GPU di cluster Anda. Jalankan kode berikut untuk menginstal plugin:

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/v0.9.0/  
nvidia-device-plugin.yml
```

3. Untuk memvalidasi berapa banyak GPUs yang tersedia di setiap node cluster Anda, jalankan perintah berikut:

```
kubectl get nodes "-o=custom-  
columns=NAME:.metadata.name,GPU:.status.allocatable.nvidia\.com/gpu"
```

Untuk menjalankan pekerjaan Spark RAPIDS

1. Kirimkan pekerjaan Spark RAPIDS ke EMR Amazon Anda di kluster EKS. Kode berikut menunjukkan contoh perintah untuk memulai pekerjaan. Pertama kali Anda menjalankan pekerjaan, mungkin perlu beberapa menit untuk mengunduh gambar dan menyimpannya di node.

```
aws emr-containers start-job-run \
--virtual-cluster-id VIRTUAL_CLUSTER_ID \
--execution-role-arn JOB_EXECUTION_ROLE \
--release-label emr-6.9.0-spark-rapids-latest \
--job-driver '{"sparkSubmitJobDriver": {"entryPoint": "local:///usr/lib/
spark/examples/jars/spark-examples.jar","entryPointArguments": ["10000"],
"sparkSubmitParameters":"--class org.apache.spark.examples.SparkPi "}}' \
---configuration-overrides '{"applicationConfiguration": [{"classification":
"spark-defaults","properties": {"spark.executor.instances":
"2","spark.executor.memory": "2G"}}],"monitoringConfiguration":
{"cloudWatchMonitoringConfiguration": {"logGroupName": "LOG_GROUP
_NAME"}, "s3MonitoringConfiguration": {"logUri": "LOG_GROUP_STREAM"}}}'
```

2. Untuk memvalidasi bahwa Akselerator Spark RAPIDS diaktifkan, periksa log driver Spark. Log ini disimpan baik di dalam CloudWatch atau di lokasi S3 yang Anda tentukan saat Anda menjalankan `start-job-run` perintah. Contoh berikut umumnya menunjukkan seperti apa garis log:

```
22/11/15 00:12:44 INFO RapidsPluginUtils: RAPIDS Accelerator build:
{version=22.08.0-amzn-0, user=release, url=, date=2022-11-03T03:32:45Z, revision=,
cudf_version=22.08.0, branch=}
22/11/15 00:12:44 INFO RapidsPluginUtils: RAPIDS Accelerator JNI build:
{version=22.08.0, user=, url=https://github.com/NVIDIA/spark-rapids-jni.git,
date=2022-08-18T04:14:34Z, revision=a1b23cd_sample, branch=HEAD}
22/11/15 00:12:44 INFO RapidsPluginUtils: cudf build: {version=22.08.0,
user=, url=https://github.com/rapidsai/cudf.git, date=2022-08-18T04:14:34Z,
revision=a1b23ce_sample, branch=HEAD}
22/11/15 00:12:44 WARN RapidsPluginUtils: RAPIDS Accelerator 22.08.0-amzn-0 using
cudf 22.08.0.
22/11/15 00:12:44 WARN RapidsPluginUtils:
spark.rapids.sql.multiThreadedRead.numThreads is set to 20.
22/11/15 00:12:44 WARN RapidsPluginUtils: RAPIDS Accelerator is enabled, to disable
GPU support set `spark.rapids.sql.enabled` to false.
22/11/15 00:12:44 WARN RapidsPluginUtils: spark.rapids.sql.explain is set to
`NOT_ON_GPU`. Set it to 'NONE' to suppress the diagnostics logging about the query
placement on the GPU.
```

3. Untuk melihat operasi yang akan dijalankan pada GPU, lakukan langkah-langkah berikut untuk mengaktifkan logging tambahan. Perhatikan konfigurasi `spark.rapids.sql.explain` : ALL ""

```
aws emr-containers start-job-run \
--virtual-cluster-id VIRTUAL_CLUSTER_ID \
--execution-role-arn JOB_EXECUTION_ROLE \
--release-label emr-6.9.0-spark-rapids-latest \
--job-driver '{"sparkSubmitJobDriver": {"entryPoint": "local:///usr/lib/
spark/examples/jars/spark-examples.jar","entryPointArguments": ["10000"],
"sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi "}}' \
---configuration-overrides '{"applicationConfiguration":
[{"classification": "spark-defaults","properties":
{"spark.rapids.sql.explain":"ALL","spark.executor.instances":
"2","spark.executor.memory": "2G"}],"monitoringConfiguration":
{"cloudWatchMonitoringConfiguration": {"logGroupName":
"LOG_GROUP_NAME"},"s3MonitoringConfiguration": {"logUri": "LOG_GROUP_STREAM"}}}'
```

Perintah sebelumnya adalah contoh pekerjaan yang menggunakan GPU. Outputnya akan terlihat seperti contoh di bawah ini. Lihat kunci ini untuk bantuan memahami output:

- *— menandai operasi yang bekerja pada GPU
- !— menandai operasi yang tidak dapat berjalan pada GPU
- @— menandai operasi yang bekerja pada GPU, tetapi tidak akan berjalan karena ada di dalam paket yang tidak dapat berjalan pada GPU

```
22/11/15 01:22:58 INFO GpuOverrides: Plan conversion to the GPU took 118.64 ms
22/11/15 01:22:58 INFO GpuOverrides: Plan conversion to the GPU took 4.20 ms
22/11/15 01:22:58 INFO GpuOverrides: GPU plan transition optimization took 8.37 ms
22/11/15 01:22:59 WARN GpuOverrides:
  *Exec <ProjectExec> will run on GPU
    *Expression <Alias> substring(cast(date#149 as string), 0, 7) AS month#310
will run on GPU
      *Expression <Substring> substring(cast(date#149 as string), 0, 7) will run
on GPU
        *Expression <Cast> cast(date#149 as string) will run on GPU
  *Exec <SortExec> will run on GPU
    *Expression <SortOrder> date#149 ASC NULLS FIRST will run on GPU
  *Exec <ShuffleExchangeExec> will run on GPU
    *Partitioning <RangePartitioning> will run on GPU
      *Expression <SortOrder> date#149 ASC NULLS FIRST will run on GPU
    *Exec <UnionExec> will run on GPU
```

```

!Exec <ProjectExec> cannot run on GPU because not all expressions can
be replaced
  @Expression <AttributeReference> customerID#0 could run on GPU
  @Expression <Alias> Charge AS kind#126 could run on GPU
    @Expression <Literal> Charge could run on GPU
  @Expression <AttributeReference> value#129 could run on GPU
  @Expression <Alias> add_months(2022-11-15, cast(-(cast(_we0#142 as
bigint) + last_month#128L) as int)) AS date#149 could run on GPU
    ! <AddMonths> add_months(2022-11-15, cast(-
(cast(_we0#142 as bigint) + last_month#128L) as int)) cannot run
on GPU because GPU does not currently support the operator class
org.apache.spark.sql.catalyst.expressions.AddMonths
      @Expression <Literal> 2022-11-15 could run on GPU
      @Expression <Cast> cast(-(cast(_we0#142 as bigint) +
last_month#128L) as int) could run on GPU
        @Expression <UnaryMinus> -(cast(_we0#142 as bigint) +
last_month#128L) could run on GPU
          @Expression <Add> (cast(_we0#142 as bigint) +
last_month#128L) could run on GPU
            @Expression <Cast> cast(_we0#142 as bigint) could run on
GPU
              @Expression <AttributeReference> _we0#142 could run on
GPU
                @Expression <AttributeReference> last_month#128L could run
on GPU

```

Menggunakan integrasi Amazon Redshift untuk Apache Spark di Amazon EMR di EKS

Dengan Amazon EMR rilis 6.9.0 dan yang lebih baru, setiap gambar rilis menyertakan konektor antara [Apache](#) Spark dan Amazon Redshift. Dengan cara ini, Anda dapat menggunakan Spark di Amazon EMR di EKS untuk memproses data yang disimpan di Amazon Redshift. Integrasi ini didasarkan pada [konektor spark-redshift open-source](#). Untuk Amazon EMR di EKS, integrasi [Amazon Redshift untuk Apache Spark disertakan sebagai integrasi asli](#).

Topik

- [Meluncurkan aplikasi Spark menggunakan integrasi Amazon Redshift untuk Apache Spark](#)
- [Mengautentikasi dengan integrasi Amazon Redshift untuk Apache Spark](#)
- [Membaca dan menulis dari dan ke Amazon Redshift](#)

- [Pertimbangan dan batasan saat menggunakan konektor Spark](#)

Meluncurkan aplikasi Spark menggunakan integrasi Amazon Redshift untuk Apache Spark

Untuk menggunakan integrasi, Anda harus meneruskan dependensi Spark Redshift yang diperlukan dengan pekerjaan Spark Anda. Anda harus menggunakan `--jars` untuk menyertakan pustaka terkait konektor Redshift. Untuk melihat lokasi file lain yang didukung oleh `--jars` opsi, lihat bagian [Advanced Dependency Management](#) dari dokumentasi Apache Spark.

- `spark-redshift.jar`
- `spark-avro.jar`
- `RedshiftJDBC.jar`
- `minimal-json.jar`

Untuk meluncurkan aplikasi Spark dengan integrasi Amazon Redshift untuk Apache Spark di Amazon EMR pada rilis EKS 6.9.0 atau yang lebih baru, gunakan perintah contoh berikut. Perhatikan bahwa jalur yang tercantum dengan `--conf spark.jars` opsi adalah jalur default untuk file JAR.

```
aws emr-containers start-job-run \  
  
--virtual-cluster-id cluster_id \  
--execution-role-arn arn \  
--release-label emr-6.9.0-latest \  
--job-driver '{  
  "sparkSubmitJobDriver": {  
    "entryPoint": "s3://script_path",  
    "sparkSubmitParameters":  
      "--conf spark.kubernetes.file.upload.path=s3://upload_path  
      --conf spark.jars=  
        /usr/share/aws/redshift/jdbc/RedshiftJDBC.jar,  
        /usr/share/aws/redshift/spark-redshift/lib/spark-redshift.jar,  
        /usr/share/aws/redshift/spark-redshift/lib/spark-avro.jar,  
        /usr/share/aws/redshift/spark-redshift/lib/minimal-json.jar"  
      }  
  }  
'
```

Mengautentikasi dengan integrasi Amazon Redshift untuk Apache Spark

Bagian berikut menunjukkan opsi otentikasi dengan Amazon Redshift saat Anda mengintegrasikan dengan Apache Spark. Bagian-bagian tersebut menunjukkan cara mengambil kredensi login dan juga detail mengenai penggunaan driver JDBC dengan otentikasi IAM.

Gunakan AWS Secrets Manager untuk mengambil kredensial dan terhubung ke Amazon Redshift

Anda dapat menyimpan kredensial di Secrets Manager untuk mengautentikasi dengan aman ke Amazon Redshift. Anda dapat meminta pekerjaan Spark memanggil GetSecretValue API untuk mengambil kredensialnya:

```
from pyspark.sql import SQLContextimport boto3

sc = # existing SparkContext
sql_context = SQLContext(sc)

secretsmanager_client = boto3.client('secretsmanager',
    region_name=os.getenv('AWS_REGION'))
secret_manager_response = secretsmanager_client.get_secret_value(
    SecretId='string',
    VersionId='string',
    VersionStage='string'
)
username = # get username from secret_manager_response
password = # get password from secret_manager_response
url = "jdbc:redshift://redshifthost:5439/database?user=" + username + "&password="
    + password

# Access to Redshift cluster using Spark
```

Gunakan otentikasi berbasis IAM dengan Amazon EMR pada peran eksekusi pekerjaan EKS

Dimulai dengan Amazon EMR pada rilis EKS 6.9.0, driver Amazon Redshift JDBC versi 2.1 atau lebih tinggi dikemas ke lingkungan. Dengan driver JDBC 2.1 dan yang lebih tinggi, Anda dapat menentukan URL JDBC dan tidak menyertakan nama pengguna dan kata sandi mentah.

Sebagai gantinya, Anda dapat menentukan `jdbc:redshift:iam://` skema. Ini memerintahkan driver JDBC untuk menggunakan EMR Amazon Anda pada peran eksekusi pekerjaan EKS untuk mengambil kredensial secara otomatis.

Lihat [Mengonfigurasi koneksi JDBC atau ODBC untuk menggunakan kredensial IAM di Panduan Manajemen Amazon Redshift](#) untuk informasi selengkapnya.

Contoh URL berikut menggunakan `jdbc:redshift:iam://` skema.

```
jdbc:redshift:iam://examplecluster.abc123xyz789.us-west-2.redshift.amazonaws.com:5439/
dev
```

Izin berikut diperlukan untuk peran eksekusi pekerjaan Anda saat memenuhi ketentuan yang disediakan.

Izin	Kondisi bila diperlukan untuk peran pelaksanaan pekerjaan
<code>redshift:GetClusterCredentials</code>	Diperlukan untuk driver JDBC untuk mengambil kredensial dari Amazon Redshift
<code>redshift:DescribeCluster</code>	Diperlukan jika Anda menentukan kluster Amazon Redshift dan Wilayah AWS di URL JDBC, bukan titik akhir
<code>redshift-serverless:GetCredentials</code>	Diperlukan untuk driver JDBC untuk mengambil kredensial dari Amazon Redshift Serverless
<code>redshift-serverless:GetWorkgroup</code>	Diperlukan jika Anda menggunakan Amazon Redshift Tanpa Server dan Anda menentukan URL dalam hal nama grup kerja dan Wilayah

Kebijakan peran eksekusi pekerjaan Anda harus memiliki izin berikut.

```
{
  "Effect": "Allow",
  "Action": [
    "redshift:GetClusterCredentials",
    "redshift:DescribeCluster",
    "redshift-serverless:GetCredentials",
    "redshift-serverless:GetWorkgroup"
  ]
}
```

```
    ],
    "Resource": [
      "arn:aws:redshift:AWS_REGION:ACCOUNT_ID:dbname:CLUSTER_NAME/DATABASE_NAME",
      "arn:aws:redshift:AWS_REGION:ACCOUNT_ID:dbuser:DATABASE_NAME/USER_NAME"
    ]
  }
}
```

Otentikasi ke Amazon Redshift dengan driver JDBC

Tetapkan nama pengguna dan kata sandi di dalam URL JDBC

Untuk mengautentikasi pekerjaan Spark ke cluster Amazon Redshift, Anda dapat menentukan nama dan kata sandi database Amazon Redshift di URL JDBC.

Note

Jika Anda meneruskan kredensi database di URL, siapa pun yang memiliki akses ke URL juga dapat mengakses kredensialnya. Metode ini umumnya tidak disarankan karena ini bukan opsi yang aman.

Jika keamanan tidak menjadi perhatian untuk aplikasi Anda, Anda dapat menggunakan format berikut untuk mengatur nama pengguna dan kata sandi di URL JDBC:

```
jdbc:redshift://redshifthost:5439/database?user=username&password=password
```

Membaca dan menulis dari dan ke Amazon Redshift

Contoh kode berikut digunakan PySpark untuk membaca dan menulis data sampel dari dan ke database Amazon Redshift dengan API sumber data dan dengan SparkSQL.

Data source API

Gunakan PySpark untuk membaca dan menulis data sampel dari dan ke database Amazon Redshift dengan API sumber data.

```
import boto3
from pyspark.sql import SQLContext
```

```

sc = # existing SparkContext
sql_context = SQLContext(sc)

url = "jdbc:redshift:iam://redshifthost:5439/database"
aws_iam_role_arn = "arn:aws:iam::accountID:role/roleName"

df = sql_context.read \
    .format("io.github.spark_redshift_community.spark.redshift") \
    .option("url", url) \
    .option("dbtable", "tableName") \
    .option("tempdir", "s3://path/for/temp/data") \
    .option("aws_iam_role", "aws_iam_role_arn") \
    .load()

df.write \
    .format("io.github.spark_redshift_community.spark.redshift") \
    .option("url", url) \
    .option("dbtable", "tableName_copy") \
    .option("tempdir", "s3://path/for/temp/data") \
    .option("aws_iam_role", "aws_iam_role_arn") \
    .mode("error") \
    .save()

```

SparkSQL

Gunakan PySpark untuk membaca dan menulis data sampel dari dan ke database Amazon Redshift menggunakan SparkSQL.

```

import boto3
import json
import sys
import os
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .enableHiveSupport() \
    .getOrCreate()

url = "jdbc:redshift:iam://redshifthost:5439/database"
aws_iam_role_arn = "arn:aws:iam::accountID:role/roleName"

bucket = "s3://path/for/temp/data"

```

```
tableName = "tableName" # Redshift table name

s = f"""CREATE TABLE IF NOT EXISTS {tableName} (country string, data string)
  USING io.github.spark_redshift_community.spark.redshift
  OPTIONS (dbtable '{tableName}', tempdir '{bucket}', url '{url}', aws_iam_role
    '{aws_iam_role_arn}' ); """

spark.sql(s)

columns = ["country" ,"data"]
data = [("test-country", "test-data")]
df = spark.sparkContext.parallelize(data).toDF(columns)

# Insert data into table
df.write.insertInto(tableName, overwrite=False)
df = spark.sql(f"SELECT * FROM {tableName}")
df.show()
```

Pertimbangan dan batasan saat menggunakan konektor Spark

Konektor Spark mendukung berbagai cara untuk mengelola kredensial, untuk mengkonfigurasi keamanan, dan untuk terhubung dengan layanan lain. AWS Biasakan diri dengan rekomendasi dalam daftar ini untuk mengkonfigurasi koneksi fungsional dan tangguh.

- Kami menyarankan Anda mengaktifkan SSL untuk koneksi JDBC dari Spark di Amazon EMR ke Amazon Redshift.
- Kami menyarankan Anda mengelola kredensial untuk cluster Amazon Redshift sebagai praktik AWS Secrets Manager terbaik. Lihat [Menggunakan AWS Secrets Manager untuk mengambil kredensial untuk menghubungkan ke Amazon Redshift](#) sebagai contoh.
- Kami menyarankan Anda meneruskan peran IAM dengan parameter `aws_iam_role` untuk parameter autentikasi Amazon Redshift.
- Parameter `tempformat` saat ini tidak mendukung format Parquet.
- `tempdirURI` menunjuk ke lokasi Amazon S3. Direktori temp ini tidak dibersihkan secara otomatis dan karenanya dapat menambah biaya tambahan.
- Pertimbangkan rekomendasi berikut untuk Amazon Redshift:
 - Kami menyarankan Anda memblokir akses publik ke cluster Amazon Redshift.
 - Kami menyarankan Anda mengaktifkan pencatatan [audit Amazon Redshift](#).

- Sebaiknya aktifkan enkripsi saat [istirahat Amazon Redshift](#).
- Pertimbangkan rekomendasi berikut untuk Amazon S3:
 - Kami merekomendasikan untuk [memblokir akses publik ke bucket Amazon S3](#).
 - Kami menyarankan Anda menggunakan [enkripsi sisi server Amazon S3 untuk mengenkripsi bucket S3](#) yang Anda gunakan.
 - Sebaiknya gunakan [kebijakan siklus hidup Amazon S3](#) untuk menentukan aturan retensi bucket S3.
 - Amazon EMR selalu memverifikasi kode yang diimpor dari sumber terbuka ke dalam gambar. Demi keamanan, kami tidak mendukung kunci AWS akses encoding di `tempdir` URI sebagai metode otentikasi dari Spark ke Amazon S3.

Untuk informasi selengkapnya tentang penggunaan konektor dan parameter yang didukung, lihat sumber daya berikut:

- [Integrasi Amazon Redshift untuk Apache Spark di Panduan Manajemen](#) Amazon Redshift
- [Repositori spark-redshift komunitas](#) di Github

Menggunakan Volcano sebagai penjadwal khusus untuk Apache Spark di Amazon EMR di EKS

Dengan Amazon EMR di EKS, Anda dapat menggunakan operator Spark atau `spark-submit` untuk menjalankan pekerjaan Spark dengan penjadwal kustom Kubernetes. Tutorial ini mencakup cara menjalankan pekerjaan Spark dengan penjadwal Volcano pada antrian khusus.

Ikhtisar

[Volcano](#) dapat membantu mengelola penjadwalan Spark dengan fungsi-fungsi lanjutan seperti penjadwalan antrian, penjadwalan berbagi adil, dan reservasi sumber daya. Untuk informasi lebih lanjut tentang manfaat Volcano, lihat [Mengapa Spark memilih Volcano sebagai penjadwal batch bawaan di Kubernetes di](#) blog CNCF The Linux Foundation.

Instal dan atur Volcano

1. Pilih salah satu perintah `kubectl` berikut untuk menginstal Volcano, tergantung pada kebutuhan arsitektur Anda:

```
# x86_64
kubectl apply -f https://raw.githubusercontent.com/volcano-sh/volcano/v1.5.1/
installer/volcano-development.yaml
# arm64:
kubectl apply -f https://raw.githubusercontent.com/volcano-sh/volcano/v1.5.1/
installer/volcano-development-arm64.yaml
```

2. Siapkan contoh antrian Volcano. Antrian adalah kumpulan. [PodGroups](#) Antrian mengadopsi FIFO dan merupakan dasar untuk pembagian sumber daya.

```
cat << EOF > volcanoQ.yaml
apiVersion: scheduling.volcano.sh/v1beta1
kind: Queue
metadata:
  name: sparkqueue
spec:
  weight: 4
  reclaimable: false
  capability:
    cpu: 10
    memory: 20Gi
EOF

kubectl apply -f volcanoQ.yaml
```

3. Unggah contoh PodGroup manifes ke Amazon S3. PodGroup adalah sekelompok pod dengan asosiasi yang kuat. Anda biasanya menggunakan penjadwalan PodGroup untuk batch. Kirimkan sampel berikut PodGroup ke antrian yang Anda tentukan di langkah sebelumnya.

```
cat << EOF > podGroup.yaml
apiVersion: scheduling.volcano.sh/v1beta1
kind: PodGroup
spec:
  # Set minMember to 1 to make a driver pod
  minMember: 1
  # Specify minResources to support resource reservation.
  # Consider the driver pod resource and executors pod resource.
  # The available resources should meet the minimum requirements of the Spark job
  # to avoid a situation where drivers are scheduled, but they can't schedule
  # sufficient executors to progress.
  minResources:
    cpu: "1"
```

```
memory: "1Gi"
# Specify the queue. This defines the resource queue that the job should be
submitted to.
queue: sparkqueue
EOF

aws s3 mv podGroup.yaml s3://bucket-name
```

Jalankan aplikasi Spark dengan penjadwal Volcano dengan operator Spark

1. Jika Anda belum melakukannya, selesaikan langkah-langkah di bagian berikut untuk menyiapkan:
 - a. [Instal dan atur Volcano](#)
 - b. [Menyiapkan operator Spark untuk Amazon EMR di EKS](#)
 - c. [Instal operator Spark](#)

Sertakan argumen berikut saat Anda menjalankan helm `install spark-operator-demo` perintah:

```
--set batchScheduler.enable=true
--set webhook.enable=true
```

2. Buat file SparkApplication definisi `spark-pi.yaml` dengan `batchScheduler` dikonfigurasi.

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: spark-pi
  namespace: spark-operator
spec:
  type: Scala
  mode: cluster
  image: "895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.10.0:latest"
  imagePullPolicy: Always
  mainClass: org.apache.spark.examples.SparkPi
  mainApplicationFile: "local:///usr/lib/spark/examples/jars/spark-examples.jar"
  sparkVersion: "3.3.1"
```

```

batchScheduler: "volcano" #Note: You must specify the batch scheduler name as
'volcano'
restartPolicy:
  type: Never
volumes:
  - name: "test-volume"
    hostPath:
      path: "/tmp"
      type: Directory
driver:
  cores: 1
  coreLimit: "1200m"
  memory: "512m"
  labels:
    version: 3.3.1
  serviceAccount: emr-containers-sa-spark
  volumeMounts:
    - name: "test-volume"
      mountPath: "/tmp"
executor:
  cores: 1
  instances: 1
  memory: "512m"
  labels:
    version: 3.3.1
  volumeMounts:
    - name: "test-volume"
      mountPath: "/tmp"

```

3. Kirim aplikasi Spark dengan perintah berikut. Ini juga menciptakan SparkApplication objek yang disebut spark-pi:

```
kubectl apply -f spark-pi.yaml
```

4. Periksa peristiwa untuk SparkApplication objek dengan perintah berikut:

```
kubectl describe pods spark-pi-driver --namespace spark-operator
```

Acara pod pertama akan menunjukkan bahwa Volcano telah menjadwalkan pod:

Type	Reason	Age	From	Message
----	-----	----	----	-----

```
Normal Scheduled 23s volcano
pi-driver to integration-worker2
```

```
Successfully assigned default/spark-
```

Jalankan aplikasi Spark dengan penjadwal Volcano dengan **spark-submit**

1. Pertama, selesaikan langkah-langkah di [Menyiapkan spark-submit untuk Amazon EMR di EKS](#) bagian ini. Anda harus membangun spark-submit distribusi Anda dengan dukungan Volcano. Untuk informasi selengkapnya, lihat bagian Build dari [Using Volcano as Customized Scheduler for Spark on Kubernetes](#) di dokumentasi Apache Spark.
2. Tetapkan nilai untuk variabel lingkungan berikut:

```
export SPARK_HOME=spark-home
export MASTER_URL=k8s://Amazon-EKS-cluster-endpoint
```

3. Kirim aplikasi Spark dengan perintah berikut:

```
$SPARK_HOME/bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master $MASTER_URL \
  --conf spark.kubernetes.container.image=895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.10.0:latest \
  --conf spark.kubernetes.authenticate.driver.serviceAccountName=spark \
  --deploy-mode cluster \
  --conf spark.kubernetes.namespace=spark-operator \
  --conf spark.kubernetes.scheduler.name=volcano \
  --conf spark.kubernetes.scheduler.volcano.podGroupTemplateFile=/path/to/podgroup-template.yaml \
  --conf
spark.kubernetes.driver.pod.featureSteps=org.apache.spark.deploy.k8s.features.VolcanoFeatu
\
  --conf
spark.kubernetes.executor.pod.featureSteps=org.apache.spark.deploy.k8s.features.VolcanoFea
\
  local:///usr/lib/spark/examples/jars/spark-examples.jar 20
```

4. Periksa peristiwa untuk SparkApplication objek dengan perintah berikut:

```
kubectl describe pod spark-pi --namespace spark-operator
```

Acara pod pertama akan menunjukkan bahwa Volcano telah menjadwalkan pod:

Type	Reason	Age	From	Message
----	-----	----	----	-----
Normal	Scheduled	23s	volcano	Successfully assigned default/spark-pi-driver to integration-worker2

Menggunakan YuniKorn sebagai penjadwal khusus untuk Apache Spark di Amazon EMR di EKS

Dengan Amazon EMR di EKS, Anda dapat menggunakan operator Spark atau spark-submit untuk menjalankan pekerjaan Spark dengan penjadwal kustom Kubernetes. Tutorial ini mencakup cara menjalankan pekerjaan Spark dengan YuniKorn penjadwal pada antrian khusus dan penjadwalan geng.

Ikhtisar

[Apache YuniKorn](#) dapat membantu mengelola penjadwalan Spark dengan penjadwalan yang sadar aplikasi sehingga Anda dapat memiliki kontrol halus pada kuota dan prioritas sumber daya. Dengan penjadwalan geng, YuniKorn menjadwalkan aplikasi hanya jika permintaan sumber daya minimal untuk aplikasi dapat dipenuhi. Untuk informasi selengkapnya, lihat [Apa itu penjadwalan geng](#) di situs YuniKorn dokumentasi Apache.

Buat kluster Anda dan siapkan untuk YuniKorn

Gunakan langkah-langkah berikut untuk menerapkan kluster Amazon EKS. Anda dapat mengubah Wilayah AWS (`region`) dan Availability Zones (`availabilityZones`).

1. Tentukan cluster Amazon EKS:

```
cat <<EOF >eks-cluster.yaml
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: emr-eks-cluster
  region: eu-west-1
```

```
vpc:
  clusterEndpoints:
    publicAccess: true
    privateAccess: true

iam:
  withOIDC: true

nodeGroups:
  - name: spark-jobs
    labels: { app: spark }
    instanceType: m5.xlarge
    desiredCapacity: 2
    minSize: 2
    maxSize: 3
    availabilityZones: ["eu-west-1a"]
EOF
```

2. Buat cluster:

```
eksctl create cluster -f eks-cluster.yaml
```

3. Buat namespace spark-job tempat Anda akan menjalankan pekerjaan Spark:

```
kubectl create namespace spark-job
```

4. Selanjutnya, buat role dan role binding Kubernetes. Ini diperlukan untuk akun layanan yang digunakan oleh Spark job run.

a. Tentukan akun layanan, peran, dan pengikatan peran untuk pekerjaan Spark.

```
cat <<EOF >emr-job-execution-rbac.yaml
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: spark-sa
  namespace: spark-job
automountServiceAccountToken: false
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
```

```
metadata:
  name: spark-role
  namespace: spark-job
rules:
  - apiGroups: [ "", "batch", "extensions" ]
    resources: [ "configmaps", "serviceaccounts", "events", "pods", "pods/
exec", "pods/log", "pods/
portforward", "secrets", "services", "persistentvolumeclaims" ]
    verbs: [ "create", "delete", "get", "list", "patch", "update", "watch" ]
  ---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: spark-sa-rb
  namespace: spark-job
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: spark-role
subjects:
  - kind: ServiceAccount
    name: spark-sa
    namespace: spark-job
EOF
```

- b. Terapkan definisi role dan role binding Kubernetes dengan perintah berikut:

```
kubectl apply -f emr-job-execution-rbac.yaml
```

Instal dan atur YuniKorn

1. Gunakan perintah kubectl berikut untuk membuat namespace yunikorn guna menerapkan penjadwal Yunikorn:

```
kubectl create namespace yunikorn
```

2. Untuk menginstal scheduler, jalankan perintah Helm berikut:

```
helm repo add yunikorn https://apache.github.io/yunikorn-release
```

```
helm repo update
```

```
helm install yunikorn yunikorn/yunikorn --namespace yunikorn
```

Jalankan aplikasi Spark dengan YuniKorn penjadwal dengan operator Spark

1. Jika Anda belum melakukannya, selesaikan langkah-langkah di bagian berikut untuk menyiapkan:
 - a. [Buat klaster Anda dan siapkan untuk YuniKorn](#)
 - b. [Instal dan atur YuniKorn](#)
 - c. [Menyiapkan operator Spark untuk Amazon EMR di EKS](#)
 - d. [Instal operator Spark](#)

Sertakan argumen berikut saat Anda menjalankan `helm install spark-operator-demo` perintah:

```
--set batchScheduler.enable=true  
--set webhook.enable=true
```

2. Buat file SparkApplication definisispark-pi.yaml.

Untuk digunakan YuniKorn sebagai penjadwal untuk pekerjaan Anda, Anda harus menambahkan anotasi dan label tertentu ke definisi aplikasi Anda. Anotasi dan label menentukan antrian untuk pekerjaan Anda dan strategi penjadwalan yang ingin Anda gunakan.

Dalam contoh berikut, anotasi `schedulingPolicyParameters` mengatur penjadwalan geng untuk aplikasi. Kemudian, contoh membuat kelompok tugas, atau “geng” tugas, untuk menentukan kapasitas minimum yang harus tersedia sebelum menjadwalkan pod untuk memulai eksekusi pekerjaan. Dan akhirnya, ini menentukan dalam definisi kelompok tugas untuk menggunakan kelompok simpul dengan "app": "spark" label, seperti yang didefinisikan di [Buat klaster Anda dan siapkan untuk YuniKorn](#) bagian.

```
apiVersion: "sparkoperator.k8s.io/v1beta2"  
kind: SparkApplication
```

```
metadata:
  name: spark-pi
  namespace: spark-job
spec:
  type: Scala
  mode: cluster
  image: "895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.10.0:latest"
  imagePullPolicy: Always
  mainClass: org.apache.spark.examples.SparkPi
  mainApplicationFile: "local:///usr/lib/spark/examples/jars/spark-examples.jar"
  sparkVersion: "3.3.1"
  restartPolicy:
    type: Never
  volumes:
    - name: "test-volume"
      hostPath:
        path: "/tmp"
        type: Directory
  driver:
    cores: 1
    coreLimit: "1200m"
    memory: "512m"
    labels:
      version: 3.3.1
    annotations:
      yunikorn.apache.org/schedulingPolicyParameters: "placeholderTimeoutSeconds=30
gangSchedulingStyle=Hard"
      yunikorn.apache.org/task-group-name: "spark-driver"
      yunikorn.apache.org/task-groups: |-
        [{
          "name": "spark-driver",
          "minMember": 1,
          "minResource": {
            "cpu": "1200m",
            "memory": "1Gi"
          },
          "nodeSelector": {
            "app": "spark"
          }
        },
        {
          "name": "spark-executor",
          "minMember": 1,
          "minResource": {
```

```

        "cpu": "1200m",
        "memory": "1Gi"
    },
    "nodeSelector": {
        "app": "spark"
    }
  ]]
serviceAccount: spark-sa
volumeMounts:
  - name: "test-volume"
    mountPath: "/tmp"
executor:
  cores: 1
  instances: 1
  memory: "512m"
  labels:
    version: 3.3.1
  annotations:
    yunikorn.apache.org/task-group-name: "spark-executor"
  volumeMounts:
    - name: "test-volume"
      mountPath: "/tmp"

```

3. Kirim aplikasi Spark dengan perintah berikut. Ini juga menciptakan SparkApplication objek yang disebut spark-pi:

```
kubectl apply -f spark-pi.yaml
```

4. Periksa peristiwa untuk SparkApplication objek dengan perintah berikut:

```
kubectl describe sparkapplication spark-pi --namespace spark-job
```

Acara pod pertama akan menunjukkan bahwa YuniKorn telah menjadwalkan pod:

Type	Reason	Age	From	Message
----	-----	----	----	-----
Normal	Scheduling	3m12s	yunikorn	spark-operator/org-apache-spark-examples-sparkpi-2a777a88b98b8a95-driver is queued and waiting for allocation
Normal	GangScheduling	3m12s	yunikorn	Pod belongs to the taskGroup spark-driver, it will be scheduled as a gang member
Normal	Scheduled	3m10s	yunikorn	Successfully assigned spark
Normal	PodBindSuccessful	3m10s	yunikorn	Pod spark-operator/

Normal TaskCompleted	2m3s	yunikorn	Task spark-operator/
Normal Pulling	3m10s	kubelet	Pulling

Jalankan aplikasi Spark dengan YuniKorn scheduler dengan **spark-submit**

1. Pertama, selesaikan langkah-langkah di [Menyiapkan spark-submit untuk Amazon EMR di EKS](#) bagian ini.
2. Tetapkan nilai untuk variabel lingkungan berikut:

```
export SPARK_HOME=spark-home
export MASTER_URL=k8s://Amazon-EKS-cluster-endpoint
```

3. Kirim aplikasi Spark dengan perintah berikut:

Dalam contoh berikut, anotasi `schedulingPolicyParameters` mengatur penjadwalan geng untuk aplikasi. Kemudian, contoh membuat kelompok tugas, atau “geng” tugas, untuk menentukan kapasitas minimum yang harus tersedia sebelum menjadwalkan pod untuk memulai eksekusi pekerjaan. Dan akhirnya, ini menentukan dalam definisi kelompok tugas untuk menggunakan kelompok simpul dengan "app": "spark" label, seperti yang didefinisikan di [Buat klaster Anda dan siapkan untuk YuniKorn](#) bagian.

```
$SPARK_HOME/bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master $MASTER_URL \
  --conf spark.kubernetes.container.image=895885662937.dkr.ecr.us-  
west-2.amazonaws.com/spark/emr-6.10.0:latest \
  --conf spark.kubernetes.authenticate.driver.serviceAccountName=spark-sa \
  --deploy-mode cluster \
  --conf spark.kubernetes.namespace=spark-job \
  --conf spark.kubernetes.scheduler.name=yunikorn \
  --conf spark.kubernetes.driver.annotation.yunikorn.apache.org/  
schedulingPolicyParameters="placeholderTimeoutSeconds=30 gangSchedulingStyle=Hard" \
  \
  --conf spark.kubernetes.driver.annotation.yunikorn.apache.org/task-group-  
name="spark-driver" \
  --conf spark.kubernetes.executor.annotation.yunikorn.apache.org/task-group-  
name="spark-executor" \
  --conf spark.kubernetes.driver.annotation.yunikorn.apache.org/task-groups='[{
```

```

    "name": "spark-driver",
    "minMember": 1,
    "minResource": {
      "cpu": "1200m",
      "memory": "1Gi"
    },
    "nodeSelector": {
      "app": "spark"
    }
  },
  {
    "name": "spark-executor",
    "minMember": 1,
    "minResource": {
      "cpu": "1200m",
      "memory": "1Gi"
    },
    "nodeSelector": {
      "app": "spark"
    }
  }
]]' \
local:///usr/lib/spark/examples/jars/spark-examples.jar 20

```

4. Periksa peristiwa untuk SparkApplication objek dengan perintah berikut:

```
kubectl describe pod spark-driver-pod --namespace spark-job
```

Acara pod pertama akan menunjukkan bahwa YuniKorn telah menjadwalkan pod:

Type	Reason	Age	From	Message
Normal	Scheduling	3m12s	yunikorn	spark-operator/org-apache-spark-examples-sparkpi-2a777a88b98b8a95-driver is queued and waiting for allocation
Normal	GangScheduling	3m12s	yunikorn	Pod belongs to the taskGroup spark-driver, it will be scheduled as a gang member
Normal	Scheduled	3m10s	yunikorn	Successfully assigned spark
Normal	PodBindSuccessful	3m10s	yunikorn	Pod spark-operator/
Normal	TaskCompleted	2m3s	yunikorn	Task spark-operator/
Normal	Pulling	3m10s	kubelet	Pulling

Keamanan di Amazon EMR di EKS

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. Auditor pihak ketiga secara teratur menguji dan memverifikasi efektivitas keamanan kami sebagai bagian dari [Program AWS Kepatuhan Program AWS Kepatuhan](#) . Untuk mempelajari tentang program kepatuhan yang berlaku untuk Amazon EMR, lihat [AWS Layanan dalam Lingkup berdasarkan AWS Layanan Program Kepatuhan dalam Lingkup oleh Program](#) .
- 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 Amazon EMR di EKS. Topik berikut menunjukkan kepada Anda cara mengonfigurasi Amazon EMR di EKS untuk memenuhi tujuan keamanan dan kepatuhan Anda. Anda juga mempelajari cara menggunakan AWS layanan lain yang membantu Anda memantau dan mengamankan EMR Amazon Anda di sumber daya EKS.

Topik

- [Amazon EMR pada praktik terbaik keamanan EKS](#)
- [Perlindungan data](#)
- [Manajemen Identitas dan Akses](#)
- [Menggunakan Amazon EMR di EKS dengan AWS Lake Formation untuk kontrol akses berbutir halus](#)
- [Pencatatan dan pemantauan](#)
- [Menggunakan Hibah Akses Amazon S3 dengan Amazon EMR di EKS](#)
- [Validasi kepatuhan untuk Amazon EMR di EKS](#)
- [Ketahanan di Amazon EMR di EKS](#)

- [Keamanan infrastruktur dalam Amazon EMR di EKS](#)
- [Konfigurasi dan analisis kerentanan](#)
- [Connect ke Amazon EMR di EKS Menggunakan antarmuka VPC endpoint](#)
- [Atur akses lintas akun untuk Amazon EMR di EKS](#)

Amazon EMR pada praktik terbaik keamanan EKS

Amazon EMR di EKS menyediakan sejumlah fitur keamanan untuk dipertimbangkan ketika 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.

Note

Untuk praktik terbaik keamanan lainnya, lihat [Amazon EMR pada praktik terbaik keamanan EKS](#).

Terapkan prinsip hak istimewa paling rendah

Amazon EMR di EKS menyediakan kebijakan akses granular untuk aplikasi menggunakan IAM role, seperti peran eksekusi. Peran eksekusi ini dipetakan ke akun layanan Kubernetes melalui kebijakan kepercayaan IAM role. Amazon EMR di EKS menciptakan pod dalam namespace Amazon EKS terdaftar yang mengeksekusi kode aplikasi yang disediakan pengguna. Pod job yang menjalankan kode aplikasi mengambil peran eksekusi saat menghubungkan ke AWS layanan lain. Kami merekomendasikan bahwa peran eksekusi diberikan hanya seperangkat minimum hak istimewa yang diperlukan oleh tugas, seperti mencakup aplikasi Anda dan akses ke tujuan log. Kami juga merekomendasikan mengaudit tugas untuk izin secara teratur dan pada setiap perubahan pada kode aplikasi.

Daftar kontrol akses untuk titik akhir

Titik akhir terkelola dapat dibuat hanya untuk kluster EKS yang telah dikonfigurasi untuk menggunakan setidaknya satu subnet privat di VPC Anda. Konfigurasi ini membatasi akses ke penyeimbang beban yang dibuat oleh titik akhir terkelola sehingga mereka hanya dapat diakses

dari VPC Anda. Untuk lebih meningkatkan keamanan, kami sarankan Anda mengonfigurasi grup keamanan dengan penyeimbang beban ini sehingga mereka dapat membatasi lalu lintas masuk ke serangkaian alamat IP yang dipilih.

Dapatkan pembaruan keamanan terbaru untuk gambar kustom

Untuk menggunakan gambar kustom dengan Amazon EMR di EKS, Anda dapat menginstal biner dan perpustakaan pada gambar. Anda bertanggung jawab untuk patch keamanan biner yang Anda tambahkan ke gambar. Gambar Amazon EMR di EKS secara teratur di-patch dengan patch keamanan terbaru. Untuk mendapatkan gambar terbaru, Anda harus membangun kembali gambar kustom setiap kali ada versi gambar dasar baru dari rilis Amazon EMR. Untuk informasi lebih lanjut, lihat [Amazon EMR pada rilis EKS](#) dan [Detail untuk memilih URI gambar dasar](#).

Batasi akses kredensial pod

Kubernetes mendukung beberapa metode penetapan kredensial ke pod. Penyediaan beberapa penyedia kredensial dapat meningkatkan kompleksitas model keamanan Anda. Amazon EMR di EKS telah mengadopsi penggunaan [IAM role untuk akun layanan \(IRSA\)](#) sebagai penyedia kredensial standar dalam namespace EKS terdaftar. Metode lain tidak didukung, termasuk [kube2iam](#), [kiam](#) dan menggunakan profil instans EC2 dari instans yang berjalan di kluster.

Mengisolasi kode aplikasi yang tidak tepercaya

Amazon EMR di EKS tidak memeriksa integritas kode aplikasi yang dikirimkan oleh pengguna sistem. Jika Anda menjalankan kluster virtual multi-tenanted yang dikonfigurasi menggunakan beberapa peran eksekusi yang dapat digunakan untuk mengirimkan tugas oleh penyewa tidak tepercaya yang menjalankan kode arbitrer, ada risiko aplikasi berbahaya meningkatkan hak istimewanya. Dalam situasi ini, pertimbangkan untuk mengisolasi peran eksekusi dengan hak istimewa yang sama ke kluster virtual yang berbeda.

Izin kontrol akses berbasis peran (RBAC)

Administrator harus mengontrol ketat izin kontrol akses berbasis peran (RBAC) untuk namespace terkelola Amazon EMR di EKS. Minimal, izin berikut tidak boleh diberikan kepada pengirim tugas di namespace terkelola Amazon EMR di EKS.

- Izin Kubernetes RBAC untuk memodifikasi configmap - karena Amazon EMR di EKS menggunakan configmap Kubernetes untuk menghasilkan templat pod terkelola yang memiliki nama akun layanan terkelola. Atribut ini seharusnya tidak bermutasi.

- Izin Kubernetes RBAC untuk `exec` ke pod Amazon EMR di EKS - untuk menghindari memberikan akses ke templat pod terkelola yang memiliki nama SA terkelola. Atribut ini seharusnya tidak bermutasi. Izin ini juga dapat memberikan akses ke token JWT dipasang ke pod yang kemudian dapat digunakan untuk mengambil kredensial peran eksekusi.
- Izin Kubernetes RBAC untuk membuat pod - untuk mencegah pengguna membuat pod menggunakan Kubernetes ServiceAccount yang dapat dipetakan ke peran IAM dengan hak istimewa lebih dari pengguna. AWS
- Izin Kubernetes RBAC untuk menerapkan webhook yang bermutasi - untuk mencegah pengguna menggunakan webhook yang bermutasi untuk mengubah nama Kubernetes untuk pod yang dibuat oleh Amazon EMR di EKS. ServiceAccount
- Izin RBAC Kubernetes untuk membaca rahasia Kubernetes - untuk mencegah pengguna membaca data rahasia yang tersimpan dalam rahasia ini.

Membatasi akses ke nodegroup IAM role atau kredensial profil instans

- Kami menyarankan Anda menetapkan AWS izin minimum untuk peran IAM nodegroup. Ini membantu untuk menghindari eskalasi hak istimewa dengan kode yang dapat dijalankan menggunakan kredensial profil instans dari simpul pekerja EKS.
- Untuk memblokir akses sepenuhnya ke kredensial profil instans untuk semua pod yang berjalan di namespace terkelola Amazon EMR di EKS, kami sarankan Anda menjalankan perintah `iptables` pada simpul EKS. Untuk informasi lebih lanjut, lihat [Membatasi akses ke kredensial profil instans Amazon EC2](#). Namun, penting untuk mencakup dengan benar IAM role akun layanan Anda sehingga pod Anda memiliki semua izin yang diperlukan. Sebagai contoh, IAM role simpul diberikan izin untuk menarik gambar kontainer dari Amazon ECR. Jika pod tidak ditugaskan izin tersebut, pod tidak dapat menarik gambar kontainer dari Amazon ECR. Plugin VPC CNI juga perlu diperbarui. Untuk informasi lebih lanjut, lihat [Panduan: Memperbarui plugin VPC CNI untuk menggunakan IAM role untuk akun layanan](#).

Perlindungan data

[Model tanggung jawab AWS bersama](#) berlaku untuk perlindungan data di Amazon EMR di EKS. Seperti yang dijelaskan dalam model AWS ini, bertanggung jawab untuk melindungi infrastruktur global yang menjalankan semua AWS Cloud. Anda bertanggung jawab untuk memelihara kendali terhadap konten yang di-hosting pada infrastruktur ini. Konten ini meliputi konfigurasi keamanan dan tugas manajemen untuk berbagai layanan AWS yang Anda gunakan. Untuk informasi selengkapnya

tentang privasi data, lihat [FAQ Privasi Data](#). Untuk informasi tentang perlindungan data di Eropa, lihat [Model Tanggung Jawab AWS Bersama dan posting blog GDPR](#) di Blog AWS Keamanan.

Untuk tujuan perlindungan data, kami menyarankan Anda untuk melindungi kredensial AWS akun dan menyiapkan akun individual dengan AWS Identity and Access Management (IAM). Dengan cara ini, setiap pengguna hanya diberi izin yang diperlukan untuk memenuhi tanggung jawab tugas mereka. Kami juga merekomendasikan agar Anda mengamankan data Anda dengan cara-cara berikut ini:

- Gunakan autentikasi multi-faktor (MFA) pada setiap akun.
- Gunakan SSL/TLS untuk berkomunikasi dengan AWS sumber daya. Kami merekomendasikan TLS 1.2 atau versi yang lebih baru.
- Siapkan API dan pencatatan aktivitas pengguna dengan AWS CloudTrail.
- Gunakan solusi AWS enkripsi, bersama dengan semua kontrol keamanan default dalam AWS layanan.
- Gunakan layanan keamanan terkelola tingkat lanjut seperti Amazon Macie, yang membantu menemukan dan mengamankan data pribadi yang disimpan di Amazon S3.
- Gunakan opsi enkripsi Amazon EMR di EKS pada opsi enkripsi untuk mengenkripsi data at rest dan transit.
- Jika Anda memerlukan modul kriptografi tervalidasi FIPS 140-2 saat mengakses AWS melalui antarmuka baris perintah atau API, gunakan titik akhir FIPS. Untuk informasi lebih lanjut tentang titik akhir FIPS yang tersedia, lihat [Standar Pemrosesan Informasi Federal \(FIPS\) 140-2](#).

Kami sangat merekomendasikan agar Anda tidak memasukkan informasi identifikasi sensitif apapun, seperti nomor rekening pelanggan Anda, ke dalam kolom isian teks bebas seperti kolom Nama. Ini termasuk saat Anda bekerja dengan Amazon EMR di EKS atau AWS layanan lain menggunakan konsol, API AWS CLI, atau AWS SDK. Data apa pun yang Anda masukkan ke dalam Amazon EMR di EKS atau layanan lain mungkin akan diambil untuk dimasukkan ke dalam log diagnostik. Saat Anda menyediakan URL ke peladen eksternal, jangan menyertakan informasi kredensial dalam URL untuk memvalidasi permintaan Anda ke peladen tersebut.

Enkripsi saat istirahat

Enkripsi data membantu mencegah pengguna yang tidak sah membaca data pada kluster dan sistem penyimpanan data terkait. Ini termasuk data yang disimpan ke media persisten, yang dikenal sebagai

data at rest, dan data yang mungkin dicegat saat perjalanan jaringan, yang dikenal sebagai data dalam transit.

Enkripsi data memerlukan kunci dan sertifikat. Anda dapat memilih dari beberapa opsi, termasuk kunci yang dikelola oleh AWS Key Management Service, kunci yang dikelola oleh Amazon S3, dan kunci serta sertifikat dari penyedia khusus yang Anda berikan. Saat menggunakan AWS KMS sebagai penyedia kunci Anda, biaya berlaku untuk penyimpanan dan penggunaan kunci enkripsi. Untuk informasi lebih lanjut, lihat [AWS KMS Harga](#).

Sebelum Anda menentukan opsi enkripsi, tentukan sistem manajemen kunci dan sertifikat yang ingin Anda gunakan. Kemudian buat kunci dan sertifikat untuk penyedia kustom yang Anda tentukan sebagai bagian dari pengaturan enkripsi.

Enkripsi saat istirahat untuk data EMRFS di Amazon S3

Enkripsi Amazon S3 bekerja dengan objek EMR File System (EMRFS) yang dibaca dari dan ditulis ke Amazon S3. Anda menentukan Amazon S3 server-side encryption (SSE) atau client-side encryption (CSE) sebagai Mode enkripsi default saat Anda mengaktifkan enkripsi saat istirahat. Secara opsional, Anda dapat menentukan metode enkripsi yang berbeda untuk setiap bucket menggunakan Per penimpaan enkripsi bucket. Keamanan Lapisan Pengangkutan (TLS) terlepas dari apakah enkripsi Amazon S3 diaktifkan, Keamanan Lapisan Pengangkutan (TLS) mengenkripsi objek EMRFS dalam transit antara simpul kluster EMR dan Amazon S3. Untuk informasi selengkapnya tentang enkripsi Amazon S3, lihat [Melindungi Data Menggunakan Enkripsi](#) di Panduan Developer Amazon Simple Storage Service.

Note

Saat Anda menggunakan AWS KMS, biaya berlaku untuk penyimpanan dan penggunaan kunci enkripsi. Untuk informasi lebih lanjut, lihat [AWS KMS Harga](#).

Enkripsi sisi server Amazon S3

Saat Anda mengatur enkripsi sisi server Amazon S3, Amazon S3 akan mengenkripsi data pada tingkat objek saat menulis data ke disk dan mendekripsi data saat diakses. Untuk informasi selengkapnya tentang SSE, lihat [Melindungi Data Menggunakan Server-Side Enkripsi](#) di Panduan Pengembang Layanan Penyimpanan Sederhana Amazon.

Anda dapat memilih antara dua sistem pengelolaan kunci yang berbeda ketika Anda menentukan SSE di Amazon EMR di EKS:

- SSE-S3- Amazon S3 mengelola kunci untuk Anda.
- SSE-KMS- Anda menggunakan AWS KMS key untuk mengatur dengan kebijakan yang sesuai untuk Amazon EMR di EKS.

SSE dengan kunci yang disediakan pelanggan (SSE-C) tidak tersedia untuk digunakan dengan Amazon EMR di EKS.

Tip

Untuk mengurangi AWS KMS biaya saat menggunakan SSE-KMS, pertimbangkan untuk mengaktifkan Kunci Bucket Amazon S3 di bucket Amazon S3 Anda. Kunci Bucket Amazon S3 menggunakan kunci tingkat ember berumur pendek untuk mengurangi panggilan AWS KMS API hingga 99 persen. Sebelum mengaktifkan Amazon S3 Bucket Keys, tinjau IAM AWS KMS dan kebijakan utama Anda — konteks enkripsi berubah dari objek Amazon S3 ARN ke ARN bucket, yang dapat memengaruhi kebijakan yang menggunakan objek ARN untuk kontrol akses. Untuk informasi selengkapnya, lihat [Mengurangi biaya SSE-KMS dengan Kunci Bucket Amazon S3](#) di Panduan Pengguna Layanan Penyimpanan Sederhana Amazon.

Enkripsi di sisi klien Amazon S3

Dengan enkripsi sisi klien Amazon S3, enkripsi dan dekripsi Amazon S3 dilakukan di klien EMRFS pada kluster Anda. Objek dienkripsi sebelum diunggah ke Amazon S3 dan didekripsi setelah diunduh. Penyedia yang Anda tentukan menyediakan kunci enkripsi yang digunakan klien. Klien dapat menggunakan kunci yang disediakan oleh AWS KMS (CSE-KMS) atau kelas Java kustom yang menyediakan kunci root sisi klien (CSE-C). Spesifikasi enkripsi sedikit berbeda antara CSE-KMS dan CSE-C, tergantung pada penyedia yang ditentukan dan metadata objek yang didekripsi atau dienkripsi. Untuk informasi selengkapnya tentang perbedaan ini, lihat [Melindungi Data Menggunakan Client-Side Enkripsi](#) di Panduan Pengembang Layanan Penyimpanan Sederhana Amazon.

Note

Amazon S3 CSE hanya memastikan bahwa data EMRFS yang dipertukarkan dengan Amazon S3 dienkripsi; tidak semua data pada volume instans kluster dienkripsi. Selain itu,

karena Hue tidak menggunakan EMRFS, objek yang Hue S3 File Browser tulis ke Amazon S3 tidak dienkripsi.

Enkripsi disk lokal

Apache Spark mendukung mengenkripsi data sementara ditulis ke disk lokal. Ini mencakup file acak, spill acak, dan blok data yang disimpan pada disk untuk variabel baik caching maupun siaran. Ini tidak mencakup enkripsi data output yang dihasilkan oleh aplikasi dengan API seperti `saveAsHadoopFile` atau `saveAsTable`. Ini juga mungkin tidak mencakup file sementara yang dibuat secara eksplisit oleh pengguna. Untuk informasi selengkapnya, lihat [Enkripsi Penyimpanan Lokal](#) dalam dokumentasi Spark. Spark tidak mendukung data terenkripsi pada disk lokal, seperti data menengah yang ditulis ke disk lokal oleh proses eksekutor ketika data tidak cocok dalam memori. Data yang bertahan ke disk dicakup untuk waktu aktif tugas, dan kunci yang digunakan untuk mengenkripsi data dihasilkan secara dinamis oleh Spark untuk setiap tugas berjalan. Setelah pekerjaan Spark berakhir, tidak ada proses lain yang dapat mendekripsi data.

Untuk driver dan pod eksekutor, Anda mengenkripsi data at rest yang bertahan untuk volume terpasang. Ada tiga opsi penyimpanan AWS native berbeda yang dapat Anda gunakan dengan Kubernetes: [EBS](#), [EFS](#), dan [FSx for Lustre](#). Ketiganya menawarkan enkripsi saat istirahat menggunakan kunci yang dikelola layanan atau file AWS KMS key. Untuk informasi selengkapnya lihat [Panduan Praktik Terbaik EKS](#). Dengan pendekatan ini, semua data yang disimpan ke volume terpasang dienkripsi.

Manajemen kunci

Anda dapat mengonfigurasi KMS untuk memutar tombol KMS Anda secara otomatis. Ini merotasi kunci Anda setahun sekali sambil menyimpan kunci lama tanpa batas waktu sehingga data Anda masih dapat didekripsi. Untuk informasi tambahan, lihat [Memutar. AWS KMS keys](#)

Enkripsi saat bergerak

Beberapa mekanisme enkripsi diaktifkan dengan enkripsi dalam transit. Ini adalah fitur sumber daya terbuka, khusus aplikasi, dan dapat bervariasi dengan rilis Amazon EMR di EKS. Fitur enkripsi khusus aplikasi berikut dapat diaktifkan dengan Amazon EMR di EKS:

- Spark
 - Komunikasi RPC internal antara komponen Spark, seperti layanan transfer blok dan layanan shuffle eksternal, dienkripsi menggunakan cipher AES-256 di Amazon EMR versi 5.9.0 dan yang

lebih baru. Dalam rilis sebelumnya, komunikasi RPC internal dienkripsi menggunakan SASL dengan DIGEST-MD5 sebagai cipher.

- Komunikasi protokol HTTP dengan antarmuka pengguna seperti Spark History Server dan server HTTPS-enabled file dienkripsi menggunakan konfigurasi SSL Spark. Untuk informasi lebih lanjut, lihat [Konfigurasi SSL](#) di dokumentasi Spark.

Untuk informasi lebih lanjut, lihat [Pengaturan keamanan Spark](#).

- Anda harus mengizinkan hanya koneksi terenkripsi melalui HTTPS (TLS) menggunakan [aws:SecureTransport condition pada kebijakan IAM](#) bucket Amazon S3.
- Hasil kueri yang di-stream ke klien JDBC atau ODBC dienkripsi menggunakan TLS.

Manajemen Identitas dan Akses

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 diotorisasi (memiliki izin) untuk menggunakan sumber daya Amazon EMR di EKS. IAM adalah Layanan AWS yang dapat Anda gunakan tanpa biaya tambahan.

Topik

- [Audiens](#)
- [Mengautentikasi dengan identitas](#)
- [Mengelola akses menggunakan kebijakan](#)
- [Bagaimana Amazon EMR di EKS bekerja dengan IAM](#)
- [Menggunakan peran terkait layanan untuk Amazon EMR di EKS](#)
- [Kebijakan terkelola untuk Amazon EMR di EKS](#)
- [Menggunakan peran eksekusi tugas dengan Amazon EMR di EKS](#)
- [Contoh kebijakan berbasis identitas untuk Amazon EMR di EKS](#)
- [Kebijakan untuk kendali akses berbasis tanda](#)
- [Memecahkan Masalah identitas dan akses Amazon EMR di EKS](#)

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 EMR di EKS](#))
- Administrator layanan - tentukan akses pengguna dan mengirimkan permintaan izin (lihat [Bagaimana Amazon EMR di EKS bekerja dengan IAM](#))
- Administrator IAM - tulis kebijakan untuk mengelola akses (lihat [Contoh kebijakan berbasis identitas untuk Amazon EMR di EKS](#))

Mengautentikasi dengan identitas

Otentikasi adalah cara Anda masuk AWS menggunakan kredensi identitas Anda. Anda harus diautentikasi sebagai Pengguna root akun AWS, pengguna IAM, atau dengan mengasumsikan peran IAM.

Anda dapat masuk sebagai identitas federasi menggunakan kredensial dari sumber identitas seperti AWS IAM Identity Center (Pusat Identitas IAM), autentikasi masuk tunggal, atau kredensial 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 kredensi 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 kredensial 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 melakukannya.

Kebijakan berbasis identitas

Kebijakan berbasis identitas 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.

Kebijakan berbasis identitas dapat berupa kebijakan inline (disematkan langsung ke dalam satu identitas) atau kebijakan terkelola (kebijakan mandiri yang dilampirkan pada banyak identitas). Untuk mempelajari cara memilih antara kebijakan terkelola dan kebijakan inline, lihat [Pilih antara kebijakan terkelola dan kebijakan inline](#) dalam Panduan Pengguna IAM.

Kebijakan berbasis sumber daya

Kebijakan berbasis sumber daya 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.

Kebijakan berbasis sumber daya merupakan 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 (SCPs) — Tentukan 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 (RCPs) — Tetapkan izin maksimum yang tersedia untuk sumber daya di akun Anda. Untuk informasi selengkapnya, lihat [Kebijakan kontrol sumber daya \(RCPs\)](#) di Panduan AWS Organizations Pengguna.
- 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 EMR di EKS bekerja dengan IAM

Sebelum menggunakan IAM untuk mengelola akses ke Amazon EMR di EKS, pelajari fitur IAM apa yang tersedia untuk digunakan dengan Amazon EMR di EKS.

Fitur IAM yang dapat Anda gunakan dengan Amazon EMR di EKS

Fitur IAM	Dukungan Amazon EMR di EKS
Kebijakan berbasis identitas	Ya
Kebijakan berbasis sumber daya	Tidak
Tindakan kebijakan	Ya
Sumber daya kebijakan	Ya
Kunci kondisi kebijakan	Ya
ACLs	Tidak
ABAC (tanda dalam kebijakan)	Ya
Kredensial sementara	Ya
Izin principal	Ya

Fitur IAM	Dukungan Amazon EMR di EKS
Peran layanan	Tidak
Peran terkait layanan	Ya

Untuk mendapatkan tampilan tingkat tinggi tentang cara Amazon EMR di EKS dan layanan AWS lainnya bekerja dengan sebagian besar fitur IAM, [AWS lihat layanan yang bekerja dengan IAM di Panduan Pengguna IAM](#).

Kebijakan berbasis identitas untuk Amazon EMR di EKS

Mendukung kebijakan berbasis identitas: Ya

Kebijakan berbasis identitas adalah dokumen kebijakan izin JSON yang dapat Anda lampirkan ke sebuah identitas, seperti pengguna IAM, grup pengguna IAM, atau peran IAM. 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 IAM JSON](#) dalam Panduan Pengguna IAM.

Contoh kebijakan berbasis identitas untuk Amazon EMR di EKS

Untuk melihat contoh kebijakan berbasis identitas Amazon EMR di EKS, lihat [Contoh kebijakan berbasis identitas untuk Amazon EMR di EKS](#).

Kebijakan berbasis sumber daya di dalam Amazon EMR di EKS

Mendukung kebijakan berbasis sumber daya: Tidak

Kebijakan berbasis sumber daya 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

Untuk mengaktifkan akses lintas akun, Anda dapat menentukan secara spesifik seluruh akun atau entitas IAM di akun lain sebagai principal dalam kebijakan berbasis sumber daya. Untuk informasi selengkapnya, lihat [Akses sumber daya lintas akun di IAM](#) dalam Panduan Pengguna IAM.

Tindakan kebijakan untuk Amazon EMR di EKS

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 Amazon EMR di EKS, lihat [Kunci tindakan, sumber daya, dan kondisi untuk Amazon EMR di EKS](#) dalam Referensi Otorisasi Layanan.

Tindakan kebijakan di Amazon EMR di EKS menggunakan prefiks berikut sebelum tindakan:

```
emr-containers
```

Untuk menetapkan beberapa tindakan dalam satu pernyataan, pisahkan dengan koma.

```
"Action": [  
  "emr-containers:action1",  
  "emr-containers:action2"  
]
```

Untuk melihat contoh kebijakan berbasis identitas Amazon EMR di EKS, lihat [Contoh kebijakan berbasis identitas untuk Amazon EMR di EKS](#).

Sumber daya kebijakan untuk Amazon EMR di EKS

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 EMR Amazon pada jenis sumber daya EKS dan jenisnya ARNs, lihat Sumber daya yang [ditentukan oleh Amazon EMR di EKS](#) dalam Referensi Otorisasi Layanan. Untuk mempelajari tindakan mana yang dapat Anda tentukan ARN setiap sumber daya, lihat [Tindakan, sumber daya, dan kunci ketentuan untuk Amazon EMR di EKS](#).

Untuk melihat contoh kebijakan berbasis identitas Amazon EMR di EKS, lihat [Contoh kebijakan berbasis identitas untuk Amazon EMR di EKS](#).

Kunci ketentuan kebijakan untuk Amazon EMR di EKS

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 Amazon EMR di EKS dan untuk mempelajari tindakan dan sumber daya mana yang dapat Anda gunakan dengan kunci ketentuan, lihat [Kunci tindakan, sumber daya, dan kondisi untuk Amazon EMR di EKS](#) dalam Referensi Otorisasi Layanan.

Untuk melihat contoh kebijakan berbasis identitas Amazon EMR di EKS, lihat [Contoh kebijakan berbasis identitas untuk Amazon EMR di EKS](#).

Daftar kontrol akses (ACLs) di Amazon EMR di EKS

Mendukung ACLs: Tidak

Access control lists (ACLs) mengontrol prinsipal mana (anggota akun, pengguna, atau peran) yang memiliki izin untuk mengakses sumber daya. ACLs mirip dengan kebijakan berbasis sumber daya, meskipun mereka tidak menggunakan format dokumen kebijakan JSON.

Kontrol akses berbasis atribut (ABAC) dengan Amazon EMR di EKS

Mendukung ABAC (tanda dalam kebijakan) Ya

Kontrol akses berbasis atribut (ABAC) adalah strategi otorisasi yang menentukan izin berdasarkan atribut tanda. 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 dengan langkah-langkah untuk mengatur ABAC, lihat [Gunakan kontrol akses berbasis atribut \(ABAC\)](#) di Panduan Pengguna IAM.

Menggunakan kredensial Sementara dengan Amazon EMR di EKS

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.

Izin prinsipal lintas layanan untuk Amazon EMR di EKS

Mendukung sesi akses terusan (FAS): Ya

Sesi akses terusan (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 Amazon EMR di EKS

Mendukung peran layanan	Tidak
-------------------------	-------

Peran terkait layanan untuk Amazon EMR di EKS

Mendukung peran terkait layanan	Ya
---------------------------------	----

Untuk informasi selengkapnya tentang cara membuat atau mengelola peran terkait layanan, lihat [AWS layanan yang bekerja dengan IAM](#). Cari layanan dalam tabel yang memiliki Yes di kolom Peran terkait layanan. Pilih tautan Ya untuk melihat dokumentasi peran terkait layanan untuk layanan tersebut.

Menggunakan peran terkait layanan untuk Amazon EMR di EKS

[Amazon EMR di EKS menggunakan peran terkait layanan AWS Identity and Access Management \(IAM\)](#). Peran terkait layanan adalah jenis IAM role unik yang terhubung langsung ke Amazon EMR di EKS. Peran terkait layanan telah ditentukan sebelumnya oleh Amazon EMR di EKS dan mencakup semua izin yang diperlukan layanan untuk memanggil layanan lain atas nama Anda. AWS

Peran tertaut layanan mempermudah pengaturan Amazon EMR di EKS karena Anda tidak perlu menambahkan izin yang diperlukan secara manual. Amazon EMR di EKS menentukan izin dari peran tertaut layanan, kecuali jika ditentukan lain, hanya Amazon EMR di EKS yang dapat mengambil perannya. Izin yang ditentukan mencakup kebijakan kepercayaan dan kebijakan izin, serta bahwa kebijakan izin tidak dapat dilampirkan ke entitas IAM lainnya.

Anda dapat menghapus peran yang terhubung dengan layanan hanya setelah menghapus sumber daya terkait terlebih dahulu. Ini melindungi sumber daya Amazon EMR di EKS Anda karena Anda tidak dapat secara tidak sengaja menghapus izin untuk mengakses sumber daya.

Untuk informasi tentang layanan lain yang mendukung peran yang terhubung dengan layanan, lihat [Layanan AWS yang Berfungsi dengan IAM](#) dan cari layanan yang memiliki Ya di kolom Peran yang Terhubung dengan Layanan. Pilih Ya dengan tautan untuk melihat dokumentasi peran terkait layanan untuk layanan tersebut.

Izin peran terkait layanan untuk Amazon EMR di EKS

Amazon EMR di EKS menggunakan peran terkait layanan bernama.

`AWSServiceRoleForAmazonEMRContainers`

`AWSServiceRoleForAmazonEMRContainers` peran terkait layanan memercayakan layanan berikut untuk menjalankan peran tersebut:

- `emr-containers.amazonaws.com`

Kebijakan izin peran `AmazonEMRContainersServiceRolePolicy` memungkinkan Amazon EMR di EKS untuk menyelesaikan serangkaian tindakan pada sumber daya yang ditentukan, seperti yang ditunjukkan pernyataan kebijakan berikut.

Note

Konten kebijakan terkelola berubah, sehingga kebijakan yang ditampilkan di sini mungkin out-of-date. Lihat sebagian besar dokumentasi up-to-date kebijakan di [Amazon EMRContainers ServiceRolePolicy](#) di Panduan Referensi Kebijakan AWS Terkelola.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "eks:DescribeCluster",
        "eks:ListNodeGroups",
        "eks:DescribeNodeGroup",
        "ec2:DescribeRouteTables",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
```

```

    "elasticloadbalancing:DescribeInstanceHealth",
    "elasticloadbalancing:DescribeLoadBalancers",
    "elasticloadbalancing:DescribeTargetGroups",
    "elasticloadbalancing:DescribeTargetHealth",
    "eks:ListPodIdentityAssociations",
    "eks:DescribePodIdentityAssociation"
  ],
  "Resource": [
    "*"
  ],
  "Sid": "AllowEKSDescribecluster"
},
{
  "Effect": "Allow",
  "Action": [
    "acm:ImportCertificate",
    "acm:AddTagsToCertificate"
  ],
  "Resource": [
    "*"
  ],
  "Condition": {
    "StringEquals": {
      "aws:RequestTag/emr-container:endpoint:managed-certificate": "true"
    }
  },
  "Sid": "AllowACMImportcertificate"
},
{
  "Effect": "Allow",
  "Action": [
    "acm>DeleteCertificate"
  ],
  "Resource": [
    "*"
  ],
  "Condition": {
    "StringEquals": {
      "aws:ResourceTag/emr-container:endpoint:managed-certificate": "true"
    }
  },
  "Sid": "AllowACMDeletecertificate"
}
]

```

}

Anda harus mengonfigurasi izin untuk mengizinkan entitas IAM (seperti pengguna, grup, atau peran) untuk membuat, mengedit, atau menghapus peran terkait layanan. Untuk informasi lebih lanjut, lihat [Izin Peran yang Terhubung dengan Layanan](#) di Panduan Pengguna IAM.

Membuat Peran Terkait Layanan untuk Amazon EMR di EKS

Anda tidak perlu membuat peran terkait layanan secara manual. Saat Anda membuat klaster virtual, Amazon EMR di EKS membuat peran terkait layanan bagi Anda.

Jika Anda menghapus peran terkait layanan ini, lalu ingin membuatnya lagi, Anda dapat menggunakan proses yang sama untuk membuat ulang peran tersebut di akun Anda. Saat Anda membuat klaster virtual, Amazon EMR di EKS membuat peran terkait layanan bagi Anda lagi.

Anda juga dapat menggunakan konsol IAM untuk membuat peran terkait layanan dengan kasus penggunaan Amazon EMR di EKS. Di AWS CLI atau AWS API, buat peran terkait layanan dengan nama `emr-containers.amazonaws.com` layanan. Untuk informasi lebih lanjut, lihat [Membuat Peran yang Terhubung dengan Layanan](#) di Panduan Pengguna IAM. Jika Anda menghapus peran yang terhubung dengan layanan ini, Anda dapat menggunakan proses yang sama untuk membuat ulang peran tersebut.

Mengedit peran terkait layanan untuk Amazon EMR di EKS

Amazon EMR di EKS tidak mengizinkan Anda untuk mengedit peran terkait layanan `AWSServiceRoleForAmazonEMRContainers`. Setelah membuat peran terkait layanan, Anda tidak dapat mengubah nama peran karena berbagai entitas mungkin mereferensikan peran tersebut. Namun, Anda dapat mengedit penjelasan peran menggunakan IAM. Untuk informasi lebih lanjut, lihat [Mengedit Peran yang Terhubung dengan Layanan](#) di Panduan Pengguna IAM.

Menghapus peran terkait layanan untuk Amazon EMR di EKS

Jika Anda tidak perlu lagi menggunakan fitur atau layanan yang memerlukan peran terkait layanan, kami menyarankan Anda menghapus peran tersebut. Dengan begitu, Anda tidak memiliki entitas yang tidak digunakan yang tidak dipantau atau dipelihara secara aktif. Tetapi, Anda harus membersihkan sumber daya peran yang terhubung dengan layanan sebelum menghapusnya secara manual.

Note

Jika layanan Amazon EMR di EKS menggunakan peran tersebut ketika Anda mencoba menghapus sumber daya, penghapusan mungkin gagal. Jika hal itu terjadi, tunggu beberapa menit dan coba lagi.

Untuk menghapus sumber daya Amazon EMR di EKS yang digunakan oleh **AWSServiceRoleForAmazonEMRContainers**

1. Buka konsol Amazon EMR.
2. Pilih kluster virtual.
3. Pada halaman `Virtual Cluster` pilih Hapus.
4. Ulangi prosedur ini untuk setiap kluster virtual lainnya di akun Anda.

Untuk menghapus peran tertaut layanan secara manual menggunakan IAM

Gunakan konsol IAM, the AWS CLI, atau AWS API untuk menghapus peran `AWSServiceRoleForAmazonEMRContainers` terkait layanan. Untuk informasi lebih lanjut, lihat [Menghapus Peran Tertaut Layanan](#) dalam Panduan Pengguna IAM.

Wilayah yang Didukung untuk Peran Terkait Layanan Amazon EMR di EKS

Amazon EMR di EKS memberikan dukungan dengan peran terkait layanan di semua Wilayah tempat layanan tersedia. Lihat informasi yang lebih lengkap di [Amazon EMR pada titik akhir dan kuota layanan EKS](#).

Kebijakan terkelola untuk Amazon EMR di EKS

Lihat detail tentang pembaruan kebijakan AWS terkelola untuk Amazon EMR di EKS sejak 1 Maret 2021.

Ubah	Deskripsi	Date
AmazonEMRContainerServiceRolePolicy - Menambahkan izin baca untuk mencantumkan	Izin berikut ditambahkan ke kebijakan <code>:eks:ListPodIdentityAssociations</code> , <code>:eks:DescribePodIdentityAssociation</code> .	3 Februari 2023

Ubah	Deskripsi	Date
<p>kan asosiasi identitas pod EKS dalam sebuah klaster, dan izin baca lainnya untuk mengembalikan informasi deskriptif tentang asosiasi identitas pod di klaster. Untuk informasi selengkapnya, lihat Amazon EMRContainers ServiceRolePolicy.</p>		
<p>AmazonEMRContainersServiceRolePolicy - Menambahkan izin untuk mendeskripsikan dan mencantumkan nodegroup Amazon EKS, mendeskripsikan grup target penyeimbang beban, dan menjelaskan kesehatan target penyeimbang beban.</p>	<p>Izin berikut ditambahkan ke kebijakan:eks:ListNodeGroups , eks:DescribeNodeGroup elasticloadbalancing:DescribeTargetGroups ,elasticloadbalancing:DescribeTargetHealth .</p>	<p>13 Maret 2023</p>
<p>AmazonEMRContainersServiceRolePolicy - Menambahkan izin untuk mengimpor dan menghapus sertifikat di AWS Certificate Manager.</p>	<p>Izin berikut ditambahkan ke kebijakan:acm:ImportCertificate ,acm:AddTagsToCertificate ,acm>DeleteCertificate .</p>	<p>Desember 3, 2021</p>
<p>Amazon EMR di EKS mulai melacak perubahan</p>	<p>Amazon EMR di EKS mulai melacak perubahan untuk kebijakan yang AWS dikelola.</p>	<p>1 Maret 2021</p>

Menggunakan peran eksekusi tugas dengan Amazon EMR di EKS

Untuk menggunakan perintah `StartJobRun` untuk mengirimkan tugas berjalan di kluster EKS, Anda harus terlebih dahulu mengorientasi peran eksekusi tugas yang akan digunakan dengan kluster virtual. Untuk informasi selengkapnya, lihat [Untuk membuat peran eksekusi tugas](#) di [Menyiapkan Amazon EMR di EKS](#). Anda juga dapat mengikuti petunjuk di bagian [Buat Peran IAM untuk pelaksanaan pekerjaan](#) di Amazon EMR di EKS Workshop.

Izin berikut harus disertakan dalam kebijakan kepercayaan untuk peran eksekusi tugas.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::AWS_ACCOUNT_ID:oidc-provider/OIDC_PROVIDER"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "OIDC_PROVIDER:aud": "sts.amazonaws.com"
        },
        "StringLike": {
          "OIDC_PROVIDER:sub": "system:serviceaccount:NAMESPACE:emr-containers-
sa-*-*AWS_ACCOUNT_ID-BASE36_ENCODED_ROLE_NAME"
        }
      }
    }
  ]
}
```

Kebijakan kepercayaan pada contoh sebelumnya hanya memberikan izin ke akun layanan Kubernetes yang dikelola EMR Amazon dengan nama yang cocok dengan polanya. `emr-containers-sa-*-*AWS_ACCOUNT_ID-BASE36_ENCODED_ROLE_NAME` Akun layanan dengan pola ini akan dibuat secara otomatis pada pengiriman pekerjaan, dan dicakup ke namespace tempat Anda mengirimkan pekerjaan. Kebijakan kepercayaan ini memungkinkan akun layanan ini untuk mengasumsikan peran eksekusi dan mendapatkan kredensial sementara peran eksekusi. Akun

layanan dari kluster Amazon EKS yang berbeda atau dari namespace yang berbeda dalam kluster EKS yang sama dibatasi untuk mengasumsikan peran eksekusi.

Anda dapat menjalankan perintah berikut untuk memperbarui kebijakan kepercayaan secara otomatis dalam format yang diberikan di atas.

```
aws emr-containers update-role-trust-policy \  
  --cluster-name cluster \  
  --namespace namespace \  
  --role-name iam_role_name_for_job_execution
```

Mengontrol akses ke peran eksekusi

Administrator untuk kluster Amazon EKS Anda dapat membuat EMR Amazon multi-tenant di kluster virtual EKS tempat administrator IAM dapat menambahkan beberapa peran eksekusi. Karena penyewa yang tidak tepercaya dapat menggunakan peran eksekusi ini untuk mengirimkan pekerjaan yang menjalankan kode arbitrer, Anda mungkin ingin membatasi penyewa tersebut sehingga mereka tidak dapat menjalankan kode yang mendapatkan izin yang ditetapkan ke satu atau beberapa peran eksekusi ini. Untuk membatasi kebijakan IAM yang dilampirkan pada identitas IAM, administrator IAM dapat menggunakan kunci kondisi Nama Sumber Daya Amazon (ARN) opsional. `emr-containers:ExecutionRoleArn` Kondisi ini menerima daftar peran eksekusi ARNs yang memiliki izin ke kluster virtual, seperti yang ditunjukkan oleh kebijakan izin berikut.

JSON

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "emr-containers:StartJobRun"  
      ],  
      "Resource": [  
        "arn:aws:emr-containers:*:*:/virtualclusters/VIRTUAL_CLUSTER_ID"  
      ],  
      "Condition": {  
        "ArnEquals": {  
          "emr-containers:ExecutionRoleArn": [  
            "arn:aws:iam:*:*:role/execution_role_name_1",
```

```
        "arn:aws:iam::*:role/execution_role_name_2"
      ]
    }
  },
  "Sid": "AllowEMRCONTAINERSStartjobrun"
}
]
```

Jika Anda ingin mengizinkan semua peran eksekusi yang dimulai dengan awalan tertentu, seperti `MyRole`, Anda dapat mengganti operator kondisi `ArnEquals` dengan `ArnLike` operator, dan Anda dapat mengganti `execution_role_arn` nilai dalam kondisi dengan karakter wildcard*. Misalnya, `arn:aws:iam::AWS_ACCOUNT_ID:role/MyRole*`. Semua [kunci kondisi ARN](#) lainnya juga didukung.

Note

Dengan Amazon EMR di EKS, Anda tidak dapat memberikan izin untuk peran eksekusi berdasarkan tag atau atribut. Amazon EMR di EKS tidak mendukung kontrol akses berbasis tag (TBAC) atau kontrol akses berbasis atribut (ABAC) untuk peran eksekusi.

Contoh kebijakan berbasis identitas untuk Amazon EMR di EKS

Secara default, pengguna dan peran tidak memiliki izin untuk membuat atau memodifikasi Amazon EMR pada sumber daya EKS. 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 Amazon EMR di EKS, termasuk format ARNs untuk setiap jenis sumber daya, lihat [Kunci tindakan, sumber daya, dan kondisi untuk Amazon EMR di EKS](#) dalam Referensi Otorisasi Layanan.

Topik

- [Praktik terbaik kebijakan](#)
- [Menggunakan konsol Amazon EMR di EKS](#)

- [Mengizinkan pengguna melihat izin mereka sendiri](#)

Praktik terbaik kebijakan

Kebijakan berbasis identitas menentukan apakah seseorang dapat membuat, mengakses, atau menghapus Amazon EMR pada sumber daya EKS 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 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 Amazon EMR di EKS

Untuk mengakses konsol Amazon EMR di EKS, Anda harus memiliki set izin minimum. Izin ini harus memungkinkan Anda untuk membuat daftar dan melihat detail tentang Amazon EMR pada sumber daya EKS 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 EMR Amazon di konsol EKS, lampirkan juga EMR Amazon di EKS ConsoleAccess atau kebijakan ReadOnly AWS terkelola ke entitas. Untuk informasi selengkapnya, lihat [Menambahkan izin ke 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": "*"
  }
]
```

Kebijakan untuk kendali akses berbasis tanda

Anda dapat menggunakan kondisi di kebijakan berbasis identitas Anda untuk mengontrol akses ke kluster virtual dan tugas berjalan berbasis tanda. Untuk informasi lebih lanjut tentang penandaan, lihat [Menandai Sumber Daya Amazon EMR di EKS Anda](#).

Contoh berikut menunjukkan skenario yang berbeda dan cara untuk menggunakan operator syarat dengan kunci syarat Amazon EMR di EKS. Pernyataan kebijakan IAM ini dimaksudkan untuk tujuan demonstrasi saja dan tidak boleh digunakan di lingkungan produksi. Ada beberapa cara untuk menggabungkan pernyataan kebijakan untuk memberikan dan menolak izin sesuai dengan kebutuhan Anda. Untuk informasi selengkapnya tentang perencanaan dan pengujian kebijakan IAM, lihat [Panduan pengguna IAM](#).

Important

Secara eksplisit menolak izin untuk tindakan penandaan adalah pertimbangan penting. Hal ini mencegah pengguna dari penandaan sumber daya dan dengan demikian memberikan sendiri izin yang tidak ingin Anda berikan. Jika tindakan penandaan untuk sumber daya tidak ditolak, pengguna dapat memodifikasi tanda dan menghindari maksud kebijakan berbasis tanda. Untuk contoh kebijakan yang menolak tindakan penandaan, lihat [Tolak akses untuk menambah dan menghapus tanda](#).

Contoh di bawah menunjukkan kebijakan izin berbasis identitas yang digunakan untuk mengontrol tindakan yang diizinkan dengan klaster virtual Amazon EMR di EKS.

Izinkan tindakan hanya pada sumber daya dengan nilai tanda tertentu

Dalam contoh kebijakan berikut, operator `StringEquals` kondisi mencoba mencocokkan `dev` dengan nilai untuk departemen tag. Jika departemen tanda belum ditambahkan ke klaster virtual, atau tidak mengandung `dev` nilai, kebijakan tersebut tidak berlaku, dan tindakan tersebut tidak diizinkan oleh kebijakan ini. Jika tidak ada pernyataan kebijakan lain mengizinkan tindakan, pengguna hanya dapat bekerja dengan klaster virtual yang memiliki tanda ini dengan nilai ini.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:DescribeVirtualCluster"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/department": "dev"
        }
      },
      "Sid": "AllowEMRCONTAINERDescribevirtualcluster"
    }
  ]
}
```

```

    }
  ]
}

```

Anda juga dapat menentukan beberapa nilai tanda menggunakan operator syarat. Misalnya, untuk mengizinkan tindakan pada kluster virtual di mana tanda department berisi nilai dev atau test, Anda bisa mengganti blok syarat di contoh sebelumnya dengan berikut ini.

```

"Condition": {
  "StringEquals": {
    "aws:ResourceTag/department": ["dev", "test"]
  }
}

```

Memerlukan penandaan ketika sumber daya dibuat

Pada contoh di bawah ini, tanda perlu diterapkan saat membuat kluster virtual.

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:CreateVirtualCluster"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/department": "dev"
        }
      },
      "Sid": "AllowEMRCONTAINERSCreatevirtualcluster"
    }
  ]
}

```

Pernyataan kebijakan berikut mengizinkan pengguna untuk membuat klaster virtual hanya jika klaster memiliki tanda `department`, yang dapat berisi nilai apa pun.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:CreateVirtualCluster"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "Null": {
          "aws:RequestTag/department": "false"
        }
      },
      "Sid": "AllowEMRCONTAINERSCreatevirtualcluster"
    }
  ]
}
```

Tolak akses untuk menambah dan menghapus tanda

Efek dari kebijakan ini adalah untuk menolak izin pengguna untuk menambah atau menghapus tanda apa pun pada klaster virtual yang ditandai dengan tanda `department` yang berisi nilai `dev`.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "emr-containers:TagResource",

```

```

    "emr-containers:UntagResource"
  ],
  "Resource": [
    "*"
  ],
  "Condition": {
    "StringNotEquals": {
      "aws:ResourceTag/department": "dev"
    }
  },
  "Sid": "AllowEMRCONTAINERSTagresource"
}
]
}

```

Memecahkan Masalah identitas dan akses Amazon EMR di EKS

Gunakan informasi berikut untuk membantu Anda mendiagnosis dan mengatasi masalah umum yang mungkin Anda temui saat bekerja menggunakan Amazon EMR di EKS dan IAM.

Topik

- [Saya tidak berwenang untuk melakukan tindakan di Amazon EMR di EKS](#)
- [Saya tidak berwenang untuk melakukan iam: PassRole](#)
- [Saya ingin mengizinkan orang di luar AWS akun saya untuk mengakses EMR Amazon saya di sumber daya EKS](#)

Saya tidak berwenang untuk melakukan tindakan di Amazon EMR di EKS

Jika Konsol Manajemen AWS memberitahu Anda bahwa Anda tidak berwenang untuk melakukan tindakan, maka Anda harus menghubungi administrator Anda untuk bantuan. Administrator Anda adalah orang yang memberikan nama pengguna dan kata sandi Anda.

Contoh kesalahan berikut terjadi ketika pengguna `mateojackson` mencoba menggunakan konsol untuk melihat detail tentang suatu sumber daya `my-example-widget` fiktif, tetapi tidak memiliki izin `emr-containers:GetWidget` fiktif.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform: emr-containers:GetWidget on resource: my-example-widget
```

Dalam hal ini, Mateo meminta administratornya untuk memperbarui kebijakannya untuk mengizinkan dia mengakses sumber daya *my-example-widget* menggunakan tindakan `emr-containers:GetWidget`.

Saya tidak berwenang untuk melakukan `iam:PassRole`

Jika Anda menerima kesalahan bahwa Anda tidak diizinkan untuk melakukan `iam:PassRole` tindakan, kebijakan Anda harus diperbarui agar Anda dapat meneruskan peran ke Amazon EMR di EKS.

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 Amazon EMR di EKS. 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 AWS akun saya untuk mengakses EMR Amazon saya di sumber daya EKS

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 (ACLs), Anda dapat menggunakan kebijakan tersebut untuk memberi orang akses ke sumber daya Anda.

Untuk mempelajari selengkapnya, periksa referensi berikut:

- Untuk mempelajari apakah Amazon EMR di EKS mendukung fitur ini, lihat [Bagaimana Amazon EMR di EKS 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](#).

Menggunakan Amazon EMR di EKS dengan AWS Lake Formation untuk kontrol akses berbutir halus

Dengan Amazon EMR rilis 7.7 dan yang lebih tinggi, Anda dapat memanfaatkan AWS Lake Formation untuk menerapkan kontrol akses berbutir halus pada tabel Katalog Data AWS Glue yang didukung oleh bucket Amazon S3. Kemampuan ini memungkinkan Anda mengonfigurasi tabel, baris, kolom, dan kontrol akses tingkat sel untuk kueri baca dalam EMR Amazon Anda di EKS Spark Jobs.

Topik

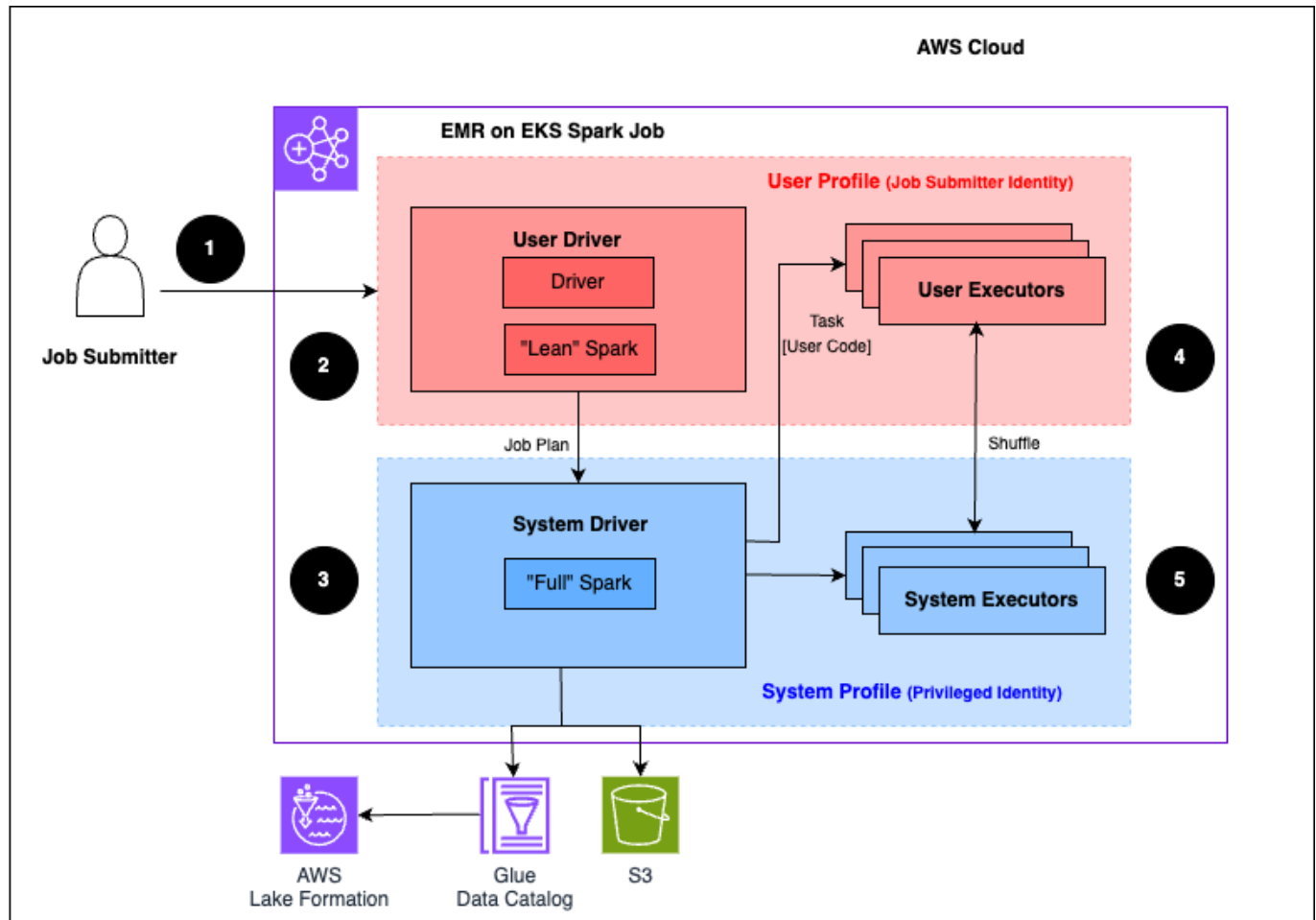
- [Bagaimana Amazon EMR di EKS bekerja dengan Lake Formation AWS](#)
- [Aktifkan Lake Formation dengan Amazon EMR di EKS](#)
- [Pertimbangan dan batasan](#)
- [Pemecahan masalah](#)

Bagaimana Amazon EMR di EKS bekerja dengan Lake Formation AWS

Menggunakan Amazon EMR di EKS dengan Lake Formation memungkinkan Anda menerapkan lapisan izin pada setiap Spark Job untuk menerapkan kontrol izin Lake Formation saat Amazon EMR di EKS menjalankan pekerjaan. Amazon EMR di EKS menggunakan [profil sumber daya Spark untuk membuat dua profil](#) untuk menjalankan pekerjaan secara efektif. Profil Pengguna mengeksekusi kode yang disediakan pengguna, sementara profil sistem memberlakukan kebijakan

Lake Formation. Setiap Lake Formation mengaktifkan Job menggunakan dua driver Spark, satu untuk profil Pengguna, dan satu lagi untuk profil Sistem. Untuk informasi lebih lanjut, lihat [Apa itu AWS Lake Formation](#).

Berikut ini adalah ikhtisar tingkat tinggi tentang bagaimana Amazon EMR di EKS mendapatkan akses ke data yang dilindungi oleh kebijakan keamanan Lake Formation.



Langkah-langkah berikut menjelaskan proses ini:

1. Pengguna mengirimkan Spark Job ke Amazon EMR berkemampuan AWS Lake Formation di cluster virtual EKS.
2. Amazon EMR pada layanan EKS menyiapkan Driver Pengguna dan menjalankan pekerjaan di Profil Pengguna. Driver Pengguna menjalankan versi lean Spark yang tidak memiliki kemampuan untuk meluncurkan tugas, meminta pelaksana, mengakses Amazon S3 atau Katalog Data Glue. Itu hanya membangun rencana Job.
3. Amazon EMR pada layanan EKS menyiapkan driver kedua yang disebut System Driver dan menjalankannya di Profil Sistem (dengan identitas istimewa). Amazon EKS menyiapkan saluran

TLS terenkripsi antara dua driver untuk komunikasi. Pengemudi Pengguna menggunakan saluran untuk mengirim rencana pekerjaan ke Driver Sistem. Driver Sistem tidak menjalankan kode yang dikirimkan pengguna. Ini menjalankan Spark penuh dan berkomunikasi dengan Amazon S3 dan Katalog Data untuk akses data. Ini meminta pelaksana dan mengkompilasi Job Plan ke dalam urutan tahapan eksekusi.

4. Amazon EMR pada layanan EKS kemudian menjalankan tahapan pada pelaksana. Kode Pengguna dalam tahap apa pun dijalankan secara eksklusif pada pelaksana profil Pengguna.
5. Tahapan yang membaca data dari tabel Katalog Data yang dilindungi oleh Lake Formation atau yang menerapkan filter keamanan didelegasikan ke pelaksana Sistem.

Aktifkan Lake Formation dengan Amazon EMR di EKS

Dengan Amazon EMR rilis 7.7 dan yang lebih tinggi, Anda dapat memanfaatkan AWS Lake Formation untuk menerapkan kontrol akses berbutir halus pada tabel Katalog Data yang didukung oleh Amazon S3. Kemampuan ini memungkinkan Anda mengonfigurasi kontrol akses tingkat tabel, baris, kolom, dan sel untuk kueri baca dalam EMR Amazon Anda di EKS Spark Jobs.

Bagian ini mencakup cara membuat konfigurasi keamanan dan mengatur Lake Formation agar berfungsi dengan Amazon EMR. Ini juga menjelaskan cara membuat cluster virtual dengan Konfigurasi Keamanan yang Anda buat untuk Lake Formation. Bagian-bagian ini dimaksudkan untuk diselesaikan secara berurutan.

Langkah 1: Siapkan izin kolom, baris, atau tingkat sel berbasis Lake Formation

Pertama, untuk menerapkan izin tingkat baris dan kolom dengan Lake Formation, administrator danau data untuk Lake Formation harus mengatur Tag LakeFormationAuthorizedCallerSesi. Lake Formation menggunakan tag sesi ini untuk mengotorisasi penelepon dan menyediakan akses ke danau data.

Arahkan ke konsol AWS Lake Formation dan pilih opsi Pengaturan integrasi aplikasi dari bagian Administrasi di sidebar. Kemudian, centang kotak Izinkan mesin eksternal memfilter data di lokasi Amazon S3 yang terdaftar di Lake Formation. Tambahkan AWS Akun IDs tempat Spark Jobs akan berjalan, dan tag Session Values.

Application integration settings [Learn more](#)

Application integration settings

Use the options below to control which third-party engines are allowed to read and filter data in Amazon S3 locations registered with Lake Formation.

Allow external engines to filter data in Amazon S3 locations registered with Lake Formation

Check this box to allow third-party engines to access data in Amazon S3 locations that are registered with Lake Formation.

Session tag values

Enter one or more strings that match the LakeFormationAuthorizedCaller session tag defined for third-party engines.

Clear all

EMR on EKS Engine X

Enter one or several string values separated by comma.

AWS account IDs

Enter the external AWS account IDs from where third-party engines are allowed to access locations registered with Lake Formation.

Clear all

012345678901 X

Account

Enter one or more AWS account IDs. Press enter after each ID.

Allow external engines to access data in Amazon S3 locations with full table access

When you enable this option, Lake Formation will return credentials to the integrated application directly without IAM session tag validation.

Cancel

Save

Perhatikan bahwa Tag `LakeFormationAuthorizedCallerSesi` yang diteruskan di sini diteruskan `SecurityConfiguration` nanti saat Anda mengatur peran IAM, di bagian 3.

Langkah 2: Siapkan izin EKS RBAC

Kedua, Anda mengatur izin untuk kontrol akses berbasis peran.

Berikan Izin Kluster EKS ke EMR Amazon di layanan EKS

EMR Amazon di Layanan EKS harus memiliki izin Peran Kluster EKS sehingga dapat membuat izin namespace silang untuk Driver Sistem untuk memutar pelaksana Pengguna di namespace Pengguna.

Buat Peran Cluster

Contoh ini mendefinisikan izin untuk kumpulan sumber daya.

```
vim emr-containers-cluster-role.yaml
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: emr-containers
rules:
  - apiGroups: [""]
    resources: ["namespaces"]
    verbs: ["get"]
  - apiGroups: [""]
    resources: ["serviceaccounts", "services", "configmaps", "events", "pods", "pods/
log"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"deletecollection", "annotate", "patch", "label"]
  - apiGroups: [""]
    resources: ["secrets"]
    verbs: ["create", "patch", "delete", "watch"]
  - apiGroups: ["apps"]
    resources: ["statefulsets", "deployments"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete", "annotate",
"patch", "label"]
  - apiGroups: ["batch"]
    resources: ["jobs"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete", "annotate",
"patch", "label"]
  - apiGroups: ["extensions", "networking.k8s.io"]
    resources: ["ingresses"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete", "annotate",
"patch", "label"]
  - apiGroups: ["rbac.authorization.k8s.io"]
    resources: ["clusterroles", "clusterrolebindings", "roles", "rolebindings"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"deletecollection", "annotate", "patch", "label"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"deletecollection", "annotate", "patch", "label"]
  - apiGroups: ["kyverno.io"]
    resources: ["clusterpolicies"]
    verbs: ["create", "delete"]
---
```

```
kubectl apply -f emr-containers-cluster-role.yaml
```

Buat Binding Peran Cluster

```
vim emr-containers-cluster-role-binding.yaml
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: emr-containers
subjects:
- kind: User
  name: emr-containers
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: emr-containers
  apiGroup: rbac.authorization.k8s.io
---
```

```
kubectl apply -f emr-containers-cluster-role-binding.yaml
```

Menyediakan akses Namespace ke EMR Amazon pada layanan EKS

Buat dua ruang nama Kubernetes, satu untuk driver Pengguna dan pelaksana, dan satu lagi untuk driver & pelaksana Sistem, dan aktifkan Amazon EMR pada akses layanan EKS untuk mengirimkan Pekerjaan di Ruang Nama Pengguna dan Sistem. Ikuti panduan yang ada untuk menyediakan akses untuk setiap namespace, yang tersedia di [Aktifkan akses cluster menggunakan aws-auth](#)

Langkah 3: Siapkan Peran IAM untuk komponen profil pengguna dan sistem

Ketiga, Anda mengatur peran untuk komponen tertentu. Spark Job yang mendukung Lake Formation memiliki dua komponen, User dan System. Driver Pengguna dan pelaksana berjalan di namespace Pengguna, dan terikat dengan yang diteruskan di JobExecutionRole API. StartJobRun Driver dan pelaksana Sistem berjalan di namespace Sistem, dan terikat pada peran. QueryEngine

Konfigurasi peran Query Engine

QueryEngine Peran terkait dengan Komponen Ruang Sistem, dan akan memiliki izin untuk mengasumsikan tag JobExecutionRolewith LakeFormationAuthorizedCallerSession. Kebijakan Izin IAM untuk peran Mesin Kueri adalah sebagai berikut:

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AssumeJobRoleWithSessionTagAccessForSystemDriver",
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ],
      "Resource": [
        "arn:aws:iam::*:role/JobExecutionRole"
      ],
      "Condition": {
        "StringLike": {
          "aws:RequestTag/LakeFormationAuthorizedCaller": "EMR on EKS Engine"
        }
      }
    },
    {
      "Sid": "AssumeJobRoleWithSessionTagAccessForSystemExecutor",
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole"
      ],
      "Resource": [
        "arn:aws:iam::*:role/JobExecutionRole"
      ]
    },
    {
      "Sid": "CreateCertificateAccessForTLS",
      "Effect": "Allow",
      "Action": [
        "emr-containers:CreateCertificate"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}

```

```
}
```

Konfigurasi kebijakan Trust dari peran Query Engine untuk mempercayai namespace Sistem Kubernetes.

```
aws emr-containers update-role-trust-policy \  
  --cluster-name eks cluster \  
  --namespace eks system namespace \  
  --role-name query_engine_iam_role_name
```

Untuk informasi selengkapnya, lihat [Memperbarui kebijakan kepercayaan peran](#).

Konfigurasi Peran Eksekusi Job

Izin Lake Formation mengontrol akses ke sumber daya Katalog Data AWS Glue, lokasi Amazon S3, dan data dasar di lokasi tersebut. Izin IAM mengontrol akses ke Lake Formation dan AWS Glue APIs dan sumber daya. Meskipun Anda mungkin memiliki izin Lake Formation untuk mengakses tabel di Katalog Data (SELECT), operasi Anda gagal jika Anda tidak memiliki izin IAM pada operasi `glue:Get*` API.

Kebijakan Izin IAM dari `JobExecutionRole: JobExecutionPeran` harus memiliki Pernyataan Kebijakan dalam Kebijakan Izinnya.

JSON

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "GlueCatalogAccess",  
      "Effect": "Allow",  
      "Action": [  
        "glue:Get*",  
        "glue:Create*",  
        "glue:Update*"  
      ],  
      "Resource": [  
        "*"  
      ]  
    }  
  ]  
}
```

```

    },
    {
      "Sid": "LakeFormationAccess",
      "Effect": "Allow",
      "Action": [
        "lakeformation:GetDataAccess"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "CreateCertificateAccessForTLS",
      "Effect": "Allow",
      "Action": [
        "emr-containers:CreateCertificate"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}

```

Kebijakan Kepercayaan IAM untuk JobExecutionRole:

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "TrustQueryEngineRoleForSystemDriver",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:role/QueryExecutionRole"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ],
      "Condition": {

```

```

    "StringLike": {
      "aws:RequestTag/LakeFormationAuthorizedCaller": "EMR on EKS Engine"
    }
  },
  {
    "Sid": "TrustQueryEngineRoleForSystemExecutor",
    "Effect": "Allow",
    "Principal": {
      "AWS": "arn:aws:iam::123456789012:role/QueryEngineRole"
    },
    "Action": "sts:AssumeRole"
  }
]
}

```

Konfigurasi Kebijakan Kepercayaan Peran eksekusi Job untuk mempercayai namespace pengguna Kubernetes:

```

aws emr-containers update-role-trust-policy \
  --cluster-name eks cluster \
  --namespace eks User namespace \
  --role-name job_execution_role_name

```

Untuk informasi selengkapnya, lihat [Memperbarui kebijakan kepercayaan peran pelaksanaan pekerjaan](#).

Langkah 4: Pengaturan konfigurasi keamanan

Untuk menjalankan pekerjaan yang mendukung Lake Formation, Anda harus membuat konfigurasi keamanan.

```

aws emr-containers create-security-configuration \
  --name 'security-configuration-name' \
  --security-configuration '{
    "authorizationConfiguration": {
      "lakeFormationConfiguration": {
        "authorizedSessionTagValue": "SessionTag configured in LakeFormation",
        "secureNamespaceInfo": {
          "clusterId": "eks-cluster-name",
          "namespace": "system-namespace-name"
        }
      }
    }
  }'

```

```

    },
    "queryEngineRoleArn": "query-engine-IAM-role-ARN"
  }
}
}'

```

Pastikan bahwa Tag Sesi yang diteruskan di bidang `authorizedSessionTagNilai` dapat mengotorisasi Lake Formation. Tetapkan nilai ke yang dikonfigurasi di Lake Formation, di [Langkah 1: Siapkan izin kolom, baris, atau tingkat sel berbasis Lake Formation](#).

Langkah 5: Buat cluster virtual

Buat EMR Amazon di kluster virtual EKS dengan konfigurasi keamanan.

```

aws emr-containers create-virtual-cluster \
--name my-lf-enabled-vc \
--container-provider '{
  "id": "eks-cluster",
  "type": "EKS",
  "info": {
    "eksInfo": {
      "namespace": "user-namespace"
    }
  }
}' \
--security-configuration-id SecurityConfiguraionId

```

Pastikan `SecurityConfigurationId` dari langkah sebelumnya diteruskan, sehingga konfigurasi otorisasi Lake Formation diterapkan ke semua Pekerjaan yang berjalan di cluster virtual. Untuk informasi selengkapnya, lihat [Mendaftarkan klaster Amazon EKS dengan Amazon EMR](#).

Langkah 6: Kirim Job di FGAC Enabled VirtualCluster

Process for Job Submission sama untuk pekerjaan non Lake Formation dan Lake Formation. Untuk informasi selengkapnya, lihat [Mengirimkan pekerjaan yang dijalankan dengan StartJobRun](#).

Spark Driver, Executor, dan Event Logs dari System Driver disimpan di Bucket S3 Akun AWS Layanan untuk debugging. Sebaiknya konfigurasi Kunci KMS yang dikelola pelanggan di Job Run untuk mengenkripsi semua log yang disimpan dalam bucket layanan. AWS Untuk informasi selengkapnya tentang mengaktifkan enkripsi log, lihat Mengenkripsi [EMR Amazon di log EKS](#).

Pertimbangan dan batasan

Perhatikan pertimbangan dan batasan berikut saat Anda menggunakan Lake Formation dengan Amazon EMR di EKS:

- Amazon EMR di EKS mendukung kontrol akses berbutir halus melalui Lake Formation hanya untuk Apache Hive, Apache Iceberg, Apache Hudi, dan Format tabel Delta. Format Apache Hive termasuk Parquet, ORC, dan xSV.
- `DynamicResourceAllocation` diaktifkan secara default, dan Anda tidak dapat mematikan `DynamicResourceAllocation` untuk pekerjaan Lake Formation. Karena nilai default `spark.dynamicAllocation.maxExecutors` konfigurasi DRA adalah tak terbatas, harap konfigurasi nilai yang sesuai berdasarkan beban kerja Anda.
- Secara default, `spark.dynamicAllocation.preallocateExecutors` diaktifkan di Amazon EMR Spark, yang dapat menyebabkan churn kontainer berlebihan saat `spark.dynamicAllocation.initialExecutors` dan `spark.dynamicAllocation.minExecutors` tidak disetel. Untuk konfigurasi yang disarankan untuk mengelola prealokasi pelaksana, lihat bagian di [Performa Tautan ke Amazon EMR pada panduan praktik terbaik EKS GitHub](#)
- Pekerjaan berkemampuan Lake Formation tidak mendukung penggunaan EMR khusus pada Gambar EKS di System Driver dan System Executors.
- Anda hanya dapat menggunakan Lake Formation dengan pekerjaan Spark.
- EMR di EKS dengan Lake Formation hanya mendukung satu sesi Spark selama pekerjaan.
- EMR di EKS dengan Lake Formation hanya mendukung kueri tabel lintas akun yang dibagikan melalui tautan sumber daya.
- Berikut ini tidak didukung:
 - Kumpulan data terdistribusi yang tangguh (RDD)
 - Streaming percikan
 - Menulis dengan izin yang diberikan Lake Formation
 - Kontrol akses untuk kolom bersarang
- EMR pada EKS memblokir fungsionalitas yang mungkin merusak isolasi lengkap driver sistem, termasuk yang berikut:
 - UDTs, HiveUDFs, dan fungsi apa pun yang ditentukan pengguna yang melibatkan kelas khusus
 - Sumber data kustom

- Pasokan stoples tambahan untuk ekstensi Spark, konektor, atau perintah metastore ANALYZE TABLE
- Untuk menegakkan kontrol akses, EXPLAIN PLAN dan operasi DDL seperti DESCRIBE TABLE tidak mengekspos informasi terbatas.
- Amazon EMR di EKS membatasi akses ke driver sistem Spark log pada pekerjaan yang mendukung Lake Formation. Karena driver sistem berjalan dengan lebih banyak akses, peristiwa dan log yang dihasilkan driver sistem dapat mencakup informasi sensitif. Untuk mencegah pengguna atau kode yang tidak sah mengakses data sensitif ini, EMR di EKS menonaktifkan akses ke log driver sistem. Untuk pemecahan masalah, hubungi AWS dukungan.
- Jika Anda mendaftarkan lokasi tabel dengan Lake Formation, jalur akses data melewati kredensial yang disimpan Lake Formation, terlepas dari izin IAM untuk EMR pada peran eksekusi pekerjaan EKS. Jika Anda salah mengonfigurasi peran yang terdaftar dengan lokasi tabel, pekerjaan yang dikirimkan yang menggunakan peran dengan izin IAM S3 ke lokasi tabel akan gagal.
- Menulis ke tabel Lake Formation menggunakan izin IAM daripada izin yang diberikan Lake Formation. Jika peran eksekusi pekerjaan Anda memiliki izin S3 yang diperlukan, Anda dapat menggunakannya untuk menjalankan operasi penulisan.

Berikut ini adalah pertimbangan dan batasan saat menggunakan Apache Iceberg:

- Anda hanya dapat menggunakan Apache Iceberg dengan katalog sesi dan tidak sewenang-wenang bernama katalog.
- Tabel gunung es yang terdaftar di Lake Formation hanya mendukung tabel metadatahistory,,, metadata_log_entriesnapshots, files dan manifests refs Amazon EMR menyembunyikan kolom yang mungkin memiliki data sensitif, seperti partitions,, dan path summaries Batasan ini tidak berlaku untuk tabel Gunung Es yang tidak terdaftar di Lake Formation.
- Tabel yang tidak Anda daftarkan di Lake Formation mendukung semua prosedur yang disimpan Gunung Es. Prosedur register_table dan migrate prosedur tidak didukung untuk tabel apa pun.
- Kami menyarankan Anda menggunakan Iceberg DataFrameWriter V2 alih-alih V1.

Untuk informasi selengkapnya, lihat [Memahami EMR Amazon tentang konsep dan terminologi EKS dan Aktifkan akses kluster untuk Amazon EMR](#) di EKS.

Penafian untuk administrator data

Note

Saat Anda memberikan akses ke sumber daya Lake Formation ke peran IAM untuk EMR di EKS, Anda harus memastikan administrator atau operator kluster EMR adalah administrator tepercaya. Ini sangat relevan untuk sumber daya Lake Formation yang dibagikan di berbagai organisasi dan AWS akun.

Tanggung jawab untuk administrator EKS

- SystemNamespace harus dilindungi. Tidak ada pengguna atau sumber daya atau entitas atau perkakas yang diizinkan untuk memiliki izin RBAC Kubernetes pada sumber daya Kubernetes di namespace. System
- Tidak ada pengguna atau sumber daya atau entitas kecuali EMR pada layanan EKS yang memiliki CREATE akses ke POD, CONFIG_MAP, dan SECRET di namespace. User
- Systemdriver dan System pelaksana berisi data sensitif. Jadi, peristiwa Spark, log driver Spark, dan log pelaksana Spark di System namespace tidak boleh diteruskan ke sistem penyimpanan log eksternal.

Pemecahan masalah

Pencatatan log

EMR di EKS menggunakan profil sumber daya Spark untuk membagi eksekusi pekerjaan. Amazon EMR di EKS menggunakan profil pengguna untuk menjalankan kode yang Anda berikan, sementara profil sistem memberlakukan kebijakan Lake Formation. Anda dapat mengakses log untuk wadah yang dijalankan sebagai profil pengguna dengan mengonfigurasi StartJobRun permintaan.

[MonitoringConfiguration](#)

Spark History Server

Server Sejarah Spark memiliki semua peristiwa Spark yang dihasilkan dari profil pengguna dan peristiwa yang disunting yang dihasilkan dari driver sistem. Anda dapat melihat semua kontainer dari driver pengguna dan sistem di tab Executors. Namun, tautan log hanya tersedia untuk profil pengguna.

Job gagal dengan izin Lake Formation yang tidak mencukupi

Pastikan bahwa peran eksekusi pekerjaan Anda memiliki izin untuk dijalankan SELECT dan DESCRIBE di atas meja yang Anda akses.

Job dengan eksekusi RDD gagal

EMR di EKS saat ini tidak mendukung operasi kumpulan data terdistribusi tangguh (RDD) pada pekerjaan yang mendukung Lake Formation.

Tidak dapat mengakses file data di Amazon S3

Pastikan Anda telah mendaftarkan lokasi data lake di Lake Formation.

Pengecualian validasi keamanan

EMR di EKS mendeteksi kesalahan validasi keamanan. Hubungi AWS dukungan untuk bantuan.

Berbagi Katalog Data AWS Glue dan tabel di seluruh akun

Anda dapat berbagi database dan tabel di seluruh akun dan masih menggunakan Lake Formation. Untuk informasi selengkapnya, lihat [Berbagi data lintas akun di Lake Formation](#) dan [Bagaimana cara membagikan Katalog Data AWS Glue dan tabel lintas akun menggunakan AWS Lake Formation?](#) .

Iceberg Job melempar kesalahan inisialisasi tidak menyetel wilayah AWS

Pesannya adalah sebagai berikut:

```
25/02/25 13:33:19 ERROR SparkFGACEExceptionSanitizer: Client received error with id =
b921f9e6-f655-491f-b8bd-b2842cdc20c7,
reason = IllegalArgumentException, message = Cannot initialize
LakeFormationAwsClientFactory, please set client.region to a valid aws region
```

Pastikan konfigurasi Spark `spark.sql.catalog.catalog_name.client.region` diatur ke wilayah yang valid.

Iceberg Job melempar SparkUnsupportedOperationException

Pesannya adalah sebagai berikut:

```
25/02/25 13:53:15 ERROR SparkFGACEptionSanitizer: Client received error with id =
921fef42-0800-448b-bef5-d283d1278ce0,
reason = SparkUnsupportedOperationException, message = Either glue.id or glue.account-
id is set with non-default account.
Cross account access with fine-grained access control is only supported with AWS
Resource Access Manager.
```

Pastikan Konfigurasi Spark `spark.sql.catalog.catalog_name.glue.account-id` diatur ke id akun yang valid.

Iceberg Job gagal dengan “403 Access Denied” selama operasi MERGE

Pesannya adalah sebagai berikut:

```
software.amazon.awssdk.services.s3.model.S3Exception: Access Denied (Service: S3,
Status Code: 403,
...
at
software.amazon.awssdk.services.s3.DefaultS3Client.deleteObject(DefaultS3Client.java:3365)
at org.apache.iceberg.aws.s3.S3FileIO.deleteFile(S3FileIO.java:162)
at org.apache.iceberg.io.FileIO.deleteFile(FileIO.java:86)
at
org.apache.iceberg.io.RollingFileWriter.closeCurrentWriter(RollingFileWriter.java:129)
```

Nonaktifkan operasi Hapus S3 di Spark dengan menambahkan properti berikut. `--conf spark.sql.catalog.s3-table-name.s3.delete-enabled=false`.

Pencatatan dan pemantauan

Untuk mendeteksi insiden, menerima peringatan ketika insiden terjadi, dan menanggapi, gunakan opsi ini dengan Amazon EMR di EKS:

- Monitor Amazon EMR di EKS dengan AWS CloudTrail - [AWS CloudTrail](#) menyediakan catatan tindakan yang diambil oleh pengguna, peran, atau AWS layanan di Amazon EMR di EKS. Ini menangkap panggilan dari konsol Amazon EMR dan panggilan kode ke operasi API Amazon EMR di EKS sebagai peristiwa. Ini memungkinkan Anda untuk menentukan permintaan yang diajukan ke Amazon EMR di EKS, alamat IP dari mana permintaan dibuat, siapa yang mengajukan permintaan, kapan dibuat, dan detail tambahan. Untuk informasi selengkapnya, lihat [Logging Amazon EMR pada panggilan EKS API menggunakan AWS CloudTrail](#).

- Gunakan CloudWatch Acara dengan Amazon EMR di EKS - Acara memberikan aliran CloudWatch peristiwa sistem yang mendekati waktu nyata yang menjelaskan perubahan sumber daya. AWS CloudWatch Peristiwa menjadi sadar akan perubahan operasional saat terjadi, meresponsnya, dan mengambil tindakan korektif seperlunya, dengan mengirim pesan untuk merespons lingkungan, mengaktifkan fungsi, membuat perubahan, dan menangkap informasi negara. Untuk menggunakan CloudWatch Acara dengan Amazon EMR di EKS, buat aturan yang memicu pada EMR Amazon pada panggilan EKS API melalui CloudTrail Untuk informasi selengkapnya, lihat [Pantau pekerjaan dengan Amazon CloudWatch Events](#).

Mengenkripsi Amazon EMR pada log EKS dengan penyimpanan terkelola

Bagian berikut menunjukkan kepada Anda cara mengonfigurasi enkripsi untuk log.

Aktifkan enkripsi

Untuk mengenkripsi log dalam penyimpanan terkelola dengan kunci KMS Anda sendiri, gunakan konfigurasi berikut saat Anda mengirimkan pekerjaan.

```
"monitoringConfiguration": {
  "managedLogs": {
    "allowAWSToRetainLogs": "ENABLED",
    "encryptionKeyArn": "KMS key arn"
  },
  "persistentAppUI": "ENABLED"
}
```

`allowAWSToRetainLogs` Konfigurasi ini memungkinkan AWS untuk mempertahankan log namespace sistem saat menjalankan pekerjaan menggunakan Native FGAC.

`persistentAppUI` Konfigurasi memungkinkan AWS untuk menyimpan log peristiwa yang digunakan untuk menghasilkan UI Spark. `encryptionKeyArn` ini digunakan untuk menentukan ARN kunci KMS yang ingin Anda gunakan untuk mengenkripsi log yang disimpan oleh AWS

Izin yang diperlukan untuk enkripsi log

Pengguna yang mengirimkan pekerjaan atau melihat UI Spark harus diizinkan

tindakan `kms:DescribeKey`, `kms:GenerateDataKey`, dan `kms:Decrypt` untuk kunci enkripsi.

Izin ini digunakan untuk memverifikasi validitas kunci dan memeriksa apakah pengguna memiliki izin yang diperlukan untuk membaca dan menulis log yang dienkripsi dengan kunci KMS. Jika pengguna

yang mengirimkan pekerjaan tidak memiliki izin kunci yang diperlukan, Amazon EMR di EKS menolak pengiriman job run.

Contoh Kebijakan IAM untuk Peran yang Digunakan untuk Memanggil StartJobRun

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "emr-containers:StartJobRun"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow",
      "Sid": "AllowEMRCONTAINERSStartjobrun"
    },
    {
      "Action": [
        "kms:DescribeKey",
        "kms:Decrypt",
        "kms:GenerateDataKey"
      ],
      "Resource": [
        "arn:aws:kms:*:*:key/key-id"
      ],
      "Effect": "Allow",
      "Sid": "AllowKMSDescribekey"
    }
  ]
}
```

Anda juga harus mengkonfigurasi kunci KMS untuk mengizinkan `persistentappui.elasticmapreduce.amazonaws.com` dan Prinsipal `elasticmapreduce.amazonaws.com` Layanan untuk `kms:GenerateDataKey` `kms:Decrypt` Ini memungkinkan EMR untuk membaca dan menulis log yang dienkripsi dengan kunci KMS ke penyimpanan terkelola.

Contoh Kebijakan Kunci KMS

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:DescribeKey"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringLike": {
          "kms:viaService": "emr-containers.*.amazonaws.com"
        }
      },
      "Sid": "AllowKMSDescribekey"
    },
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringLike": {
          "kms:viaService": "emr-containers.*.amazonaws.com",
          "kms:EncryptionContext:aws:emr-containers:virtualClusterId": "virtual cluster id"
        }
      },
      "Sid": "AllowKMSDecryptGenerate"
    },
    {
      "Effect": "Allow",
      "Action": [
```

```

        "kms:Decrypt",
        "kms:GenerateDataKey"
    ],
    "Resource": [
        "*"
    ],
    "Condition": {
        "StringLike": {
            "kms:EncryptionContext:aws:emr-containers:virtualClusterId": "virtual
cluster id"
        },
        "ArnLike": {
            "aws:SourceArn": "arn:aws:emr-containers:*:*:/
virtualclusters/virtual_cluster_id"
        }
    },
    "Sid": "AllowKMSDecryptService"
}
]
}

```

Sebagai praktik keamanan terbaik, kami menyarankan Anda menambahkan `kms:viaService`, `kms:EncryptionContext`, dan `aws:SourceArn` kondisi. Kondisi ini membantu memastikan kunci hanya digunakan oleh Amazon EMR di EKS dan hanya digunakan untuk log yang dihasilkan dari pekerjaan yang berjalan di cluster virtual tertentu.

Logging Amazon EMR pada panggilan EKS API menggunakan AWS CloudTrail

Amazon EMR di EKS terintegrasi dengan AWS CloudTrail, layanan yang menyediakan catatan tindakan yang diambil oleh pengguna, peran, atau AWS layanan di Amazon EMR di EKS. CloudTrail menangkap semua panggilan API untuk Amazon EMR di EKS sebagai acara. Panggilan yang direkam mencakup panggilan dari konsol Amazon EMR di EKS dan panggilan kode ke operasi API Amazon EMR di EKS. Jika Anda membuat jejak, Anda dapat mengaktifkan pengiriman CloudTrail acara secara terus menerus ke bucket Amazon S3, termasuk acara untuk Amazon EMR di EKS. Jika Anda tidak mengonfigurasi jejak, Anda masih dapat melihat peristiwa terbaru di CloudTrail konsol dalam Riwayat acara. Dengan menggunakan informasi yang dikumpulkan oleh CloudTrail, Anda dapat menentukan permintaan yang dibuat ke Amazon EMR di EKS, alamat IP dari mana permintaan dibuat, siapa yang membuat permintaan, kapan dibuat, dan detail tambahan.

Untuk mempelajari selengkapnya CloudTrail, lihat [Panduan AWS CloudTrail Pengguna](#).

Amazon EMR pada informasi EKS di CloudTrail

CloudTrail diaktifkan di AWS akun Anda saat Anda membuat akun. Ketika aktivitas terjadi di Amazon EMR di EKS, aktivitas tersebut direkam dalam suatu CloudTrail peristiwa bersama dengan peristiwa AWS layanan lainnya dalam riwayat Acara. Anda dapat melihat, mencari, dan mengunduh acara terbaru di AWS akun Anda. Untuk informasi selengkapnya, lihat [Melihat peristiwa dengan Riwayat CloudTrail acara](#).

Untuk catatan peristiwa yang sedang berlangsung di AWS akun Anda, termasuk acara untuk Amazon EMR di EKS, buat jejak. Jejak memungkinkan CloudTrail untuk mengirimkan file log ke bucket Amazon S3. Secara default, saat Anda membuat jejak di konsol, jejak tersebut berlaku untuk semua AWS Wilayah. Jejak mencatat peristiwa dari semua Wilayah di AWS partisi dan mengirimkan file log ke bucket Amazon S3 yang Anda tentukan. Selain itu, Anda dapat mengonfigurasi AWS layanan lain untuk menganalisis lebih lanjut dan menindaklanjuti data peristiwa yang dikumpulkan dalam CloudTrail log. Untuk informasi selengkapnya, lihat berikut:

- [Gambaran umum untuk membuat jejak](#)
- [CloudTrail layanan dan integrasi yang didukung](#)
- [Mengonfigurasi notifikasi Amazon SNS untuk CloudTrail](#)
- [Menerima file CloudTrail log dari beberapa wilayah](#) dan [Menerima file CloudTrail log dari beberapa akun](#)

Semua Amazon EMR pada tindakan EKS dicatat oleh CloudTrail dan didokumentasikan di Amazon [EMR pada](#) dokumentasi EKS API. Misalnya, panggilan ke `CreateVirtualCluster`, `StartJobRun` dan `ListJobRuns` tindakan menghasilkan entri dalam file CloudTrail log.

Setiap entri peristiwa atau log berisi informasi tentang entitas yang membuat permintaan tersebut. Informasi identitas membantu Anda menentukan hal berikut ini:

- Apakah permintaan itu dibuat dengan kredensial pengguna root atau AWS Identity and Access Management (IAM).
- Apakah permintaan tersebut dibuat dengan kredensial keamanan sementara untuk satu peran atau pengguna gabungan.
- Apakah permintaan itu dibuat oleh AWS layanan lain.

Untuk informasi selengkapnya, lihat [elemen Identitas CloudTrail pengguna](#).

Memahami entri file log Amazon EMR di EKS

Trail adalah konfigurasi yang memungkinkan pengiriman peristiwa sebagai file log ke bucket Amazon S3 yang Anda tentukan. CloudTrail file log berisi satu atau lebih entri log. Peristiwa mewakili permintaan tunggal dari sumber manapun dan mencakup informasi tentang tindakan yang diminta, tanggal dan waktu tindakan, parameter permintaan, dan sebagainya. CloudTrail file log bukanlah jejak tumpukan yang diurutkan dari panggilan API publik, jadi file tersebut tidak muncul dalam urutan tertentu.

Contoh berikut menunjukkan entri CloudTrail log yang menunjukkan [ListJobRunstindakan](#).

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE:admin",
    "arn": "arn:aws:sts::012345678910:assumed-role/Admin/admin",
    "accountId": "012345678910",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::012345678910:role/Admin",
        "accountId": "012345678910",
        "userName": "Admin"
      },
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2020-11-04T21:49:36Z"
      }
    }
  },
  "eventTime": "2020-11-04T21:52:58Z",
  "eventSource": "emr-containers.amazonaws.com",
  "eventName": "ListJobRuns",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "203.0.113.1",
  "userAgent": "aws-cli/1.11.167 Python/2.7.10 Darwin/16.7.0 botocore/1.7.25",
  "requestParameters": {
```

```
"virtualClusterId": "1K48XXXXXXHCB"  
},  
"responseElements": null,  
"requestID": "890b8639-e51f-11e7-b038-EXAMPLE",  
"eventID": "874f89fa-70fc-4798-bc00-EXAMPLE",  
"readOnly": true,  
"eventType": "AwsApiCall",  
"recipientAccountId": "012345678910"  
}
```

Menggunakan Hibah Akses Amazon S3 dengan Amazon EMR di EKS

Ikhtisar Hibah Akses S3 untuk Amazon EMR di EKS

Dengan Amazon EMR rilis 6.15.0 dan yang lebih tinggi, Amazon S3 Access Grants menyediakan solusi kontrol akses yang dapat diskalakan yang dapat Anda gunakan untuk menambah akses ke data Amazon S3 Anda dari Amazon EMR di EKS. Jika Anda memiliki konfigurasi izin yang kompleks atau besar untuk data S3, Anda dapat menggunakan Access Grants untuk menskalakan izin data S3 untuk pengguna, peran, dan aplikasi.

Gunakan S3 Access Grants untuk menambah akses ke data Amazon S3 di luar izin yang diberikan oleh peran runtime atau peran IAM yang dilampirkan ke identitas dengan akses ke Amazon EMR Anda di klaster EKS.

Untuk informasi selengkapnya, lihat [Mengelola akses dengan Hibah Akses S3 untuk Amazon EMR di](#) Panduan Manajemen EMR Amazon dan [Mengelola akses dengan Hibah Akses S3 di Panduan Pengguna](#) Layanan Penyimpanan Sederhana Amazon.

Halaman ini menjelaskan persyaratan untuk menjalankan pekerjaan Spark di Amazon EMR di EKS dengan integrasi S3 Access Grants. Dengan Amazon EMR di EKS, S3 Access Grants memerlukan pernyataan kebijakan IAM tambahan dalam peran eksekusi untuk pekerjaan Anda, dan konfigurasi penggantian tambahan untuk API. `StartJobRun` Untuk langkah-langkah menyiapkan Hibah Akses S3 dengan penerapan EMR Amazon lainnya, lihat dokumentasi berikut:

- [Menggunakan Hibah Akses S3 dengan Amazon EMR](#)
- [Menggunakan Hibah Akses S3 dengan EMR Tanpa Server](#)

Luncurkan Amazon EMR di kluster EKS dengan S3 Access Grants untuk manajemen data

Anda dapat mengaktifkan S3 Access Grants di Amazon EMR di EKS dan meluncurkan pekerjaan Spark. Saat aplikasi Anda membuat permintaan untuk data S3, Amazon S3 menyediakan kredensial sementara yang dicakup ke bucket, awalan, atau objek tertentu.

1. Siapkan peran eksekusi pekerjaan untuk EMR Amazon Anda di kluster EKS. Sertakan izin IAM yang diperlukan yang Anda perlukan untuk menjalankan pekerjaan Spark, dan: `s3:GetDataAccess` `s3:GetAccessGrantsInstanceForPrefix`

```
{
  "Effect": "Allow",
  "Action": [
    "s3:GetDataAccess",
    "s3:GetAccessGrantsInstanceForPrefix"
  ],
  "Resource": [
    //LIST ALL INSTANCE ARNS THAT THE ROLE IS ALLOWED TO QUERY
    "arn:aws_partition:s3:Region:account-id1:access-grants/default",
    "arn:aws_partition:s3:Region:account-id2:access-grants/default"
  ]
}
```

Note

Jika Anda menentukan peran IAM yang untuk eksekusi pekerjaan yang memiliki izin tambahan untuk mengakses S3 secara langsung, maka pengguna mungkin dapat mengakses data terlepas dari izin yang Anda tentukan di S3 Access Grants

2. Kirimkan pekerjaan ke EMR Amazon Anda di kluster EKS dengan label rilis Amazon EMR 6,15 atau lebih tinggi dan `emrfs-site` klasifikasi, seperti yang ditunjukkan contoh berikut. Ganti nilai *red text* dengan nilai yang sesuai untuk skenario penggunaan Anda.

```
{
  "name": "myjob",
  "virtualClusterId": "123456",
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "emr-7.13.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
```

```
"entryPoint": "entryPoint_location",
"entryPointArguments": ["argument1", "argument2"],
"sparkSubmitParameters": "--class main_class"
}
},
"configurationOverrides": {
  "applicationConfiguration": [
    {
      "classification": "emrfs-site",
      "properties": {
        "fs.s3.s3AccessGrants.enabled": "true",
        "fs.s3.s3AccessGrants.fallbackToIAM": "false"
      }
    }
  ],
}
}
```

Akses S3 Memberikan pertimbangan dengan Amazon EMR di EKS

Untuk informasi dukungan, kompatibilitas, dan perilaku penting saat Anda menggunakan Hibah Akses Amazon S3 dengan Amazon EMR di EKS, lihat [pertimbangan Hibah Akses S3 dengan Amazon EMR di Panduan Manajemen EMR Amazon](#).

Validasi kepatuhan untuk Amazon EMR di EKS

Auditor pihak ketiga menilai keamanan dan kepatuhan Amazon EMR di EKS sebagai bagian dari AWS beberapa program kepatuhan. Program ini mencakup SOC, PCI, FedRAMP, HIPAA, dan lainnya.

Ketahanan di Amazon EMR di EKS

Infrastruktur AWS global dibangun di sekitar AWS Wilayah dan Zona Ketersediaan. AWS Wilayah menyediakan beberapa Availability Zone yang terpisah secara fisik dan terisolasi, yang terhubung dengan latensi rendah, throughput tinggi, dan jaringan yang sangat redundan. Dengan Availability Zone, Anda dapat mendesain dan mengoperasikan aplikasi dan basis data yang secara otomatis mengalami kegagalan 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.

Untuk informasi selengkapnya tentang AWS Wilayah dan Availability Zone, lihat [Infrastruktur AWS Global](#).

Selain infrastruktur AWS global, Amazon EMR di EKS menawarkan integrasi dengan Amazon S3 melalui EMRFS untuk membantu mendukung ketahanan data dan kebutuhan pencadangan Anda.

Keamanan infrastruktur dalam Amazon EMR di EKS

Sebagai layanan terkelola, Amazon EMR dilindungi oleh keamanan jaringan AWS global. Untuk informasi tentang layanan AWS keamanan dan cara AWS melindungi infrastruktur, lihat [Keamanan AWS Cloud](#). Untuk mendesain AWS lingkungan Anda menggunakan praktik terbaik untuk keamanan infrastruktur, lihat [Perlindungan Infrastruktur dalam Kerangka Kerja](#) yang AWS Diarsiteksikan dengan Baik Pilar Keamanan.

Anda menggunakan panggilan API yang AWS dipublikasikan untuk mengakses Amazon EMR melalui jaringan. Klien harus mendukung hal-hal berikut:

- Keamanan Lapisan Pengangkutan (TLS). Kami mensyaratkan TLS 1.2 dan menganjurkan TLS 1.3.
- Sandi cocok dengan sistem kerahasiaan maju sempurna (perfect forward secrecy, PFS) seperti DHE (Ephemeral Diffie-Hellman) atau ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Sebagian besar sistem modern seperti Java 7 dan versi lebih baru mendukung mode-mode ini.

Konfigurasi dan analisis kerentanan

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:

- [Validasi kepatuhan untuk Amazon EMR di EKS](#)
- [Model Tanggung Jawab Bersama](#)
- [Amazon Web Services: Gambaran Umum Proses Keamanan](#) (whitepaper)

Connect ke Amazon EMR di EKS Menggunakan antarmuka VPC endpoint

Anda dapat terhubung langsung ke Amazon EMR di EKS menggunakan [titik akhir VPC Antarmuka \(AWS PrivateLink\)](#) di Virtual Private Cloud (VPC) Anda alih-alih terhubung melalui internet. Saat Anda menggunakan titik akhir VPC antarmuka, komunikasi antara VPC Anda dan Amazon EMR di EKS dilakukan sepenuhnya di dalam jaringan. AWS Setiap titik akhir VPC diwakili oleh satu atau lebih [antarmuka jaringan Elastic \(ENIs\)](#) dengan alamat IP pribadi di subnet VPC Anda.

Titik akhir VPC antarmuka menghubungkan VPC Anda langsung ke Amazon EMR di EKS tanpa gateway internet, perangkat NAT, koneksi VPN, atau koneksi Direct Connect. AWS Instans dalam VPC Anda tidak memerlukan alamat IP publik untuk berkomunikasi dengan API Amazon EMR di EKS.

Anda dapat membuat titik akhir VPC antarmuka untuk terhubung ke Amazon EMR di EKS menggunakan perintah `aws` (). Konsol Manajemen AWS atau Command Line Interface (CLI) AWS. Untuk informasi selengkapnya, lihat [Membuat Titik Akhir Antarmuka](#).

Setelah Anda membuat antarmuka VPC endpoint, jika Anda mengaktifkan nama host DNS privat untuk titik akhir, titik akhir Amazon EMR di EKS default menyelesaikan ke VPC endpoint Anda. Titik akhir nama layanan default untuk Amazon EMR di EKS adalah dalam format berikut.

```
emr-containers.Region.amazonaws.com
```

Jika Anda tidak mengaktifkan nama host DNS privat, Amazon VPC menyediakan nama titik akhir DNS yang dapat Anda gunakan dalam format berikut.

```
VPC_Endpoint_ID.emr-containers.Region.vpce.amazonaws.com
```

Untuk informasi selengkapnya, lihat [Titik Akhir VPC Antarmuka \(AWS PrivateLink\) di Panduan Pengguna Amazon VPC](#). Amazon EMR di EKS mendukung panggilan ke semua [Tindakan API](#) di dalam VPC Anda.

Anda dapat melampirkan kebijakan VPC endpoint ke VPC endpoint untuk mengontrol akses untuk prinsipal IAM. Anda juga dapat mengasosiasikan grup keamanan dengan VPC endpoint untuk mengontrol akses masuk dan keluar berdasarkan asal dan tujuan lalu lintas jaringan, seperti rentang alamat IP. Untuk informasi selengkapnya, lihat [Mengontrol Akses ke Layanan dengan VPC Endpoints](#).

Buat Kebijakan VPC Endpoint untuk Amazon EMR di EKS

Anda dapat membuat kebijakan untuk Amazon VPC endpoint untuk Amazon EMR di EKS untuk menentukan hal berikut:

- Prinsip-prinsip yang dapat atau tidak dapat melakukan tindakan
- Tindakan yang dapat dilakukan
- Sumber daya yang dapat digunakan untuk mengambil tindakan

Untuk informasi selengkapnya, lihat [Mengontrol Akses ke Layanan dengan VPC Endpoint](#) dalam Panduan Pengguna Amazon VPC.

Example Kebijakan Titik Akhir VPC untuk Menolak Semua Akses Dari Akun Tertentu AWS

Kebijakan titik akhir VPC berikut menolak **123456789012** semua akses AWS akun ke sumber daya menggunakan titik akhir.

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Resource": "*",
      "Principal": "*"
    },
    {
      "Action": "*",
      "Effect": "Deny",
      "Resource": "*",
      "Principal": {
        "AWS": [
          "123456789012"
        ]
      }
    }
  ]
}
```

Example Kebijakan VPC Endpoint untuk Mengizinkan Akses VPC Hanya ke Prinsipal IAM (Pengguna) Tertentu

Kebijakan titik akhir VPC berikut memungkinkan akses penuh hanya ke akun pengguna IAM. *lijuan* AWS *123456789012* Semua prinsipal IAM lain ditolak aksesnya menggunakan titik akhir.

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Resource": "*",
      "Principal": {
        "AWS": [
          "arn:aws:iam::123456789012:user/lijuan"
        ]
      }
    }
  ]
}
```

Example Kebijakan VPC Endpoint untuk Mengizinkan Operasi Read-Only Amazon EMR di EKS

Kebijakan titik akhir VPC berikut hanya mengizinkan AWS akun *123456789012* untuk melakukan EMR Amazon yang ditentukan pada tindakan EKS.

Tindakan yang ditentukan memberikan akses setara dengan read-only untuk Amazon EMR di EKS. Semua tindakan lain pada VPC ditolak untuk akun yang ditentukan. Semua akun lain ditolak akses apa pun. Untuk daftar tindakan Amazon EMR di EKS, lihat [Kunci Tindakan, Sumber Daya, dan Kondisi untuk Amazon EMR di EKS](#).

```
{
  "Statement": [
    {
      "Action": [
        "emr-containers:DescribeJobRun",
        "emr-containers:DescribeVirtualCluster",
        "emr-containers:ListJobRuns",
        "emr-containers:ListTagsForResource",
        "emr-containers:ListVirtualClusters"
      ],
      "Effect": "Allow",
    }
  ]
}
```

```

    "Resource": "*",
    "Principal": {
      "AWS": [
        "123456789012"
      ]
    }
  ]
}

```

Example Kebijakan VPC Endpoint Menolak Akses ke Kluster Virtual Tertentu

Kebijakan titik akhir VPC berikut memungkinkan akses penuh untuk semua akun dan prinsipal, tetapi menolak akses apa pun untuk AWS akun `123456789012` ke tindakan yang dilakukan pada cluster virtual dengan ID cluster. `A1B2CD34EF5G` Amazon EMR lainnya pada tindakan EKS yang tidak mendukung izin tingkat sumber daya untuk kluster virtual masih diizinkan. Untuk daftar EMR Amazon pada tindakan EKS dan jenis sumber daya yang sesuai, lihat [Tindakan, Sumber Daya, dan Kunci Kondisi untuk Amazon EMR di EKS - di Panduan Pengguna](#). AWS Identity and Access Management

```

{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Resource": "*",
      "Principal": "*"
    },
    {
      "Action": "*",
      "Effect": "Deny",
      "Resource": "arn:aws:emr-containers:us-west-2:123456789012:/virtualclusters/A1B2CD34EF5G",
      "Principal": {
        "AWS": [
          "123456789012"
        ]
      }
    }
  ]
}

```

Atur akses lintas akun untuk Amazon EMR di EKS

Anda dapat mengatur akses lintas-akun untuk Amazon EMR di EKS. Akses lintas akun memungkinkan pengguna dari satu AWS akun untuk menjalankan Amazon EMR pada pekerjaan EKS dan mengakses data dasar milik AWS akun lain.

Prasyarat

Untuk mengatur akses lintas akun untuk Amazon EMR di EKS, Anda akan menyelesaikan tugas saat masuk ke akun berikut AWS :

- AccountA- AWS Akun tempat Anda membuat EMR Amazon di kluster virtual EKS dengan mendaftarkan Amazon EMR dengan namespace di cluster EKS.
- AccountB- AWS Akun yang berisi bucket Amazon S3 atau tabel DynamoDB yang Anda inginkan untuk diakses oleh EMR Amazon Anda di pekerjaan EKS.

Anda harus menyiapkan hal-hal berikut di AWS akun Anda sebelum menyiapkan akses lintas akun:

- Kluster virtual Amazon EMR di EKS di AccountA di mana Anda ingin menjalankan tugas.
- Peran eksekusi tugas di AccountA yang memiliki izin yang diperlukan untuk menjalankan tugas di kluster virtual. Untuk informasi lebih lanjut, lihat [Untuk membuat peran eksekusi tugas](#) dan [Menggunakan peran eksekusi tugas dengan Amazon EMR di EKS](#).

Cara mengakses bucket Amazon S3 lintas akun atau tabel DynamoDB

Untuk mengatur akses lintas-akun untuk Amazon EMR di EKS, selesaikan langkah-langkah berikut.

1. Buat bucket Amazon S3, `cross-account-bucket`, di AccountB. Untuk informasi lebih lanjut, lihat [Membuat bucket](#). Jika Anda ingin memiliki akses lintas-akun ke DynamoDB, Anda juga dapat membuat tabel DynamoDB di AccountB. Untuk informasi selengkapnya, lihat [Membuat tabel DynamoDB](#).
2. Buat IAM role `Cross-Account-Role-B` dalam AccountB yang dapat mengakses `cross-account-bucket`.
 1. Masuklah ke konsol IAM.
 2. Pilih Peran dan buat peran baru: `Cross-Account-Role-B`. Untuk informasi selengkapnya tentang cara membuat peran IAM, lihat [Membuat peran IAM](#) di Panduan pengguna IAM.

3. Buat kebijakan IAM yang menentukan izin untuk Cross-Account-Ro1e-B untuk mengakses S3 bucket `cross-account-bucket`, seperti yang ditunjukkan pernyataan kebijakan berikut. Kemudian lampirkan kebijakan IAM ke Cross-Account-Ro1e-B. Untuk informasi selengkapnya, lihat [Membuat Kebijakan Baru](#) di Panduan pengguna IAM.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:*"
      ],
      "Resource": [
        "arn:aws:s3:::cross-account-bucket",
        "arn:aws:s3:::cross-account-bucket/*"
      ],
      "Sid": "AllowS3"
    }
  ]
}
```

Jika akses DynamoDB diperlukan, buat kebijakan IAM yang menentukan izin untuk mengakses tabel DynamoDB lintas akun. Kemudian lampirkan kebijakan IAM ke Cross-Account-Ro1e-B. Untuk informasi selengkapnya, lihat [Buat tabel DynamoDB](#) dalam Panduan Pengguna IAM.

Berikut ini adalah kebijakan untuk mengakses tabel DynamoDB, `CrossAccountTable`.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:*"
      ],
      "Resource": [
```

```

        "arn:aws:dynamodb:us-east-1:*:table/CrossAccountTable"
    ],
    "Sid": "AllowDYNAMODB"
  }
]
}

```

3. Cara mengedit hubungan kepercayaan untuk peran Cross-Account-Role-B.

1. Untuk mengonfigurasi hubungan kepercayaan untuk peran, pilih tab Hubungan Kepercayaan di konsol IAM untuk peran yang dibuat di langkah 2: Cross-Account-Role-B.
2. Pilih Edit Hubungan Kepercayaan.
3. Tambahkan dokumen kebijakan berikut, yang memungkinkan Job-Execution-Role-A dalam AccountA untuk mengasumsikan peran Cross-Account-Role-B ini.

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowSTSAssumerole",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:role/Job-Execution-Role-A"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}

```

4. Job-Execution-Role-A Hibah AccountA dengan - STS Asumsikan izin peran untuk diasumsikan Cross-Account-Role-B.

1. Di konsol IAM untuk AWS akun AccountA, pilih Job-Execution-Role-A.
2. Tambahkan pernyataan kebijakan berikut pada Job-Execution-Role-A untuk mengizinkan tindakan AssumeRole di peran Cross-Account-Role-B.

JSON

```

{
  "Version": "2012-10-17",

```

```

"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "sts:AssumeRole"
    ],
    "Resource": [
      "arn:aws:iam::*:role/Cross-Account-Role-B"
    ],
    "Sid": "AllowSTSAssumerole"
  }
]
}

```

- Untuk akses Amazon S3, atur parameter spark-submit berikut (spark conf) saat mengirimkan tugas ke Amazon EMR di EKS.

Note

Secara default, EMRFS menggunakan peran eksekusi tugas untuk mengakses bucket S3 dari tugas. Tapi saat customAWSCredentialsProvider diatur ke AssumeRoleAWSCredentialsProvider, EMRFS menggunakan peran yang sesuai yang Anda tentukan dengan ASSUME_ROLE_CREDENTIALS_ROLE_ARN bukan dari Job-Execution-Role-A untuk akses Amazon S3.

- --conf
spark.hadoop.fs.s3.customAWSCredentialsProvider=com.amazonaws.emr.AssumeRole
- --conf
spark.kubernetes.driverEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN=arn:aws:iam::**AccountA**:role/Cross-Account-Role-B \
- --conf
spark.executorEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN=arn:aws:iam::**AccountB**:role/Cross-Account-Role-B \

Note

Anda harus menetapkan `ASSUME_ROLE_CREDENTIALS_ROLE_ARN` baik untuk pelaksana maupun driver env dalam konfigurasi tugas spark.

Untuk akses lintas akun DynamoDB, Anda harus mengatur `--conf spark.dynamodb.customAWSCredentialsProvider=com.amazonaws.emr.AssumeRoleAWSCredentialsProvider`.

6. Jalankan Amazon EMR pada tugas EKS dengan akses lintas-akun, seperti yang ditunjukkan contoh berikut.

```
aws emr-containers start-job-run \
--virtual-cluster-id 123456 \
--name myjob \
--execution-role-arn execution-role-arn \
--release-label emr-6.2.0-latest \
--job-driver '{"sparkSubmitJobDriver": {"entryPoint": "entryPoint_location",
"entryPointArguments": ["arguments_list"], "sparkSubmitParameters": "--class
<main_class> --conf spark.executor.instances=2 --conf spark.executor.memory=2G
--conf spark.executor.cores=2 --conf spark.driver.cores=1 --conf
spark.hadoop.fs.s3.customAWSCredentialsProvider=com.amazonaws.emr.AssumeRoleAWSCredentials
--conf
spark.kubernetes.driverEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN=arn:aws:iam::AccountB:role/
Cross-Account-Role-B --conf
spark.executorEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN=arn:aws:iam::AccountB:role/
Cross-Account-Role-B"}} ' \
--configuration-overrides '{"applicationConfiguration": [{"classification":
"spark-defaults", "properties": {"spark.driver.memory": "2G"}}],
"monitoringConfiguration": {"cloudWatchMonitoringConfiguration":
{"logGroupName": "log_group_name", "logStreamNamePrefix": "log_stream_prefix"},
"persistentAppUI": "ENABLED", "s3MonitoringConfiguration": {"logUri": "s3://
my_s3_log_location" }]]'
```

Menandai Sumber Daya Amazon EMR di EKS Anda

Untuk membantu Anda mengelola Amazon EMR di EKS, Anda dapat menetapkan metadata Anda sendiri ke setiap sumber daya menggunakan tanda. Topik ini memberikan gambaran umum dari fungsi tanda dan menunjukkan kepada Anda cara membuat tanda.

Topik

- [Dasar-dasar tag](#)
- [Beri tanda pada sumber daya Anda](#)
- [Pembatasan tag](#)
- [Bekerja dengan tag menggunakan AWS CLI dan Amazon EMR di EKS API](#)

Dasar-dasar tag

Tag adalah label yang Anda tetapkan ke AWS sumber daya. Setiap tag terdiri dari satu kunci dan satu nilai opsional, yang keduanya Anda tentukan sendiri.

Tag memungkinkan Anda untuk mengkategorikan AWS sumber daya berdasarkan atribut seperti tujuan, pemilik, atau lingkungan. Saat Anda memiliki banyak sumber daya dengan jenis yang sama, Anda dapat dengan cepat mengidentifikasi sumber daya tertentu berdasarkan tanda yang telah Anda tetapkan. Misalnya, Anda dapat menentukan serangkaian tanda untuk Amazon EMR Anda pada kluster EKS untuk membantu Anda melacak setiap pemilik dan tingkat tumpukan kluster. Kami menyarankan agar Anda merancang serangkaian konsisten kunci tanda untuk setiap jenis sumber daya. Kemudian Anda dapat mencari dan memfilter sumber daya berdasarkan tanda yang Anda tambahkan.

Tanda tidak secara otomatis ditetapkan ke sumber daya Anda. Setelah Anda menambahkan tag, Anda dapat mengedit kunci serta nilai tag atau menghapus tag dari sumber daya kapan saja. Jika Anda menghapus sebuah sumber daya, tag apa pun untuk sumber daya tersebut juga dihapus.

Tanda tidak memiliki makna semantik pada Amazon EMR di EKS dan diterjemahkan sebagai serangkaian karakter saja.

Nilai tanda bisa berupa string kosong, tapi tidak null. Kunci tanda tidak bisa berupa string kosong. Jika Anda menambahkan tanda yang memiliki kunci yang sama dengan tanda yang ada pada sumber daya tersebut, nilai yang baru akan menimpa nilai sebelumnya.

Jika Anda menggunakan AWS Identity and Access Management (IAM), Anda dapat mengontrol pengguna mana di AWS akun Anda yang memiliki izin untuk mengelola tag.

Untuk contoh kebijakan kontrol akses berbasis tanda, lihat [Kebijakan untuk kendali akses berbasis tanda](#).

Beri tanda pada sumber daya Anda

Anda dapat menandai klaster virtual dan tugas berjalan baru atau yang sudah ada yang berada dalam status aktif. Status aktif untuk tugas berjalan meliputi: PENDING, SUBMITTED, RUNNING, dan CANCEL_PENDING. Status aktif untuk klaster virtual meliputi: RUNNING, TERMINATING dan ARRESTED. Untuk informasi lebih lanjut, lihat [Status tugas berjalan](#) dan [Status klaster virtual](#).

Ketika klaster virtual dihentikan, tanda dibersihkan dan tidak lagi dapat diakses.

Jika Anda menggunakan EMR Amazon di EKS API, the AWS CLI, atau AWS SDK, Anda dapat menerapkan tag ke sumber daya baru menggunakan parameter tag pada tindakan API yang relevan. Anda dapat memasang tanda ke sumber daya yang ada menggunakan tindakan API TagResource.

Anda dapat menggunakan beberapa tindakan penciptaan sumber daya untuk menentukan tanda untuk sumber daya saat sumber daya diciptakan. Dalam kasus ini, jika tanda tidak dapat diterapkan saat sumber daya sedang dibuat, sumber daya gagal untuk dibuat. Mekanisme ini memastikan bahwa sumber daya yang Anda inginkan untuk ditandai pada penciptaan dibuat dengan tanda tertentu atau tidak dibuat sama sekali. Jika Anda menandai sumber daya pada saat penciptaan, Anda tidak perlu untuk menjalankan skrip penandaan khusus setelah penciptaan sumber daya.

Tabel berikut menjelaskan sumber daya Amazon EMR di EKS yang dapat ditandai.

Sumber Daya	Mendukung tag	Mendukung penyebaran tag	Mendukung penandaan pada pembuatan (Amazon EMR pada EKS API AWS CLI,, AWS dan SDK)	API untuk pembuatan (tanda dapat ditambahkan selama pembuatan)
Klaster virtual	Ya	Tidak. Tanda yang terkait dengan klaster	Ya	CreateVirtualCluster

Sumber Daya	Mendukung tag	Mendukung penyebaran tag	Mendukung penandaan pada pembuatan (Amazon EMR pada EKS API AWS CLI,, AWS dan SDK)	API untuk pembuatan (tanda dapat ditambahkan selama pembuatan)
		virtual tidak menyebarkan ke tugas berjalan yang dikirimkan ke cluster virtual tersebut.		
Tugas berjalan	Ya	Tidak	Ya	StartJobRun

Pembatasan tag

Batasan dasar berikut berlaku untuk tag:

- Jumlah maksimum tag per sumber daya – 50
- Untuk setiap sumber daya, setiap kunci tag harus unik, dan setiap kunci tag hanya dapat memuat satu nilai.
- Panjang kunci maksimum – 128 karakter Unicode dalam UTF-8
- Panjang nilai maksimum - 256 karakter Unicode di UTF-8
- Jika skema penandaan Anda digunakan di beberapa AWS layanan dan sumber daya, ingatlah bahwa layanan lain mungkin memiliki batasan pada karakter yang diizinkan. Karakter yang umumnya boleh digunakan adalah huruf, angka, spasi yang dapat diwakili dalam UTF-8, serta karakter berikut: + - = . _ : / @.
- Kunci dan nilai tag peka huruf besar dan kecil.
- Nilai tanda bisa berupa string kosong, tapi tidak null. Kunci tanda tidak bisa berupa string kosong.
- Jangan gunakan `aws :`, `AWS :`, atau kombinasi huruf besar atau kecil seperti prefiks baik untuk kunci atau nilai. Ini hanya disediakan untuk AWS digunakan.

Bekerja dengan tag menggunakan AWS CLI dan Amazon EMR di EKS API

Gunakan AWS CLI perintah berikut atau Amazon EMR pada operasi EKS API untuk menambahkan, memperbarui, membuat daftar, dan menghapus tag untuk sumber daya Anda.

Tugas	AWS CLI	Tindakan API
Penambahan atau penimpaan satu tag atau lebih	tag-sumber daya	TagResource
Membuat daftar tanda untuk sumber daya	list-tags-for-resource	ListTagsForResource
Penghapusan satu tag atau lebih	untag-sumber daya	UntagResource

Contoh-contoh berikut menunjukkan cara memberi tanda atau menghapus tanda pada sumber daya menggunakan AWS CLI.

Contoh 1: Menandai klaster virtual yang ada

Perintah berikut memberi tanda klaster virtual yang ada.

```
aws emr-containers tag-resource --resource-arn resource_ARN --tags team=devs
```

Contoh 2: Hapus tag cluster virtual yang ada

Perintah berikut menghapus tanda dari klaster virtual yang ada.

```
aws emr-containers untag-resource --resource-arn resource_ARN --tag-keys tag_key
```

Contoh 3: Daftar tag untuk sumber daya

Perintah berikut membuat daftar tanda yang terkait dengan sumber daya yang ada.

```
aws emr-containers list-tags-for-resource --resource-arn resource_ARN
```

Pemecahan masalah untuk Amazon EMR di EKS

Bagian ini menjelaskan cara memecahkan masalah dengan Amazon EMR di EKS. Untuk informasi tentang cara memecahkan masalah umum dengan Amazon EMR, lihat [Memecahkan masalah kluster di Panduan Manajemen EMR Amazon](#).

Topik

- [Memecahkan masalah pekerjaan yang menggunakan PersistentVolumeClaims \(PVC\)](#)
- [Memecahkan masalah Amazon EMR pada penskalaan otomatis vertikal EKS](#)
- [Memecahkan masalah Amazon EMR pada operator EKS Spark](#)

Memecahkan masalah pekerjaan yang menggunakan PersistentVolumeClaims (PVC)

Jika Anda perlu membuat, membuat daftar, atau menghapus PersistentVolumeClaims (PVC) untuk suatu pekerjaan tetapi tidak menambahkan izin PVC ke peran default Kubernetes `emr-container`, pekerjaan akan gagal saat Anda mengirimkannya. Tanpa izin ini, peran `emr-container` tidak dapat membuat peran yang diperlukan untuk driver Spark atau klien Spark. Tidak cukup menambahkan izin ke driver Spark atau peran klien, seperti yang disarankan oleh pesan kesalahan. Peran utama `emr-container` harus menyertakan izin yang diperlukan juga. Bagian ini menjelaskan cara menambahkan izin yang diperlukan ke peran utama `emr-container`.

Verifikasi

Untuk memverifikasi apakah peran `emr-container` Anda memiliki izin yang diperlukan atau tidak, atur variabel `NAMESPACE` dengan nilai Anda sendiri dan kemudian jalankan perintah berikut:

```
export NAMESPACE=YOUR_VALUE
kubectl describe role emr-containers -n ${NAMESPACE}
```

Selain itu, untuk memverifikasi apakah peran Spark dan klien memiliki izin yang diperlukan, jalankan perintah berikut:

```
kubectl describe role emr-containers-role-spark-driver -n ${NAMESPACE}
kubectl describe role emr-containers-role-spark-client -n ${NAMESPACE}
```

Jika izin tidak ada, lanjutkan dengan tambalan, sebagai berikut.

Patch

1. Jika pekerjaan tanpa izin sedang berjalan, hentikan pekerjaan ini.
2. Buat file bernama RBAC_Patch.py sebagai berikut:

```
import os
import subprocess as sp
import tempfile as temp
import json
import argparse
import uuid

def delete_if_exists(dictionary: dict, key: str):
    if dictionary.get(key, None) is not None:
        del dictionary[key]

def doTerminalCmd(cmd):
    with temp.TemporaryFile() as f:
        process = sp.Popen(cmd, stdout=f, stderr=f)
        process.wait()
        f.seek(0)
        msg = f.read().decode()
    return msg

def patchRole(roleName, namespace, extraRules, skipConfirmation=False):
    cmd = f"kubectl get role {roleName} -n {namespace} --output json".split(" ")
    msg = doTerminalCmd(cmd)
    if "(NotFound)" in msg and "Error" in msg:
        print(msg)
        return False
    role = json.loads(msg)
    rules = role["rules"]
    rulesToAssign = extraRules[::]
    passedRules = []
    for rule in rules:
        apiGroups = set(rule["apiGroups"])
        resources = set(rule["resources"])
        verbs = set(rule["verbs"])
        for extraRule in extraRules:
            passes = 0
            apiGroupsExtra = set(extraRule["apiGroups"])
```

```

        resourcesExtra = set(extraRule["resources"])
        verbsExtra = set(extraRule["verbs"])
        passes += len(apiGroupsExtra.intersection(apiGroups)) >=
len(apiGroupsExtra)
        passes += len(resourcesExtra.intersection(resources)) >=
len(resourcesExtra)
        passes += len(verbsExtra.intersection(verbs)) >= len(verbsExtra)
        if passes >= 3:
            if extraRule not in passedRules:
                passedRules.append(extraRule)
                if extraRule in rulesToAssign:
                    rulesToAssign.remove(extraRule)
            break
    prompt_text = "Apply Changes?"
    if len(rulesToAssign) == 0:
        print(f"The role {roleName} seems to already have the necessary
permissions!")
        prompt_text = "Proceed anyways?"
    for ruleToAssign in rulesToAssign:
        role["rules"].append(ruleToAssign)
    delete_if_exists(role, "creationTimestamp")
    delete_if_exists(role, "resourceVersion")
    delete_if_exists(role, "uid")
    new_role = json.dumps(role, indent=3)
    uid = uuid.uuid4()
    filename = f"Role-{roleName}-New_Permissions-{uid}-TemporaryFile.json"
    try:
        with open(filename, "w+") as f:
            f.write(new_role)
            f.flush()
        prompt = "y"
        if not skipConfirmation:
            prompt = input(
                doTerminalCmd(f"kubectl diff -f {filename}".split(" ")) +
f"\n{prompt_text} y/n: "
                ).lower().strip()
            while prompt != "y" and prompt != "n":
                prompt = input("Please make a valid selection. y/n:
").lower().strip()
            if prompt == "y":
                print(doTerminalCmd(f"kubectl apply -f {filename}".split(" ")))
    except Exception as e:
        print(e)
    os.remove(f"./{filename}")

```

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument("-n", "--namespace",
                        help="Namespace of the Role. By default its the
VirtualCluster's namespace",
                        required=True,
                        dest="namespace"
                        )

    parser.add_argument("-p", "--no-prompt",
                        help="Applies the patches without asking first",
                        dest="no_prompt",
                        default=False,
                        action="store_true"
                        )
    args = parser.parse_args()

    emrRoleRules = [
        {
            "apiGroups": [""],
            "resources": ["persistentvolumeclaims"],
            "verbs": ["list", "create", "delete", "patch"]
        }
    ]

    driverRoleRules = [
        {
            "apiGroups": [""],
            "resources": ["persistentvolumeclaims"],
            "verbs": ["list", "create", "delete", "patch", "deletecollection"]
        },
        {
            "apiGroups": [""],
            "resources": ["services"],
            "verbs": ["get", "list", "describe", "create", "delete", "watch",
"deletecollection"]
        },
        {
            "apiGroups": [""],
            "resources": ["configmaps", "pods"],
            "verbs": ["deletecollection"]
        }
    ]
```

```
]

clientRoleRules = [
    {
        "apiGroups": [""],
        "resources": ["persistentvolumeclaims"],
        "verbs": ["list", "create", "delete", "patch"]
    }
]

patchRole("emr-containers", args.namespace, emrRoleRules, args.no_prompt)
patchRole("emr-containers-role-spark-driver", args.namespace, driverRoleRules,
args.no_prompt)
patchRole("emr-containers-role-spark-client", args.namespace, clientRoleRules,
args.no_prompt)
```

3. Jalankan skrip Python:

```
python3 RBAC_Patch.py -n ${NAMESPACE}
```

4. Perbedaan kubectl antara izin baru dan yang lama muncul. Tekan y untuk menambal peran.

5. Verifikasi tiga peran dengan izin tambahan sebagai berikut:

```
kubectl describe role -n ${NAMESPACE}
```

6. Jalankan skrip python:

```
python3 RBAC_Patch.py -n ${NAMESPACE}
```

7. Setelah menjalankan perintah, itu akan menampilkan perbedaan kubectl antara izin baru dan yang lama. Tekan y untuk menambal peran.

8. Verifikasi tiga peran dengan izin tambahan:

```
kubectl describe role -n ${NAMESPACE}
```

9. Kirim pekerjaan lagi.

Patch manual

Jika izin yang diperlukan aplikasi Anda berlaku untuk sesuatu selain aturan PVC, Anda dapat menambahkan izin Kubernetes secara manual untuk klaster virtual Amazon EMR Anda sesuai kebutuhan.

Note

Peran `emr-container` adalah peran utama. Ini berarti bahwa itu harus memberikan semua izin yang diperlukan sebelum Anda dapat mengubah peran driver atau klien yang mendasarinya.

1. Unduh izin saat ini ke dalam file yml dengan menjalankan perintah di bawah ini:

```
kubectl get role -n ${NAMESPACE} emr-containers -o yaml >> emr-containers-role-patch.yaml
kubectl get role -n ${NAMESPACE} emr-containers-role-spark-driver -o yaml >> driver-role-patch.yaml
kubectl get role -n ${NAMESPACE} emr-containers-role-spark-client -o yaml >> client-role-patch.yaml
```

2. Berdasarkan izin yang diperlukan aplikasi Anda, edit setiap file dan tambahkan aturan tambahan seperti berikut ini:

- `emr-containers-role-patch.yaml`

```
- apiGroups:
  - ""
  resources:
  - persistentvolumeclaims
  verbs:
  - list
  - create
  - delete
  - patch
```

- `driver-role-patch.yaml`

```
- apiGroups:
  - ""
  resources:
  - persistentvolumeclaims
```

```
verbs:
- list
- create
- delete
- patch
- deletecollection
- apiGroups:
- ""
resources:
- services
verbs:
- get
- list
- describe
- create
- delete
- watch
- deletecollection
- apiGroups:
- ""
resources:
- configmaps
- pods
verbs:
- deletecollection
```

- client-role-patch.yaml

```
- apiGroups:
- ""
resources:
- persistentvolumeclaims
verbs:
- list
- create
- delete
- patch
```

3. Hapus atribut berikut dengan nilainya. Ini diperlukan untuk menerapkan pembaruan.

- CreationTimeStamp
- ResourceVersion
- uid

4. Akhirnya, jalankan tambalan:

```
kubectl apply -f emr-containers-role-patch.yaml
kubectl apply -f driver-role-patch.yaml
kubectl apply -f client-role-patch.yaml
```

Memecahkan masalah Amazon EMR pada penskalaan otomatis vertikal EKS

Lihat bagian berikut jika Anda mengalami masalah saat menyiapkan Amazon EMR pada operator penskalaan otomatis vertikal EKS di kluster Amazon EKS dengan Operator Lifecycle Manager. Untuk informasi selengkapnya termasuk langkah-langkah untuk menyelesaikan instalasi, lihat [Menggunakan penskalaan otomatis vertikal dengan pekerjaan Amazon EMR Spark](#).

403 Kesalahan terlarang

Jika Anda mengikuti langkah-langkah [Instal Operator Lifecycle Manager \(OLM\) di kluster Amazon EKS](#), menjalankan `olm status` perintah, dan mengembalikan 403 Forbidden kesalahan seperti di bawah ini, Anda mungkin tidak memperoleh token otentikasi ke repositori Amazon ECR untuk operator.

Untuk mengatasi masalah ini, ulangi langkah [Instal EMR Amazon pada operator penskalaan otomatis vertikal EKS](#) untuk mendapatkan token. Kemudian, coba instalasi lagi.

```
Error: FATA[0002] Failed to run bundle: pull bundle image: error pulling image IMAGE.
error resolving name : unexpected status code [manifests latest]: 403 Forbidden
```

Namespace Kubernetes tidak ditemukan

Saat Anda [menyiapkan EMR Amazon pada operator penskalaan otomatis vertikal EKS di kluster Amazon EKS](#), Anda mungkin mendapatkan `namespaces not found` kesalahan seperti yang ditunjukkan di sini:

```
FATA[0020] Failed to run bundle: create catalog: error creating catalog source:
namespaces "NAME" not found.
```

Jika namespace yang Anda tentukan tidak ada, OLM tidak akan menginstal operator penskalaan otomatis vertikal. Untuk mengatasi masalah ini, gunakan perintah berikut untuk membuat namespace. Kemudian, coba instalasi lagi.

```
kubectl create namespace NAME
```

Kesalahan saat menyimpan kredensial Docker

Untuk [mengatur penskalaan otomatis vertikal](#), Anda harus [mengautentikasi](#) dan mengambil Amazon EMR Anda pada gambar Docker terkait penskalaan otomatis vertikal EKS. Saat Anda melakukan ini, Anda mungkin mendapatkan kesalahan seperti berikut jika Docker tidak berjalan:

```
aws ecr get-login-password \  
  --region $REGION | docker login \  
  --username AWS \  
  --password-stdin $ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com
```

```
Error saving credentials: error storing credentials - err: exit status 1  
out: 'Post "http://ipc/registry/credstore-updated": dial unix backend.sock: connect: no  
such file or directory'
```

Untuk mengatasi masalah ini, konfirmasikan bahwa Docker sedang berjalan atau buka Docker Desktop. Kemudian, coba simpan kredensialmu lagi.

Memecahkan masalah Amazon EMR pada operator EKS Spark

Lihat bagian berikut jika Anda mengalami masalah dengan Amazon EMR di operator EKS Spark. Untuk informasi selengkapnya termasuk langkah-langkah untuk menyelesaikan instalasi, lihat [Menjalankan pekerjaan Spark dengan operator Spark](#).

Kesalahan pada instalasi bagan Helm

Jika Anda mengikuti langkah-langkah [Instal operator Spark](#) dan mengembalikan INSTALLATION FAILED kesalahan seperti di bawah ini ketika Anda mencoba menginstal atau memverifikasi bagan Helm, Anda mungkin tidak memperoleh token otentikasi ke repositori Amazon ECR untuk operator.

Untuk mengatasi masalah ini, ulangi langkah [Instal operator Spark](#) untuk mengautentikasi klien Helm Anda ke registri Amazon ECR. Kemudian, coba langkah instalasi lagi.

```
Error: INSTALLATION FAILED: Kubernetes cluster unreachable: the server has asked for the client to provide credentials
```

UnsupportedFileSystemException: Tidak FileSystem untuk skema "s3"

Anda mungkin menemukan pengecualian berikut di utas "main":

```
org.apache.hadoop.fs.UnsupportedFileSystemException: No FileSystem for scheme "s3"
```

Jika ini terjadi, tambahkan pengecualian berikut ke SparkApplication spesifikasi:

```
hadoopConf:
  # EMRFS filesystem
  fs.s3.customAWSCredentialsProvider:
com.amazonaws.auth.WebIdentityTokenCredentialsProvider
  fs.s3.impl: com.amazon.ws.emr.hadoop.fs.EmrFileSystem
  fs.AbstractFileSystem.s3.impl: org.apache.hadoop.fs.s3.EMRFSDelegate
  fs.s3.buffer.dir: /mnt/s3
  fs.s3.getObject.initialSocketTimeoutMilliseconds: "2000"
  mapreduce.fileoutputcommitter.algorithm.version.emr_internal_use_only.EmrFileSystem:
"2"
  mapreduce.fileoutputcommitter.cleanup-
failures.ignored.emr_internal_use_only.EmrFileSystem: "true"
sparkConf:
  # Required for EMR Runtime
  spark.driver.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/hadoop-
aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/share/aws/
emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/security/conf:/
usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-glue-datacatalog-
spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-serde.jar:/usr/share/aws/
sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/hadoop/extrajars/*
  spark.driver.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/lib/
native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/native
  spark.executor.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/hadoop-
aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/share/aws/
emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/security/conf:/
usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-glue-datacatalog-
spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-serde.jar:/usr/share/aws/
sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/hadoop/extrajars/*
  spark.executor.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/lib/
native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/native
```

Amazon EMR pada titik akhir dan kuota layanan EKS

Berikut ini adalah titik akhir layanan dan kuota layanan untuk Amazon EMR di EKS. Untuk terhubung secara terprogram ke AWS layanan, Anda menggunakan titik akhir. Selain AWS titik akhir standar, beberapa AWS layanan menawarkan titik akhir FIPS di Wilayah tertentu. Untuk informasi selengkapnya, lihat [AWS titik akhir layanan](#). Kuota layanan, juga disebut sebagai batas, adalah jumlah maksimum sumber daya layanan atau operasi untuk AWS akun Anda. Untuk informasi lebih lanjut, lihat [AWS kuota layanan](#).

Titik akhir layanan

Wilayah AWS nama	Kode	Titik akhir	Protokol
Timur AS (N. Virginia)	us-east-1	emr-containers.us-east-1.amazonaws.com	HTTPS
AS Timur (Ohio)	us-east-2	emr-containers.us-east-2.amazonaws.com	HTTPS
AS Barat (California Utara)	us-west-1	emr-containers.us-west-1.amazonaws.com	HTTPS
US West (Oregon)	us-west-2	emr-containers.us-west-2.amazonaws.com	HTTPS
Asia Pacific (Tokyo)	ap-northeast-1	emr-containers.ap-northeast-1.amazonaws.com	HTTPS
Asia Pasifik (Seoul)	ap-northeast-2	emr-containers.ap-northeast-2.amazonaws.com	HTTPS
Asia Pasifik (Osaka)	ap-northeast-3	emr-containers.ap-northeast-3.amazonaws.com	HTTPS
Asia Pasifik (Mumbai)	ap-south-1	emr-containers.ap-south-1.amazonaws.com	HTTPS

Wilayah AWS nama	Kode	Titik akhir	Protokol
Asia Pasifik (Hyderabad)	ap-south-2	emr-containers.ap-south-2.amazonaws.com	HTTPS
Asia Pasifik (Singapura)	ap-southeast-1	emr-containers.ap-southeast-1.amazonaws.com	HTTPS
Asia Pasifik (Sydney)	ap-southeast-2	emr-containers.ap-southeast-2.amazonaws.com	HTTPS
Asia Pasifik (Jakarta)	ap-southeast-3	emr-containers.ap-southeast-3.amazonaws.com	HTTPS
Asia Pasifik (Hong Kong)	ap-east-1	emr-containers.ap-east-1.amazonaws.com	HTTPS
Africa (Cape Town)	af-south-1	emr-containers.af-south-1.amazonaws.com	HTTPS
Kanada (Pusat)	ca-central-1	emr-containers.ca-central-1.amazonaws.com	HTTPS
Tiongkok (Ningxia)	cn-northwest-1	emr-containers.cn-northwest-1.amazonaws.com.cn	HTTPS
Tiongkok (Beijing)	cn-north-1	emr-containers.cn-north-1.amazonaws.com.cn	HTTPS
Eropa (Frankfurt)	eu-central-1	emr-containers.eu-central-1.amazonaws.com	HTTPS
Europe (Zurich)	eu-central-2	emr-containers.eu-central-2.amazonaws.com	HTTPS
Eropa (Irlandia)	eu-west-1	emr-containers.eu-west-1.amazonaws.com	HTTPS

Wilayah AWS nama	Kode	Titik akhir	Protokol
Eropa (London)	eu-west-2	emr-containers.eu-west-2.amazonaws.com	HTTPS
Eropa (Paris)	eu-west-3	emr-containers.eu-west-3.amazonaws.com	HTTPS
Eropa (Stockholm)	eu-north-1	emr-containers.eu-north-1.amazonaws.com	HTTPS
Europe (Milan)	eu-south-1	emr-containers.eu-south-1.amazonaws.com	HTTPS
Eropa (Spanyol)	eu-south-2	emr-containers.eu-south-2.amazonaws.com	HTTPS
Israel (Tel Aviv)	il-central-1	emr-containers.il-central-1.amazonaws.com	HTTPS
Amerika Selatan (Sao Paulo)	sa-east-1	emr-containers.sa-east-1.amazonaws.com	HTTPS
Timur Tengah (UAE)	me-central-1	emr-containers.me-central-1.amazonaws.com	HTTPS
Timur Tengah (Bahrain)	me-south-1	emr-containers.me-south-1.amazonaws.com	HTTPS
AWS GovCloud (AS-Timur)	us-gov-east-1	emr-containers.us-gov-east-1.amazonaws.com	HTTPS
AWS GovCloud (AS-Barat)	us-gov-west-1	emr-containers.us-gov-west-1.amazonaws.com	HTTPS

Kuota layanan

Amazon EMR di EKS membatasi permintaan API berikut untuk setiap AWS akun berdasarkan per wilayah. Untuk informasi selengkapnya tentang cara penerapan throttling, lihat [Throttling Permintaan](#)

[API](#) di Referensi API Amazon EC2. Anda dapat meminta peningkatan kuota pembatasan API untuk AWS akun Anda, mengikuti panduan berikut.

Tindakan API	Kapasitas maksimum bucket	Tingkat isi ulang bucket (per detik)
CancelJobRun	25	1
CreateJobTemplate	25	1
CreateManagedEndpoint	25	1
CreateSecurityConfiguration	25	1
CreateVirtualCluster	25	1
DeleteJobTemplate	25	1
DeleteManagedEndpoint	25	1
DeleteVirtualCluster	25	1
DescribeJobRun	100	20
DescribeJobTemplate	25	1
DescribeManagedEndpoint	100	5
DescribeSecurityConfiguration	25	1
DescribeVirtualCluster	100	5
GetManagedEndpoint SessionCredentials	25	1
ListJobRuns	100	5
ListJobTemplates	25	1
ListManagedEndpoints	25	1

Tindakan API	Kapasitas maksimum bucket	Tingkat isi ulang bucket (per detik)
ListSecurityConfigurations	25	1
ListVirtualClusters	100	5
StartJobRun	25	1
Throttle quota for all EMR on EKS API requests	200	20

Saat Anda membuat AWS akun, kami menetapkan kuota default (juga disebut sebagai batas) pada AWS sumber daya Anda per wilayah. Jika Anda mencoba melebihi kuota sumber daya, permintaan gagal. Jika ini terjadi, Anda dapat mengurangi penggunaan sumber daya atau meminta peningkatan kuota.

Konsol Service Quotas adalah lokasi pusat tempat Anda dapat melihat dan mengelola kuota untuk AWS layanan, dan meminta peningkatan kuota untuk banyak sumber daya yang Anda gunakan. Gunakan informasi kuota yang disediakan di sini untuk mengelola AWS infrastruktur Anda. Rencanakan permintaan peningkatan kuota sebelum Anda membutuhkannya.

Lihat kuota dan permintaan kuota meningkat

Anda dapat melihat kuota layanan Anda saat ini untuk setiap Wilayah dengan menggunakan konsol Service Quotas. Untuk petunjuk terperinci, lihat [Melihat kuota Layanan](#) di Panduan Pengguna Service Quotas.

Anda dapat meminta peningkatan kuota baik dari AWS Management Console atau menggunakan AWS CLI. Langkah-langkahnya dirinci dalam [Meminta kenaikan kuota](#).

Amazon EMR pada rilis EKS

Rilis Amazon EMR adalah seperangkat aplikasi sumber terbuka dari ekosistem big data. Setiap rilis terdiri dari aplikasi, komponen, dan fitur big data yang berbeda yang Anda pilih untuk meminta Amazon EMR di EKS menyebarkan dan mengonfigurasi ketika Anda menjalankan tugas berjalan Anda.

Dimulai dengan Amazon EMR rilis 5.32.0 dan 6.2.0, Anda dapat menerapkan Amazon EMR di EKS. Opsi deployment ini tidak tersedia dengan versi rilis Amazon EMR sebelumnya. Anda harus menentukan versi rilis yang didukung ketika Anda mengirimkan tugas Anda.

Amazon EMR di EKS menggunakan bentuk berikut dari label rilis: `emr-x.x.x-latest` atau `emr-x.x.x-yyyymmdd` dengan tanggal rilis tertentu. Misalnya, `emr-7.13.0-latest` atau `emr-7.13.0-20210129`. Saat Anda menggunakan `-latest` akhiran, Anda memastikan bahwa versi EMR Amazon Anda selalu menyertakan pembaruan keamanan terbaru.

Note

Untuk perbandingan antara Amazon EMR di EKS dan Amazon EMR yang berjalan di EC2, lihat FAQ Amazon [EMR](#) di situs web. AWS

Topik

- [AWS runtime untuk Apache Spark \(emr-spark-8.0.0\) di EKS](#)
- [Amazon EMR pada rilis EKS 7.13.0](#)
- [Amazon EMR pada rilis EKS 7.12.0](#)
- [Amazon EMR pada rilis EKS 7.11.0](#)
- [Amazon EMR pada rilis EKS 7.10.0](#)
- [Amazon EMR pada rilis EKS 7.9.0](#)
- [Amazon EMR pada rilis EKS 7.8.0](#)
- [Amazon EMR pada rilis EKS 7.7.0](#)
- [Amazon EMR pada rilis EKS 7.6.0](#)
- [Amazon EMR pada rilis EKS 7.5.0](#)

- [Amazon EMR pada rilis EKS 7.4.0](#)
- [Amazon EMR pada rilis EKS 7.3.0](#)
- [Amazon EMR pada rilis EKS 7.2.0](#)
- [Amazon EMR pada rilis EKS 7.1.0](#)
- [Amazon EMR pada rilis EKS 7.0.0](#)
- [Amazon EMR pada rilis EKS 6.15.0](#)
- [Amazon EMR pada rilis EKS 6.14.0](#)
- [Amazon EMR pada rilis EKS 6.13.0](#)
- [Amazon EMR pada rilis EKS 6.12.0](#)
- [Amazon EMR pada rilis EKS 6.11.0](#)
- [Amazon EMR pada rilis EKS 6.10.0](#)
- [Amazon EMR pada rilis EKS 6.9.0](#)
- [Amazon EMR pada rilis EKS 6.8.0](#)
- [Amazon EMR pada rilis EKS 6.7.0](#)
- [Amazon EMR pada rilis EKS 6.6.0](#)
- [Amazon EMR pada rilis EKS 6.5.0](#)
- [Amazon EMR pada rilis EKS 6.4.0](#)
- [Amazon EMR pada rilis EKS 6.3.0](#)
- [Amazon EMR pada rilis EKS 6.2.0](#)
- [Amazon EMR pada rilis EKS 5.36.0](#)
- [Amazon EMR pada rilis EKS 5.35.0](#)
- [Amazon EMR pada rilis EKS 5.34.0](#)
- [Amazon EMR pada rilis EKS 5.33.0](#)
- [Amazon EMR pada rilis EKS 5.32.0](#)

AWS runtime untuk Apache Spark (emr-spark-8.0.0) di EKS

Halaman ini menjelaskan fungsionalitas baru dan yang diperbarui untuk Amazon EMR yang khusus untuk EMR Amazon pada penerapan EKS. Untuk detail tentang Amazon EMR yang berjalan di

Amazon EC2 dan tentang rilis Amazon EMR Spark 8.0.0 secara umum, [AWS lihat runtime untuk Apache Spark \(emr-spark-8.0.0\)](#) di Panduan Rilis Amazon EMR.

AWS runtime untuk Apache Spark (emr-spark-8.0.0) di EKS

Rilis emr-spark-8.0.0 berikut tersedia untuk AWS runtime untuk Apache Spark di EKS.

- spark/emr-spark-8.0.0-latest
- spark/emr-spark-8.0.0-20260421
- notebook-spark/emr-spark-8.0.0-latest
- notebook-spark/emr-spark-8.0.0-20260421
- notebook-python/emr-spark-8.0.0-latest
- notebook-python/emr-spark-8.0.0-20260421
- livy/emr-spark-8.0.0-latest
- livy/emr-spark-8.0.0-20260421

Catatan rilis

Catatan rilis untuk AWS runtime untuk Apache Spark (emr-spark-8.0.0) di EKS:

- Aplikasi yang didukung - AWS SDK for Java 2.41.32, Apache Spark 4.0.2-amzn-0, Apache Hudi 1.1.0-amzn-0, Apache Iceberg 1.10.1-amzn-0, Delta Lake 4.0.0-amzn-1-spark
- Komponen yang didukung -emr-ddb,emr-goodies,hadoop-client,hoodie,hoodie-spark,iceberg,spark-kubernetes.
- Klasifikasi konfigurasi yang didukung

Untuk digunakan dengan [StartJobRundan](#) dan [CreateManagedEndpointAPI](#):

Klasifikasi	Deskripsi
core-site	Ubah nilai dalam file core-site.xml Hadoop.
spark-metrics	Ubah nilai dalam file metrics.properties Spark.

Klasifikasi	Deskripsi
spark-defaults	Ubah nilai dalam file spark-defaults.conf Spark.
spark-env	Ubah nilai di lingkungan Spark.
spark-hive-site	Ubah nilai dalam file hive-site.xml Spark.
spark-log4j2	Ubah nilai dalam file log4j2.properties Spark.
emr-job-submitter	Konfigurasi untuk pod pengirim pekerjaan .

Untuk digunakan secara khusus dengan [CreateManagedEndpointAPI](#):

Klasifikasi	Deskripsi
jeg-config	Ubah nilai dalam file Jupyter Enterprise Gateway <code>jupyter_enterprise_gateway_config.py</code> .
jupyter-kernel-overrides	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi selengkapnya, lihat [Mengkonfigurasi Aplikasi](#).

Perubahan dan fitur

Fitur-fitur berikut disertakan dengan rilis AWS runtime `emr-spark-8.0.0` untuk Apache Spark di EKS:

- Apache Spark 4.0.2 GA — Rilis Spark 4.x siap produksi pertama di Amazon EMR di EKS, menampilkan mode ANSI SQL, sintaks SQL PIPE, tipe data VARIANT, skrip SQL, dan peningkatan streaming.

- Python 3.11 default — Python 3.11 adalah default untuk beban kerja dan Spark. PySpark Python 3.12 dan 3.13 juga tersedia.

Amazon EMR pada rilis EKS 7.13.0

Halaman ini menjelaskan fungsionalitas baru dan yang diperbarui untuk Amazon EMR yang khusus untuk EMR Amazon pada penerapan EKS. Untuk detail tentang Amazon EMR yang berjalan di Amazon EC2 dan tentang rilis Amazon EMR 7.13.0 secara umum, lihat Amazon EMR 7.13.0 di Panduan [Rilis Amazon EMR](#).

Amazon EMR pada rilis EKS 7.13

Rilis Amazon EMR 7.13.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis EMR-7.13.0-XXXX tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

Flink releases

Rilis Amazon EMR 7.13.0 berikut tersedia untuk Amazon EMR di EKS saat Anda menjalankan aplikasi Flink.

- [emr-7.13.0-flink-terbaru](#)
- [emr-7.13.0-batu pipi-20260410](#)

Spark releases

Rilis Amazon EMR 7.13.0 berikut tersedia untuk Amazon EMR di EKS saat Anda menjalankan aplikasi Spark.

- [emr-7.13.0-terbaru](#)
- [emr-7.13.0-20260410](#)
- emr-7.13.0-spark-rapids-latest
- emr-7.13.0-spark-rapids-20260410
- emr-7.13.0-java11-latest
- emr-7.13.0-java11-20260410
- emr-7.13.0-java8-latest
- emr-7.13.0-java8-20260410
- emr-7.13.0-spark-rapids-java8-latest

- `emr-7.13.0-spark-rapids-java8-20260410`
- `notebook-spark/emr-7.13.0-latest`
- `notebook-spark/emr-7.13.0-20260410`
- `notebook-spark/emr-7.13.0-spark-rapids-latest`
- `notebook-spark/emr-7.13.0-spark-rapids-20260410`
- `notebook-spark/emr-7.13.0-java11-latest`
- `notebook-spark/emr-7.13.0-java11-20260410`
- `notebook-spark/emr-7.13.0-java8-latest`
- `notebook-spark/emr-7.13.0-java8-20260410`
- `notebook-spark/emr-7.13.0-spark-rapids-java8-latest`
- `notebook-spark/emr-7.13.0-spark-rapids-java8-20260410`
- `notebook-python/emr-7.13.0-latest`
- `notebook-python/emr-7.13.0-20260410`
- `notebook-python/emr-7.13.0-spark-rapids-latest`
- `notebook-python/emr-7.13.0-spark-rapids-20260410`
- `notebook-python/emr-7.13.0-java11-latest`
- `notebook-python/emr-7.13.0-java11-20260410`
- `notebook-python/emr-7.13.0-java8-latest`
- `notebook-python/emr-7.13.0-java8-20260410`
- `notebook-python/emr-7.13.0-spark-rapids-java8-latest`
- `notebook-python/emr-7.13.0-spark-rapids-java8-20260410`
- `livy/emr-7.13.0-latest`
- `livy/emr-7.13.0-20260410`
- `livy/emr-7.13.0-java11-latest`
- `livy/emr-7.13.0-java11-20260410`
- `livy/emr-7.13.0-java8-latest`
- `livy/emr-7.13.0-java8-20260410`

Catatan rilis

Catatan rilis untuk Amazon EMR di EKS 7.13.0:

- Aplikasi yang didukung - AWS SDK untuk Java 2.42.12 and 1.12.797, Apache Spark 3.5.6-amzn-2, Apache Hudi 1.0.2-amzn-2, Apache Iceberg 1.10.0-amzn-1, Delta 3.3.2-amzn-2, Apache Spark RAPIDS 25.04.0-amzn-0, Apache Flink 1.20.0-amzn-7
- Komponen yang didukung - `emr-ddbemr-goodies`, `emr-s3-select`, `emrfs`, `hadoop-client`, `hudi`, `hudi-spark`, `iceberg`, `spark-kubernetes`.
- Klasifikasi konfigurasi yang didukung

Untuk digunakan dengan [StartJobRundan](#) dan [CreateManagedEndpointAPI](#):

Klasifikasi	Deskripsi
<code>core-site</code>	Ubah nilai dalam file <code>core-site.xml</code> Hadoop.
<code>emrfs-site</code>	Ubah pengaturan EMRFS.
<code>spark-metrics</code>	Ubah nilai dalam file <code>metrics.properties</code> Spark.
<code>spark-defaults</code>	Ubah nilai dalam file <code>spark-defaults.conf</code> Spark.
<code>spark-env</code>	Ubah nilai di lingkungan Spark.
<code>spark-hive-site</code>	Ubah nilai dalam file <code>hive-site.xml</code> Spark.
<code>spark-log4j2</code>	Ubah nilai dalam file <code>log4j2.properties</code> Spark.
<code>emr-job-submitter</code>	Konfigurasi untuk pod pengirim pekerjaan .

Untuk digunakan secara khusus dengan [CreateManagedEndpointAPI](#):

Klasifikasi	Deskripsi
jeg-config	Ubah nilai dalam file Jupyter Enterprise Gateway <code>jupyter_enterprise_gateway_config.py</code> .
jupyter-kernel-overrides	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi selengkapnya, lihat [Mengkonfigurasi Aplikasi](#).

Perubahan dan fitur

Fitur-fitur berikut disertakan dengan rilis 7.13.0 Amazon EMR di EKS:

- Python 3.11 default untuk PySpark dan beban kerja Spark - Python 3.11 sekarang menjadi versi Python default untuk dan beban kerja Spark. PySpark Python 3.9 tetap menjadi default untuk semua aplikasi lainnya. Baik Python 3.9 dan 3.11 disertakan dalam rilis.

emr-7.13.0-terbaru

Catatan rilis: `emr-7.13.0-latest` saat ini menunjuk ke `emr-7.13.0-20260410`.

Wilayah: `emr-7.13.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.13.0:latest`

emr-7.13.0-20260410

Catatan rilis: `emr-7.13.0-20260410` dirilis pada April 2026. Ini adalah rilis awal Amazon EMR 7.13.0 (Spark).

Wilayah: `emr-7.13.0-20260410` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.13.0-20260410`

emr-7.13.0-flink-terbaru

Catatan rilis: `emr-7.13.0-flink-latest` saat ini menunjuk ke `emr-7.13.0-flink-20260410`

Wilayah: `emr-7.13.0-flink-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.13.0-flink:latest`

emr-7.13.0-batu pipi-20260410

Catatan rilis: `emr-7.13.0-flink-20260410` dirilis pada April 2026. Ini adalah rilis awal Amazon EMR 7.13.0 (Flink).

Wilayah: `emr-7.13.0-flink-20260410` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.13.0-flink:20260410`

Amazon EMR pada rilis EKS 7.12.0

Halaman ini menjelaskan fungsionalitas baru dan yang diperbarui untuk Amazon EMR yang khusus untuk EMR Amazon pada penerapan EKS. Untuk detail tentang Amazon EMR yang berjalan di Amazon EC2 dan tentang rilis Amazon EMR 7.12.0 secara umum, lihat Amazon EMR 7.12.0 di Panduan [Rilis Amazon EMR](#).

Amazon EMR pada rilis EKS 7.12

Rilis Amazon EMR 7.12.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis EMR-7.12.0-XXXX tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

Flink releases

Rilis Amazon EMR 7.12.0 berikut tersedia untuk Amazon EMR di EKS saat Anda menjalankan aplikasi Flink.

- [emr-7.12.0-flink-terbaru](#)
- [emr-7.12.0-batu pipi-20251111](#)

Spark releases

Rilis Amazon EMR 7.12.0 berikut tersedia untuk Amazon EMR di EKS saat Anda menjalankan aplikasi Spark.

- [emr-7.12.0-terbaru](#)
- [emr-7.12.0-20251111](#)
- emr-7.12.0-spark-rapids-latest
- emr-7.12.0-spark-rapids-20251111
- emr-7.12.0-java11-latest
- emr-7.12.0-java11-20251111
- emr-7.12.0-java8-latest
- emr-7.12.0-java8-20251111
- emr-7.12.0-spark-rapids-java8-latest
- emr-7.12.0-spark-rapids-java8-20251111
- notebook-spark/emr-7.12.0-latest
- notebook-spark/emr-7.12.0-20251111
- notebook-spark/emr-7.12.0-spark-rapids-latest
- notebook-spark/emr-7.12.0-spark-rapids-20251111
- notebook-spark/emr-7.12.0-java11-latest
- notebook-spark/emr-7.12.0-java11-20251111
- notebook-spark/emr-7.12.0-java8-latest
- notebook-spark/emr-7.12.0-java8-20251111
- notebook-spark/emr-7.12.0-spark-rapids-java8-latest
- notebook-spark/emr-7.12.0-spark-rapids-java8-20251111
- notebook-python/emr-7.12.0-latest
- notebook-python/emr-7.12.0-20251111
- notebook-python/emr-7.12.0-spark-rapids-latest
- notebook-python/emr-7.12.0-spark-rapids-20251111
- notebook-python/emr-7.12.0-java11-latest
- notebook-python/emr-7.12.0-java11-20251111
- notebook-python/emr-7.12.0-java8-latest

- notebook-python/emr-7.12.0-java8-20251111
- notebook-python/emr-7.12.0-spark-rapids-java8-latest
- notebook-python/emr-7.12.0-spark-rapids-java8-20251111
- livy/emr-7.12.0-latest
- livy/emr-7.12.0-20251111
- livy/emr-7.12.0-java11-latest
- livy/emr-7.12.0-java11-20251111
- livy/emr-7.12.0-java8-latest
- livy/emr-7.12.0-java8-20251111

Catatan rilis

Catatan rilis untuk Amazon EMR di EKS 7.12.0:

- Aplikasi yang didukung - AWS SDK untuk Java 2.35.5 and 1.12.792, Apache Spark 3.5.6-amzn-1, Apache Hudi 1.0.2-amzn-1, Apache Iceberg 1.10.0-amzn-0, Delta 3.3.2-amzn-1, Apache Spark RAPIDS 25.04.0-amzn-0, Apache Flink 1.20.0-amzn-6
- Komponen yang didukung - `emr-ddbemr-goodies`, `emr-s3-select`, `emrfs`, `hadoop-client`, `hudi`, `hudi-spark`, `iceberg`, `spark-kubernetes`.
- Klasifikasi konfigurasi yang didukung

Untuk digunakan dengan [StartJobRundan](#) dan [CreateManagedEndpoint](#) API:

Klasifikasi	Deskripsi
<code>core-site</code>	Ubah nilai dalam file <code>core-site.xml</code> Hadoop.
<code>emrfs-site</code>	Ubah pengaturan EMRFS.
<code>spark-metrics</code>	Ubah nilai dalam file <code>metrics.properties</code> Spark.
<code>spark-defaults</code>	Ubah nilai dalam file <code>spark-defaults.conf</code> Spark.

Klasifikasi	Deskripsi
spark-env	Ubah nilai di lingkungan Spark.
spark-hive-site	Ubah nilai dalam file <code>hive-site.xml</code> Spark.
spark-log4j2	Ubah nilai dalam file <code>log4j2.properties</code> Spark.
emr-job-submitter	Konfigurasi untuk pod pengirim pekerjaan .

Untuk digunakan secara khusus dengan [CreateManagedEndpointAPI](#):

Klasifikasi	Deskripsi
jeg-config	Ubah nilai dalam file Jupyter Enterprise Gateway <code>jupyter_enterprise_gateway_config.py</code> .
jupyter-kernel-overrides	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi selengkapnya, lihat [Mengkonfigurasi Aplikasi](#).

Perubahan dan fitur

Fitur-fitur berikut disertakan dengan rilis 7.12.0 Amazon EMR di EKS:

- Iceberg Materialized Views — Mulai EMR 7.12.0, EMR Spark mendukung pembuatan dan pengelolaan Iceberg Materialized Views (MV).
- Akses Tabel Penuh Hudi - Mulai EMR 7.12.0, EMR sekarang mendukung kontrol Akses Tabel Penuh (FTA) untuk Apache Hudi di Apache Spark berdasarkan kebijakan Anda yang ditentukan

dalam Lake Formation. Fitur ini memungkinkan operasi baca dan tulis dari pekerjaan Amazon EMR Spark Anda di tabel terdaftar Lake Formation saat peran pekerjaan memiliki akses tabel penuh.

- Upgrade versi Iceberg - EMR 7.12.0 mendukung Apache Iceberg versi 1.10.
- Logging untuk beban kerja interaktif Livy - Memulai EMR 7.12.0, EMR mendukung pencatatan ekstensif untuk komponen sistem utama guna meningkatkan pemecahan masalah kegagalan pekerjaan Livy Spark. Fitur ini akan menyediakan layanan EMR dengan akses ke Livy tambahan dan SecretAgent log untuk menyederhanakan pemecahan masalah.

emr-7.12.0-terbaru

Catatan rilis: `emr-7.12.0-latest` saat ini menunjuk ke `emr-7.12.0-20251111`.

Wilayah: `emr-7.12.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.12.0:latest`

emr-7.12.0-20251111

Catatan rilis: `emr-7.12.0-20251111` dirilis pada November 2025. Ini adalah rilis awal Amazon EMR 7.12.0 (Spark).

Wilayah: `emr-emr-7.12.0-20251111` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.12.0-20251111`

emr-7.12.0-flink-terbaru

Catatan rilis: `emr-7.12.0-flink-latest` saat ini menunjuk ke `emr-7.12.0-flink-20251111`

Wilayah: `emr-7.12.0-flink-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.12.0-flink:latest`

emr-7.12.0-batu pipi-20251111

Catatan rilis: `7.12.0-flink-20251111` dirilis pada November 2025. Ini adalah rilis awal Amazon EMR 7.12.0 (Flink).

Wilayah: `emr-7.12.0-flink-20251111` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.12.0-flink:20251111`

Amazon EMR pada rilis EKS 7.11.0

Halaman ini menjelaskan fungsionalitas baru dan yang diperbarui untuk Amazon EMR yang khusus untuk EMR Amazon pada penerapan EKS. Untuk detail tentang Amazon EMR yang berjalan di Amazon EC2 dan tentang rilis Amazon EMR 7.11.0 secara umum, lihat Amazon EMR 7.11.0 di Panduan [Rilis Amazon EMR](#).

Amazon EMR pada rilis EKS 7.11

Rilis Amazon EMR 7.11.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis EMR-7.11.0-XXXX tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

Flink releases

Rilis Amazon EMR 7.11.0 berikut tersedia untuk Amazon EMR di EKS saat Anda menjalankan aplikasi Flink.

- [emr-7.11.0-flink-terbaru](#)
- [emr-7.11.0-batu pipi-20251020](#)

Spark releases

Rilis Amazon EMR 7.11.0 berikut tersedia untuk Amazon EMR di EKS saat Anda menjalankan aplikasi Spark.

- [emr-7.11.0-terbaru](#)
- [emr-7.11.0-20251020](#)
- `emr-7.11.0-spark-rapids-latest`
- `emr-7.11.0-spark-rapids-20251020`
- `emr-7.11.0-java11-latest`
- `emr-7.11.0-java11-20251020`

- `emr-7.11.0-java8-latest`
- `emr-7.11.0-java8-20251020`
- `emr-7.11.0-spark-rapids-java8-latest`
- `emr-7.11.0-spark-rapids-java8-20251020`
- `notebook-spark/emr-7.11.0-latest`
- `notebook-spark/emr-7.11.0-20251020`
- `notebook-spark/emr-7.11.0-spark-rapids-latest`
- `notebook-spark/emr-7.11.0-spark-rapids-20251020`
- `notebook-spark/emr-7.11.0-java11-latest`
- `notebook-spark/emr-7.11.0-java11-20251020`
- `notebook-spark/emr-7.11.0-java8-latest`
- `notebook-spark/emr-7.11.0-java8-20251020`
- `notebook-spark/emr-7.11.0-spark-rapids-java8-latest`
- `notebook-spark/emr-7.11.0-spark-rapids-java8-20251020`
- `notebook-python/emr-7.11.0-latest`
- `notebook-python/emr-7.11.0-20251020`
- `notebook-python/emr-7.11.0-spark-rapids-latest`
- `notebook-python/emr-7.11.0-spark-rapids-20251020`
- `notebook-python/emr-7.11.0-java11-latest`
- `notebook-python/emr-7.11.0-java11-20251020`
- `notebook-python/emr-7.11.0-java8-latest`
- `notebook-python/emr-7.11.0-java8-20251020`
- `notebook-python/emr-7.11.0-spark-rapids-java8-latest`
- `notebook-python/emr-7.11.0-spark-rapids-java8-20251020`
- `livy/emr-7.11.0-latest`
- `livy/emr-7.11.0-20251020`
- `livy/emr-7.11.0-java11-latest`
- `livy/emr-7.11.0-java11-20251020`

- `livy/emr-7.11.0-java8-latest`
- `livy/emr-7.11.0-java8-20251020`

Catatan rilis

Catatan rilis untuk Amazon EMR di EKS 7.11.0:

- Aplikasi yang didukung - AWS SDK untuk Java 2.35.5 and 1.12.792, Apache Spark 3.5.6-amzn-0, Apache Hudi 1.0.2-amzn-0, Apache Iceberg 1.9.1-amzn-0, Delta 3.3.2-amzn-0, Apache Spark RAPIDS 25.04.0-amzn-0, Apache Flink 1.20.0-amzn-5
- Komponen yang didukung - `emr-ddbemr-goodies`, `emr-s3-select`, `emrfs`, `hadoop-client`, `hudi`, `hudi-spark`, `iceberg`, `spark-kubernetes`.
- Klasifikasi konfigurasi yang didukung

Untuk digunakan dengan [StartJobRundan](#) dan [CreateManagedEndpointAPI](#):

Klasifikasi	Deskripsi
<code>core-site</code>	Ubah nilai dalam file <code>core-site.xml</code> Hadoop.
<code>emrfs-site</code>	Ubah pengaturan EMRFS.
<code>spark-metrics</code>	Ubah nilai dalam file <code>metrics.properties</code> Spark.
<code>spark-defaults</code>	Ubah nilai dalam file <code>spark-defaults.conf</code> Spark.
<code>spark-env</code>	Ubah nilai di lingkungan Spark.
<code>spark-hive-site</code>	Ubah nilai dalam file <code>hive-site.xml</code> Spark.
<code>spark-log4j2</code>	Ubah nilai dalam file <code>log4j2.properties</code> Spark.
<code>emr-job-submitter</code>	Konfigurasi untuk pod pengirim pekerjaan .

Untuk digunakan secara khusus dengan [CreateManagedEndpointAPI](#):

Klasifikasi	Deskripsi
jeg-config	Ubah nilai dalam file Jupyter Enterprise Gateway <code>jupyter_enterprise_gateway_config.py</code> .
jupyter-kernel-overrides	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi selengkapnya, lihat [Mengkonfigurasi Aplikasi](#).

Perubahan dan fitur

Perubahan berikut disertakan dengan rilis 7.11.0 Amazon EMR di EKS:

Amazon EMR di EKS sekarang mendukung integrasi dengan SageMaker Unified Studio melalui solusi Livy terkelola termasuk:

- Sesi Terkelola: Jenis sumber daya sesi interaktif baru yang menyediakan titik akhir HTTPS khusus untuk menjalankan sesi Spark di cluster EKS Anda melalui SageMaker Unified Studio
- Integrasi Lake Formation: Mendukung kontrol akses data dengan dua mode a) Kontrol Fine-Grained Akses b) Akses Tabel Penuh (Mode Kompatibilitas)
- Manajemen Identitas: Opsi otentikasi fleksibel a) Kontrol akses berbasis peran IAM b) kontrol akses berbasis.
- Sesi Latar Belakang Pengguna dengan integrasi: Mendukung beban kerja Spark yang berjalan lama untuk terus berjalan bahkan setelah pengguna log off dari SageMaker Unified Studio, mendukung sesi hingga 90 hari

emr-7.11.0-terbaru

Catatan rilis: `emr-7.11.0-latest` saat ini menunjuk ke `emr-7.11.0-20251020`.

Wilayah: `emr-7.11.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.11.0:latest`

emr-7.11.0-20251020

Catatan rilis: `emr-7.11.0-20251020` dirilis pada November 2025. Ini adalah rilis awal Amazon EMR 7.11.0 (Spark).

Wilayah: `emr-emr-7.11.0-20251020` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.11.0-20251020`

emr-7.11.0-flink-terbaru

Catatan rilis: `emr-7.11.0-flink-latest` saat ini menunjuk ke `emr-7.11.0-flink-20251020`

Wilayah: `emr-7.11.0-flink-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.11.0-flink:latest`

emr-7.11.0-batu pipi-20251020

Catatan rilis: `7.11.0-flink-20251020` dirilis pada November 2025. Ini adalah rilis awal Amazon EMR 7.11.0 (Flink).

Wilayah: `emr-7.11.0-flink-20251020` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.11.0-flink:20251020`

Amazon EMR pada rilis EKS 7.10.0

Halaman ini menjelaskan fungsionalitas baru dan yang diperbarui untuk Amazon EMR yang khusus untuk EMR Amazon pada penerapan EKS. Untuk detail tentang Amazon EMR yang berjalan di

Amazon EC2 dan tentang rilis Amazon EMR 7.10.0 secara umum, lihat Amazon EMR 7.10.0 di Panduan [Rilis Amazon EMR](#).

Amazon EMR pada rilis EKS 7.10

Rilis Amazon EMR 7.10.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis EMR-7.10.0-XXXX tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

Flink releases

Rilis Amazon EMR 7.10.0 berikut tersedia untuk Amazon EMR di EKS saat Anda menjalankan aplikasi Flink.

- [emr-7.10.0-flink-terbaru](#)
- [emr-7.10.0-batu pipi-20250801](#)

Spark releases

Rilis Amazon EMR 7.10.0 berikut tersedia untuk Amazon EMR di EKS saat Anda menjalankan aplikasi Spark.

- [emr-7.10.0-terbaru](#)
- [emr-7.10.0-20250801](#)
- emr-7.10.0-spark-rapids-latest
- emr-7.10.0-spark-rapids-20250801
- emr-7.10.0-java11-latest
- emr-7.10.0-java11-20250801
- emr-7.10.0-java8-latest
- emr-7.10.0-java8-20250801
- emr-7.10.0-spark-rapids-java8-latest
- emr-7.10.0-spark-rapids-java8-20250801
- notebook-spark/emr-7.10.0-latest
- notebook-spark/emr-7.10.0-20250801
- notebook-spark/emr-7.10.0-spark-rapids-latest
- notebook-spark/emr-7.10.0-spark-rapids-20250801
- notebook-spark/emr-7.10.0-java11-latest

- notebook-spark/emr-7.10.0-java11-20250801
- notebook-spark/emr-7.10.0-java8-latest
- notebook-spark/emr-7.10.0-java8-20250801
- notebook-spark/emr-7.10.0-spark-rapids-java8-latest
- notebook-spark/emr-7.10.0-spark-rapids-java8-20250801
- notebook-python/emr-7.10.0-latest
- notebook-python/emr-7.10.0-20250801
- notebook-python/emr-7.10.0-spark-rapids-latest
- notebook-python/emr-7.10.0-spark-rapids-20250801
- notebook-python/emr-7.10.0-java11-latest
- notebook-python/emr-7.10.0-java11-20250801
- notebook-python/emr-7.10.0-java8-latest
- notebook-python/emr-7.10.0-java8-20250801
- notebook-python/emr-7.10.0-spark-rapids-java8-latest
- notebook-python/emr-7.10.0-spark-rapids-java8-20250801
- livy/emr-7.10.0-latest
- livy/emr-7.10.0-20250801
- livy/emr-7.10.0-java11-latest
- livy/emr-7.10.0-java11-20250801
- livy/emr-7.10.0-java8-latest
- livy/emr-7.10.0-java8-20250801

Catatan rilis

Catatan rilis untuk Amazon EMR di EKS 7.10.0:

- Aplikasi yang didukung - AWS SDK untuk Java 2.31.48 and 1.12.782, Apache Spark 3.5.5-amzn-1, Apache Hudi 0.15.0-amzn-7, Apache Iceberg 1.8.1-amzn-0, Delta 3.3.0-amzn-2, Apache Spark RAPIDS 25.04.0-amzn-0, Apache Flink 1.20.0-amzn-4, Flink Kubernetes Operator 1.10.0-amzn-4
- Komponen yang didukung - emr-ddbemr-goodies,emr-s3-select,,emrfs,hadoop-client,hudi,hudi-spark,iceberg,spark-kubernetes.
- Klasifikasi konfigurasi yang didukung

Untuk digunakan dengan [StartJobRundan](#) dan [CreateManagedEndpointAPI](#):

Klasifikasi	Deskripsi
core-site	Ubah nilai dalam file <code>core-site.xml</code> Hadoop.
emrfs-site	Ubah pengaturan EMRFS.
spark-metrics	Ubah nilai dalam file <code>metrics.properties</code> Spark.
spark-defaults	Ubah nilai dalam file <code>spark-defaults.conf</code> Spark.
spark-env	Ubah nilai di lingkungan Spark.
spark-hive-site	Ubah nilai dalam file <code>hive-site.xml</code> Spark.
spark-log4j2	Ubah nilai dalam file <code>log4j2.properties</code> Spark.
emr-job-submitter	Konfigurasi untuk pod pengirim pekerjaan .

Untuk digunakan secara khusus dengan [CreateManagedEndpointAPI](#):

Klasifikasi	Deskripsi
jeg-config	Ubah nilai dalam file Jupyter Enterprise Gateway <code>jupyter_enterprise_gateway_config.py</code> .
jupyter-kernel-overrides	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi selengkapnya, lihat [Mengkonfigurasi Aplikasi](#).

Perubahan dan fitur

Fitur-fitur berikut disertakan dengan rilis 7.10.0 Amazon EMR di EKS:

- Sistem file S3A — Mulai dari rilis 7.10.0, sistem file S3A telah menggantikan EMRFS sebagai konektor EMR S3 default. Untuk informasi selengkapnya, lihat [EMR File System \(EMRFS\)](#).

emr-7.10.0-terbaru

Catatan rilis: `emr-7.10.0-latest` saat ini menunjuk ke `emr-7.10.0-20250801`.

Wilayah: `emr-7.10.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.10.0:latest`

emr-7.10.0-20250801

Catatan rilis: `emr-7.10.0-20250801` dirilis pada Februari 2025. Ini adalah rilis awal Amazon EMR 7.10.0 (Spark).

Wilayah: `emr-emr-7.10.0-20250801` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.10.0-20250801`

emr-7.10.0-flink-terbaru

Catatan rilis: `emr-7.10.0-flink-latest` saat ini menunjuk ke `emr-7.10.0-flink-20250801`

Wilayah: `emr-7.10.0-flink-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.10.0-flink:latest`

emr-7.10.0-batu pipi-20250801

Catatan rilis: `7.10.0-flink-20250801` dirilis pada Februari 2025. Ini adalah rilis awal Amazon EMR 7.10.0 (Flink).

Wilayah: `emr-7.10.0-flink-20250801` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.10.0-flink:20250801`

Amazon EMR pada rilis EKS 7.9.0

Halaman ini menjelaskan fungsionalitas baru dan yang diperbarui untuk Amazon EMR yang khusus untuk EMR Amazon pada penerapan EKS. Untuk detail tentang Amazon EMR yang berjalan di Amazon EC2 dan tentang rilis Amazon EMR 7.9.0 secara umum, lihat Amazon EMR 7.9.0 di Panduan [Rilis Amazon EMR](#).

Amazon EMR pada rilis EKS 7.9

Rilis Amazon EMR 7.9.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis `EMR-7.9.0-XXXX` tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

Flink releases

Rilis Amazon EMR 7.9.0 berikut tersedia untuk Amazon EMR di EKS saat Anda menjalankan aplikasi Flink.

- [emr-7.9.0-flink-terbaru](#)
- [emr-7.9.0-batu pipi-20250425](#)

Spark releases

Rilis Amazon EMR 7.9.0 berikut tersedia untuk Amazon EMR di EKS saat Anda menjalankan aplikasi Spark.

- [emr-7.9.0-terbaru](#)
- [emr-7.9.0-20250425](#)
- `emr-7.9.0-spark-rapids-latest`

- `emr-7.9.0-spark-rapids-20250425`
- `emr-7.9.0-java11-latest`
- `emr-7.9.0-java11-20250425`
- `emr-7.9.0-java8-latest`
- `emr-7.9.0-java8-20250425`
- `emr-7.9.0-spark-rapids-java8-latest`
- `emr-7.9.0-spark-rapids-java8-20250425`
- `notebook-spark/emr-7.9.0-latest`
- `notebook-spark/emr-7.9.0-20250425`
- `notebook-spark/emr-7.9.0-spark-rapids-latest`
- `notebook-spark/emr-7.9.0-spark-rapids-20250425`
- `notebook-spark/emr-7.9.0-java11-latest`
- `notebook-spark/emr-7.9.0-java11-20250425`
- `notebook-spark/emr-7.9.0-java8-latest`
- `notebook-spark/emr-7.9.0-java8-20250425`
- `notebook-spark/emr-7.9.0-spark-rapids-java8-latest`
- `notebook-spark/emr-7.9.0-spark-rapids-java8-20250425`
- `notebook-python/emr-7.9.0-latest`
- `notebook-python/emr-7.9.0-20250425`
- `notebook-python/emr-7.9.0-spark-rapids-latest`
- `notebook-python/emr-7.9.0-spark-rapids-20250425`
- `notebook-python/emr-7.9.0-java11-latest`
- `notebook-python/emr-7.9.0-java11-20250425`
- `notebook-python/emr-7.9.0-java8-latest`
- `notebook-python/emr-7.9.0-java8-20250425`
- `notebook-python/emr-7.9.0-spark-rapids-java8-latest`
- `notebook-python/emr-7.9.0-spark-rapids-java8-20250425`
- `livy/emr-7.9.0-latest`

- livy/emr-7.9.0-20250425
- livy/emr-7.9.0-java11-latest
- livy/emr-7.9.0-java11-20250425
- livy/emr-7.9.0-java8-latest
- livy/emr-7.9.0-java8-20250425

Catatan rilis

Catatan rilis untuk Amazon EMR di EKS 7.9.0

- Aplikasi yang didukung - AWS SDK untuk Java 2.31.16 and 1.12.782, Apache Spark 3.5.5, Apache Hudi 0.15.0-amzn-6, Apache Iceberg 1.7.1-amzn-2, Delta 3.3.0-amzn-1, Apache Spark RAPIDS 25.02.1-amzn-0, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.20.0-amzn-3, Flink Operator 1.10.0-amzn-3
- Komponen yang didukung - emr-ddbemr-goodies,emr-s3-select,,emrfs,hadoop-client,hudi,hudi-spark,iceberg,spark-kubernetes.
- Klasifikasi konfigurasi yang didukung

Untuk digunakan dengan [StartJobRundan](#) [CreateManagedEndpointAPI](#):

Klasifikasi	Deskripsi
core-site	Ubah nilai dalam file core-site.xml Hadoop.
emrfs-site	Ubah pengaturan EMRFS.
spark-metrics	Ubah nilai dalam file metrics.properties Spark.
spark-defaults	Ubah nilai dalam file spark-defaults.conf Spark.
spark-env	Ubah nilai di lingkungan Spark.
spark-hive-site	Ubah nilai dalam file hive-site.xml Spark.

Klasifikasi	Deskripsi
spark-log4j2	Ubah nilai dalam file log4j2.properties Spark.
emr-job-submitter	Konfigurasi untuk pod pengirim pekerjaan .

Untuk digunakan secara khusus dengan [CreateManagedEndpoint](#) API:

Klasifikasi	Deskripsi
jeg-config	Ubah nilai dalam file Jupyter Enterprise Gateway <code>jupyter_enterprise_gateway_config.py</code> .
jupyter-kernel-overrides	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi selengkapnya, lihat [Mengkonfigurasi Aplikasi](#).

Perubahan

Perubahan berikut disertakan dengan rilis 7.9.0 Amazon EMR di EKS:

- Tidak ada perubahan untuk rilis.

emr-7.9.0-terbaru

Catatan rilis: `emr-7.9.0-latest` saat ini menunjuk ke `emr-7.9.0-20250425`.

Wilayah: `emr-7.9.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.9.0:latest`

emr-7.9.0-20250425

Catatan rilis: `emr-7.9.0-20250425` dirilis pada Februari 2025. Ini adalah rilis awal Amazon EMR 7.9.0 (Spark).

Wilayah: `emr-emr-7.9.0-20250425` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.9.0-20250425`

emr-7.9.0-flink-terbaru

Catatan rilis: `emr-7.9.0-flink-latest` saat ini menunjuk ke `emr-7.9.0-flink-20250425`

Wilayah: `emr-7.9.0-flink-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.9.0-flink:latest`

emr-7.9.0-batu pipi-20250425

Catatan rilis: `7.9.0-flink-20250425` dirilis pada Februari 2025. Ini adalah rilis awal Amazon EMR 7.9.0 (Flink).

Wilayah: `emr-7.9.0-flink-20250425` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.9.0-flink:20250425`

Amazon EMR pada rilis EKS 7.8.0

Halaman ini menjelaskan fungsionalitas baru dan yang diperbarui untuk Amazon EMR yang khusus untuk EMR Amazon pada penerapan EKS. Untuk detail tentang Amazon EMR yang berjalan di Amazon EC2 dan tentang rilis Amazon EMR 7.8.0 secara umum, lihat Amazon EMR 7.8.0 di Panduan [Rilis Amazon EMR](#).

Amazon EMR pada rilis EKS 7.8

Rilis Amazon EMR 7.8.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis `EMR-7.8.0-XXXX` tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

Flink releases

Rilis Amazon EMR 7.8.0 berikut tersedia untuk Amazon EMR di EKS saat Anda menjalankan aplikasi Flink.

- [emr-7.8.0-flink-terbaru](#)
- [emr-7.8.0-batu pipi-20250228](#)

Spark releases

Rilis Amazon EMR 7.8.0 berikut tersedia untuk Amazon EMR di EKS saat Anda menjalankan aplikasi Spark.

- [emr-7.8.0-terbaru](#)
- [emr-7.8.0-20250228](#)
- emr-7.8.0-spark-rapids-latest
- emr-7.8.0-spark-rapids-20250228
- emr-7.8.0-java11-latest
- emr-7.8.0-java11-20250228
- emr-7.8.0-java8-latest
- emr-7.8.0-java8-20250228
- emr-7.8.0-spark-rapids-java8-latest
- emr-7.8.0-spark-rapids-java8-20250228
- notebook-spark/emr-7.8.0-latest
- notebook-spark/emr-7.8.0-20250228
- notebook-spark/emr-7.8.0-spark-rapids-latest
- notebook-spark/emr-7.8.0-spark-rapids-20250228
- notebook-spark/emr-7.8.0-java11-latest
- notebook-spark/emr-7.8.0-java11-20250228
- notebook-spark/emr-7.8.0-java8-latest
- notebook-spark/emr-7.8.0-java8-20250228
- notebook-spark/emr-7.8.0-spark-rapids-java8-latest

- notebook-spark/emr-7.8.0-spark-rapids-java8-20250228
- notebook-python/emr-7.8.0-latest
- notebook-python/emr-7.8.0-20250228
- notebook-python/emr-7.8.0-spark-rapids-latest
- notebook-python/emr-7.8.0-spark-rapids-20250228
- notebook-python/emr-7.8.0-java11-latest
- notebook-python/emr-7.8.0-java11-20250228
- notebook-python/emr-7.8.0-java8-latest
- notebook-python/emr-7.8.0-java8-20250228
- notebook-python/emr-7.8.0-spark-rapids-java8-latest
- notebook-python/emr-7.8.0-spark-rapids-java8-20250228
- livy/emr-7.8.0-latest
- livy/emr-7.8.0-20250228
- livy/emr-7.8.0-java11-latest
- livy/emr-7.8.0-java11-20250228
- livy/emr-7.8.0-java8-latest
- livy/emr-7.8.0-java8-20250228

Catatan rilis

Catatan rilis untuk Amazon EMR di EKS 7.8.0

- Aplikasi yang didukung - AWS SDK untuk Java 2.29.52 and 1.12.780, Apache Spark 3.5.4, Apache Hudi 0.15.0-amzn-5, Apache Iceberg 1.7.1-amzn-1, Delta 3.3.0-amzn-0, Apache Spark RAPIDS 24.12.0-amzn-0, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.20.0-amzn-2, Flink Operator 1.10.0-amzn-2
- Komponen yang didukung - emr-ddbemr-goodies,emr-s3-select,,emrfs,hadoop-client,hudi,hudi-spark,iceberg,spark-kubernetes.
- Klasifikasi konfigurasi yang didukung

Untuk digunakan dengan [StartJobRundan](#) [CreateManagedEndpoint](#)API:

Klasifikasi	Deskripsi
core-site	Ubah nilai dalam file <code>core-site.xml</code> Hadoop.
emrfs-site	Ubah pengaturan EMRFS.
spark-metrics	Ubah nilai dalam file <code>metrics.properties</code> Spark.
spark-defaults	Ubah nilai dalam file <code>spark-defaults.conf</code> Spark.
spark-env	Ubah nilai di lingkungan Spark.
spark-hive-site	Ubah nilai dalam file <code>hive-site.xml</code> Spark.
spark-log4j2	Ubah nilai dalam file <code>log4j2.properties</code> Spark.
emr-job-submitter	Konfigurasi untuk pod pengirim pekerjaan .

Untuk digunakan secara khusus dengan [CreateManagedEndpointAPI](#):

Klasifikasi	Deskripsi
jeg-config	Ubah nilai dalam file Jupyter Enterprise Gateway <code>jupyter_enterprise_gateway_config.py</code> .
jupyter-kernel-overrides	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi selengkapnya, lihat [Mengkonfigurasi Aplikasi](#).

Perubahan

Perubahan berikut disertakan dengan rilis 7.8.0 Amazon EMR di EKS:

- Native-FGAC fitur, termasuk:
 - Dukungan Iceberg untuk menjalankan pekerjaan yang melakukan tindakan pada Tabel Non-Lake Formasi dalam kluster virtual kontrol akses halus (FGAC). (Ada fallback ke IAM.)
 - Dukungan tabel S3
- Spark terhubung

emr-7.8.0-terbaru

Catatan rilis: `emr-7.8.0-latest` saat ini menunjuk ke `emr-7.8.0-20250228`.

Wilayah: `emr-7.8.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.8.0:latest`

emr-7.8.0-20250228

Catatan rilis: `emr-7.8.0-20250228` dirilis pada Februari 2025. Ini adalah rilis awal Amazon EMR 7.8.0 (Spark).

Wilayah: `emr-emr-7.8.0-20250228` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.8.0-20250228`

emr-7.8.0-flink-terbaru

Catatan rilis: `emr-7.8.0-flink-latest` saat ini menunjuk ke `emr-7.8.0-flink-20250228`

Wilayah: `emr-7.8.0-flink-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.8.0-flink:latest`

emr-7.8.0-batu pipi-20250228

Catatan rilis: `7.8.0-flink-20250228` dirilis pada Februari 2025. Ini adalah rilis awal Amazon EMR 7.8.0 (Flink).

Wilayah: `emr-7.8.0-flink-20250228` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.8.0-flink:20250228`

Amazon EMR pada rilis EKS 7.7.0

Halaman ini menjelaskan fungsionalitas baru dan yang diperbarui untuk Amazon EMR yang khusus untuk EMR Amazon pada penerapan EKS. Untuk detail tentang Amazon EMR yang berjalan di Amazon EC2 dan tentang rilis Amazon EMR 7.7.0 secara umum, lihat Amazon EMR 7.7.0 di Panduan [Rilis Amazon EMR](#).

Amazon EMR pada rilis EKS 7.7

Rilis Amazon EMR 7.7.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis `EMR-7.7.0-XXXX` tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

Flink releases

Rilis Amazon EMR 7.7.0 berikut tersedia untuk Amazon EMR di EKS saat Anda menjalankan aplikasi Flink.

- [emr-7.7.0-flink-terbaru](#)
- [emr-7.7.0-batu pipi-20250131](#)

Spark releases

Rilis Amazon EMR 7.7.0 berikut tersedia untuk Amazon EMR di EKS saat Anda menjalankan aplikasi Spark.

- [emr-7.7.0-terbaru](#)
- [emr-7.7.0-20250131](#)
- emr-7.7.0-spark-rapids-latest
- emr-7.7.0-spark-rapids-20250131
- emr-7.7.0-java11-latest
- emr-7.7.0-java11-20250131
- emr-7.7.0-java8-latest
- emr-7.7.0-java8-20250131
- emr-7.7.0-spark-rapids-java8-latest
- emr-7.7.0-spark-rapids-java8-20250131
- notebook-spark/emr-7.7.0-latest
- notebook-spark/emr-7.7.0-20250131
- notebook-spark/emr-7.7.0-spark-rapids-latest
- notebook-spark/emr-7.7.0-spark-rapids-20250131
- notebook-spark/emr-7.7.0-java11-latest
- notebook-spark/emr-7.7.0-java11-20250131
- notebook-spark/emr-7.7.0-java8-latest
- notebook-spark/emr-7.7.0-java8-20250131
- notebook-spark/emr-7.7.0-spark-rapids-java8-latest
- notebook-spark/emr-7.7.0-spark-rapids-java8-20250131
- notebook-python/emr-7.7.0-latest
- notebook-python/emr-7.7.0-20250131
- notebook-python/emr-7.7.0-spark-rapids-latest
- notebook-python/emr-7.7.0-spark-rapids-20250131
- notebook-python/emr-7.7.0-java11-latest
- notebook-python/emr-7.7.0-java11-20250131
- notebook-python/emr-7.7.0-java8-latest
- notebook-python/emr-7.7.0-java8-20250131
- notebook-python/emr-7.7.0-spark-rapids-java8-latest

- notebook-python/emr-7.7.0-spark-rapids-java8-20250131
- livy/emr-7.7.0-latest
- livy/emr-7.7.0-20250131
- livy/emr-7.7.0-java11-latest
- livy/emr-7.7.0-java11-20250131
- livy/emr-7.7.0-java8-latest
- livy/emr-7.7.0-java8-20250131

Catatan rilis

Catatan rilis untuk Amazon EMR di EKS 7.7.0

- Aplikasi yang didukung - AWS SDK untuk Java 2.29.25 and 1.12.779, Apache Spark 3.5.3-amzn-0, Apache Hudi 0.15.0-amzn-3, Apache Iceberg 1.6.1-amzn-2, Delta 3.2.1-amzn-1, Apache Spark RAPIDS 24.10.1-amzn-0, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.20.0-amzn-0, Flink Operator 1.10.0-amzn-0
- Komponen yang didukung - emr-ddbemr-goodies,emr-s3-select,,emrfs,hadoop-client,hudi,hudi-spark,iceberg,spark-kubernetes.
- Klasifikasi konfigurasi yang didukung

Untuk digunakan dengan [StartJobRundan](#) dan [CreateManagedEndpointAPI](#):

Klasifikasi	Deskripsi
core-site	Ubah nilai dalam file core-site.xml Hadoop.
emrfs-site	Ubah pengaturan EMRFS.
spark-metrics	Ubah nilai dalam file metrics.properties Spark.
spark-defaults	Ubah nilai dalam file spark-defaults.conf Spark.
spark-env	Ubah nilai di lingkungan Spark.

Klasifikasi	Deskripsi
spark-hive-site	Ubah nilai dalam file <code>hive-site.xml</code> Spark.
spark-log4j2	Ubah nilai dalam file <code>log4j2.properties</code> Spark.
emr-job-submitter	Konfigurasi untuk pod pengirim pekerjaan .

Untuk digunakan secara khusus dengan [CreateManagedEndpoint](#) API:

Klasifikasi	Deskripsi
jeg-config	Ubah nilai dalam file Jupyter Enterprise Gateway <code>jupyter_enterprise_gateway_config.py</code> .
jupyter-kernel-overrides	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi selengkapnya, lihat [Mengkonfigurasi Aplikasi](#).

Perubahan

Perubahan berikut disertakan dengan rilis 7.7.0 Amazon EMR di EKS:

- Versi Iceberg yang digunakan pada EMR 7.7.0 tidak lagi mendukung Java 8. Selain itu, Iceberg dikecualikan dari gambar Java 8 berikut: `emr-7.7.0-java8-latest` dan `emr-7.7.0-spark-rapids-java8-latest`

emr-7.7.0-terbaru

Catatan rilis: `emr-7.7.0-latest` saat ini menunjuk ke `emr-7.7.0-20250131`.

Wilayah: `emr-7.7.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.7.0:latest`

`emr-7.7.0-20250131`

Catatan rilis: `emr-7.7.0-20250131` dirilis pada Februari 2025. Ini adalah rilis awal Amazon EMR 7.7.0 (Spark).

Wilayah: `emr-emr-7.7.0-20250131` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.7.0-20250131`

`emr-7.7.0-flink-terbaru`

Catatan rilis: `emr-7.7.0-flink-latest` saat ini menunjuk ke `emr-7.7.0-flink-20250131`

Wilayah: `emr-7.7.0-flink-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.7.0-flink:latest`

`emr-7.7.0-batu pipi-20250131`

Catatan rilis: `7.7.0-flink-20250131` dirilis pada Februari 2025. Ini adalah rilis awal Amazon EMR 7.7.0 (Flink).

Wilayah: `emr-7.7.0-flink-20250131` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.7.0-flink:20250131`

Amazon EMR pada rilis EKS 7.6.0

Halaman ini menjelaskan fungsionalitas baru dan yang diperbarui untuk Amazon EMR yang khusus untuk EMR Amazon pada penerapan EKS. Untuk detail tentang Amazon EMR yang berjalan di Amazon EC2 dan tentang rilis Amazon EMR 7.6.0 secara umum, lihat Amazon EMR 7.6.0 di Panduan [Rilis Amazon EMR](#).

Amazon EMR pada rilis EKS 7.6

Rilis Amazon EMR 7.6.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis EMR-7.6.0-XXXX tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

Flink releases

Rilis Amazon EMR 7.6.0 berikut tersedia untuk Amazon EMR di EKS saat Anda menjalankan aplikasi Flink.

- [emr-7.6.0-flink-terbaru](#)
- [emr-7.6.0-batu pipi-20241213](#)

Spark releases

Rilis Amazon EMR 7.6.0 berikut tersedia untuk Amazon EMR di EKS saat Anda menjalankan aplikasi Spark.

- [emr-7.6.0-terbaru](#)
- [emr-7.6.0-20241213](#)
- emr-7.6.0-spark-rapids-latest
- emr-7.6.0-spark-rapids-20241213
- emr-7.6.0-java11-latest
- emr-7.6.0-java11-20241213
- emr-7.6.0-java8-latest
- emr-7.6.0-java8-20241213
- emr-7.6.0-spark-rapids-java8-latest
- emr-7.6.0-spark-rapids-java8-20241213
- notebook-spark/emr-7.6.0-latest
- notebook-spark/emr-7.6.0-20241213
- notebook-spark/emr-7.6.0-spark-rapids-latest
- notebook-spark/emr-7.6.0-spark-rapids-20241213
- notebook-spark/emr-7.6.0-java11-latest
- notebook-spark/emr-7.6.0-java11-20241213
- notebook-spark/emr-7.6.0-java8-latest

- notebook-spark/emr-7.6.0-java8-20241213
- notebook-spark/emr-7.6.0-spark-rapids-java8-latest
- notebook-spark/emr-7.6.0-spark-rapids-java8-20241213
- notebook-python/emr-7.6.0-latest
- notebook-python/emr-7.6.0-20241213
- notebook-python/emr-7.6.0-spark-rapids-latest
- notebook-python/emr-7.6.0-spark-rapids-20241213
- notebook-python/emr-7.6.0-java11-latest
- notebook-python/emr-7.6.0-java11-20241213
- notebook-python/emr-7.6.0-java8-latest
- notebook-python/emr-7.6.0-java8-20241213
- notebook-python/emr-7.6.0-spark-rapids-java8-latest
- notebook-python/emr-7.6.0-spark-rapids-java8-20241213
- livy/emr-7.6.0-latest
- livy/emr-7.6.0-20241213
- livy/emr-7.6.0-java11-latest
- livy/emr-7.6.0-java11-20241213
- livy/emr-7.6.0-java8-latest
- livy/emr-7.6.0-java8-20241213

Catatan rilis

Catatan rilis untuk Amazon EMR di EKS 7.6.0

- Aplikasi yang didukung - AWS SDK untuk Java 2.29.25 and 1.12.779, Apache Spark 3.5.3-amzn-0, Apache Hudi 0.15.0-amzn-3, Apache Iceberg 1.6.1-amzn-2, Delta 3.2.1-amzn-1, Apache Spark RAPIDS 24.10.1-amzn-0, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.20.0-amzn-0, Flink Operator 1.10.0-amzn-0
- Komponen yang didukung - aws-sagemaker-spark-sdk,emr-ddb,emr-goodies,,emr-s3-select,emrfs,hadoop-client,hudi,hudi-spark,iceberg,spark-kubernetes.
- Klasifikasi konfigurasi yang didukung

Untuk digunakan dengan [StartJobRundan](#) dan [CreateManagedEndpoint](#) API:

Klasifikasi	Deskripsi
core-site	Ubah nilai dalam file <code>core-site.xml</code> Hadoop.
emrfs-site	Ubah pengaturan EMRFS.
spark-metrics	Ubah nilai dalam file <code>metrics.properties</code> Spark.
spark-defaults	Ubah nilai dalam file <code>spark-defaults.conf</code> Spark.
spark-env	Ubah nilai di lingkungan Spark.
spark-hive-site	Ubah nilai dalam file <code>hive-site.xml</code> Spark.
spark-log4j2	Ubah nilai dalam file <code>log4j2.properties</code> Spark.
emr-job-submitter	Konfigurasi untuk pod pengirim pekerjaan .

Untuk digunakan secara khusus dengan [CreateManagedEndpointAPI](#):

Klasifikasi	Deskripsi
jeg-config	Ubah nilai dalam file Jupyter Enterprise Gateway <code>jupyter_enterprise_gateway_config.py</code> .
jupyter-kernel-overrides	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali disesuaikan dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi selengkapnya, lihat [Mengkonfigurasi Aplikasi](#).

Fitur penting

Fitur-fitur berikut disertakan dengan rilis 7.6.0 Amazon EMR di EKS:

- **Monitoring Configuration Support untuk Apache Spark Operator** - Konfigurasi pemantauan memungkinkan Anda dengan mudah mengatur pengarsipan log aplikasi Spark dan log operator Anda ke Amazon S3 atau ke Amazon CloudWatch Anda dapat memilih salah satu atau keduanya. Melakukan hal itu menambahkan sespan agen log ke pod operator Spark, driver, dan pod pelaksana Anda, dan selanjutnya meneruskan log komponen ini ke sink yang dikonfigurasi. Untuk informasi selengkapnya, lihat [Menggunakan konfigurasi pemantauan untuk memantau operator Spark Kubernetes dan pekerjaan Spark](#).

Perubahan

Perubahan berikut disertakan dengan rilis 7.6.0 Amazon EMR di EKS:

- Tidak ada perubahan untuk rilis.

emr-7.6.0-terbaru

Catatan rilis: `emr-7.6.0-latest` saat ini menunjuk ke `emr-7.6.0-20241213`.

Wilayah: `emr-7.6.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.6.0:latest`

emr-7.6.0-20241213

Catatan rilis: `7.6.0-20241213` dirilis pada Januari 2024. Ini adalah rilis awal Amazon EMR 7.6.0 (Spark).

Wilayah: `emr-7.6.0-20241213` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.6.0:20241213`

emr-7.6.0-flink-terbaru

Catatan rilis: `emr-7.6.0-flink-latest` saat ini menunjuk ke `emr-7.6.0-flink-20241213`

Wilayah: `emr-7.6.0-flink-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.6.0-flink:latest`

emr-7.6.0-batu pipi-20241213

Catatan rilis: `7.6.0-flink-20241213` dirilis pada Januari 2024. Ini adalah rilis awal Amazon EMR 7.6.0 (Flink).

Wilayah: `emr-7.6.0-flink-20241213` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.6.0-flink:20241213`

Amazon EMR pada rilis EKS 7.5.0

Halaman ini menjelaskan fungsionalitas baru dan yang diperbarui untuk Amazon EMR yang khusus untuk EMR Amazon pada penerapan EKS. Untuk detail tentang Amazon EMR yang berjalan di Amazon EC2 dan tentang rilis Amazon EMR 7.5.0 secara umum, lihat Amazon EMR 7.5.0 di Panduan Rilis Amazon [EMR](#).

Amazon EMR pada rilis EKS 7.5

Rilis Amazon EMR 7.5.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis `EMR-7.5.0-XXXX` tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

Catatan rilis

Catatan rilis untuk Amazon EMR di EKS 7.5.0

- Aplikasi yang didukung - AWS SDK untuk Java 2.28.8 and 1.12.772, Apache Spark 3.5.2-amzn-1, Apache Hudi 0.15.0-amzn-1, Apache Iceberg 1.6.1-amzn-0, Delta 3.2.0-amzn-1, Apache Spark RAPIDS 24.08.1-amzn-1, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.19.1-amzn-1, Flink Operator 1.9.0-amzn-0
- Komponen yang didukung - `aws-sagemaker-spark-sdkemr-ddb`, `emr-goodies`, `emr-s3-select`, `emrfs`, `hadoop-client`, `hudi`, `hudi-spark`, `iceberg`, `spark-kubernetes`.

Amazon EMR pada rilis EKS 7.4.0

Halaman ini menjelaskan fungsionalitas baru dan yang diperbarui untuk Amazon EMR yang khusus untuk EMR Amazon pada penerapan EKS. Untuk detail tentang Amazon EMR yang berjalan di Amazon EC2 dan tentang rilis Amazon EMR 7.4.0 secara umum, lihat Amazon EMR 7.4.0 di Panduan Rilis Amazon [EMR](#).

Amazon EMR pada rilis EKS 7.4

Rilis Amazon EMR 7.4.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis EMR-7.4.0-XXXX tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

Catatan rilis

Catatan rilis untuk Amazon EMR di EKS 7.4.0

- Aplikasi yang didukung - AWS SDK untuk Java 2.25.70 and 1.12.772, Apache Spark 3.5.2-amzn-0, Apache Hudi 0.15.0-amzn-1, Apache Iceberg 1.6.1-amzn-0, Delta 3.2.0-amzn-1, Apache Spark RAPIDS 24.08.1-amzn-0, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.19.1-amzn-0, Flink Operator 1.9.0-amzn-1
- Komponen yang didukung - `aws-sagemaker-spark-sdkemr-ddb`, `emr-goodies`, `emr-s3-select`, `emrfs`, `hadoop-client`, `hudi`, `hudi-spark`, `iceberg`, `spark-kubernetes`.

Amazon EMR pada rilis EKS 7.3.0

Halaman ini menjelaskan fungsionalitas baru dan yang diperbarui untuk Amazon EMR yang khusus untuk EMR Amazon pada penerapan EKS. Untuk detail tentang Amazon EMR yang berjalan di Amazon EC2 dan tentang rilis Amazon EMR 7.3.0 secara umum, lihat Amazon EMR 7.3.0 di Panduan Rilis Amazon [EMR](#).

Amazon EMR pada rilis EKS 7.3

Rilis Amazon EMR 7.3.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis EMR-7.3.0-XXXX tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

Flink releases

Rilis Amazon EMR 7.3.0 berikut tersedia untuk Amazon EMR di EKS saat Anda menjalankan aplikasi Flink.

- [emr-7.3.0-flink-terbaru](#)
- [emr-7.3.0-batu pipi-29240920](#)

Spark releases

Rilis Amazon EMR 7.3.0 berikut tersedia untuk Amazon EMR di EKS saat Anda menjalankan aplikasi Spark.

- [emr-7.3.0-terbaru](#)
- [emr-7.3.0-20240920](#)
- emr-7.3.0-spark-rapids-latest
- emr-7.3.0-spark-rapids-29240920
- emr-7.3.0-java11-latest
- emr-7.3.0-java11-29240920
- emr-7.3.0-java8-latest
- emr-7.3.0-java8-29240920
- emr-7.3.0-spark-rapids-java8-latest
- emr-7.3.0-spark-rapids-java8-29240920
- notebook-spark/emr-7.3.0-latest
- notebook-spark/emr-7.3.0-20240920
- notebook-spark/emr-7.3.0-spark-rapids-latest
- notebook-spark/emr-7.3.0-spark-rapids-29240920
- notebook-spark/emr-7.3.0-java11-latest
- notebook-spark/emr-7.3.0-java11-29240920
- notebook-spark/emr-7.3.0-java8-latest

- notebook-spark/emr-7.3.0-java8-29240920
- notebook-spark/emr-7.3.0-spark-rapids-java8-latest
- notebook-spark/emr-7.3.0-spark-rapids-java8-29240920
- notebook-python/emr-7.3.0-latest
- notebook-python/emr-7.3.0-20240920
- notebook-python/emr-7.3.0-spark-rapids-latest
- notebook-python/emr-7.3.0-spark-rapids-29240920
- notebook-python/emr-7.3.0-java11-latest
- notebook-python/emr-7.3.0-java11-29240920
- notebook-python/emr-7.3.0-java8-latest
- notebook-python/emr-7.3.0-java8-29240920
- notebook-python/emr-7.3.0-spark-rapids-java8-latest
- notebook-python/emr-7.3.0-spark-rapids-java8-29240920
- livy/emr-7.3.0-latest
- livy/emr-7.3.0-20240920
- livy/emr-7.3.0-java11-latest
- livy/emr-7.3.0-java11-29240920
- livy/emr-7.3.0-java8-latest
- livy/emr-7.3.0-java8-29240920

Catatan rilis

Catatan rilis untuk Amazon EMR di EKS 7.3.0

- Aplikasi yang didukung - AWS SDK untuk Java 2.25.70 and 1.12.747, Apache Spark 3.5.1-amzn-1, Apache Hudi 0.15.0-amzn-0, Apache Iceberg 1.5.2-amzn-0, Delta 3.2.0-amzn-0, Apache Spark RAPIDS 24.06.1-amzn-0, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.18.1-amzn-2, Flink Operator 1.9.0-amzn-0
- Komponen yang didukung - aws-sagemaker-spark-sdk,emr-ddb,emr-goodies,,emr-s3-select,emrfs,hadoop-client,hudi,hudi-spark,iceberg,spark-kubernetes.
- Klasifikasi konfigurasi yang didukung

Untuk digunakan dengan [StartJobRundan](#) dan [CreateManagedEndpoint](#)API:

Klasifikasi	Deskripsi
core-site	Ubah nilai dalam file <code>core-site.xml</code> Hadoop.
emrfs-site	Ubah pengaturan EMRFS.
spark-metrics	Ubah nilai dalam file <code>metrics.properties</code> Spark.
spark-defaults	Ubah nilai dalam file <code>spark-defaults.conf</code> Spark.
spark-env	Ubah nilai di lingkungan Spark.
spark-hive-site	Ubah nilai dalam file <code>hive-site.xml</code> Spark.
spark-log4j2	Ubah nilai dalam file <code>log4j2.properties</code> Spark.
emr-job-submitter	Konfigurasi untuk pod pengirim pekerjaan .

Untuk digunakan secara khusus dengan [CreateManagedEndpointAPI](#):

Klasifikasi	Deskripsi
jeg-config	Ubah nilai dalam file Jupyter Enterprise Gateway <code>jupyter_enterprise_gateway_config.py</code> .
jupyter-kernel-overrides	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi selengkapnya, lihat [Mengkonfigurasi Aplikasi](#).

Fitur penting

Fitur-fitur berikut disertakan dengan rilis 7.3.0 Amazon EMR di EKS.

- Peningkatan aplikasi - Amazon EMR di EKS sekarang [termasuk Operator Flink 1.9.0](#). Selain fitur lainnya, Flink Kubernetes sekarang memungkinkan Anda mengatur kuota CPU dan memori untuk autoscaler.
- Dukungan Apache Iceberg untuk Apache Flink - Apache Iceberg adalah tabel analitik besar format berkinerja tinggi sumber terbuka. Dimulai dengan Amazon EMR 7.3.0, Anda dapat menggunakan tabel Apache Iceberg saat menjalankan Apache Flink di Amazon EMR di EKS. Untuk informasi lebih lanjut, lihat EMR Amazon di EKS [Menggunakan Apache Iceberg dengan Amazon EMR di EKS](#).
- Dukungan Delta Lake untuk Apache Flink — Delta Lake adalah kerangka lapisan penyimpanan untuk arsitektur lakehouse yang biasa dibangun di Amazon S3. Dengan Amazon EMR 7.3.0 dan yang lebih tinggi, Anda dapat menggunakan tabel Delta saat menjalankan Apache Flink di Amazon EMR di EKS. Untuk informasi lebih lanjut, lihat [Menggunakan Delta Lake dengan Amazon EMR di EKS](#).

Perubahan

Perubahan berikut disertakan dengan rilis 7.3.0 Amazon EMR di EKS.

- Dengan Amazon EMR di EKS 7.3.0 dan yang lebih tinggi, Apache Flink sekarang menggunakan runtime Java 17 secara default.

emr-7.3.0-terbaru

Catatan rilis: `emr-7.3.0-latest` saat ini menunjuk ke `emr-7.3.0-20240920`.

Wilayah: `emr-7.3.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.3.0:latest`

`emr-7.3.0-20240920`

Catatan rilis: `7.3.0-20240920` dirilis pada bulan Desember 2023. Ini adalah rilis awal Amazon EMR 7.3.0 (Spark).

Wilayah: `emr-7.3.0-20240920` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.3.0:29240920`

`emr-7.3.0-flink-terbaru`

Catatan rilis: `emr-7.3.0-flink-latest` saat ini menunjuk ke `emr-7.3.0-flink-29240920`.

Wilayah: `emr-7.3.0-flink-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.3.0-flink:latest`

`emr-7.3.0-batu pipi-29240920`

Catatan rilis: `7.3.0-flink-29240920` dirilis pada Desember 2023. Ini adalah rilis awal Amazon EMR 7.3.0 (Flink).

Wilayah: `emr-7.3.0-flink-29240920` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.3.0-flink:29240920`

Amazon EMR pada rilis EKS 7.2.0

Halaman ini menjelaskan fungsionalitas baru dan yang diperbarui untuk Amazon EMR yang khusus untuk EMR Amazon pada penerapan EKS. Untuk detail tentang Amazon EMR yang berjalan di Amazon EC2 dan tentang rilis Amazon EMR 7.2.0 secara umum, lihat Amazon EMR 7.2.0 di Panduan Rilis Amazon [EMR](#).

Amazon EMR pada rilis EKS 7.2

Rilis Amazon EMR 7.2.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis `EMR-7.2.0-XXXX` tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

Flink releases

Rilis Amazon EMR 7.2.0 berikut tersedia untuk Amazon EMR di EKS saat Anda menjalankan aplikasi Flink.

- [emr-7.2.0-flink-terbaru](#)
- [emr-7.2.0-batu pipi-20240610](#)

Spark releases

Rilis Amazon EMR 7.2.0 berikut tersedia untuk Amazon EMR di EKS saat Anda menjalankan aplikasi Spark.

- [emr-7.2.0-terbaru](#)
- [emr-7.2.0-20240610](#)
- emr-7.2.0-spark-rapids-latest
- emr-7.2.0-spark-rapids-20240610
- emr-7.2.0-java11-latest
- emr-7.2.0-java11-20240610
- emr-7.2.0-java8-latest
- emr-7.2.0-java8-20240610
- emr-7.2.0-spark-rapids-java8-latest
- emr-7.2.0-spark-rapids-java8-20240610
- notebook-spark/emr-7.2.0-latest
- notebook-spark/emr-7.2.0-20240610
- notebook-spark/emr-7.2.0-spark-rapids-latest
- notebook-spark/emr-7.2.0-spark-rapids-20240610
- notebook-spark/emr-7.2.0-java11-latest
- notebook-spark/emr-7.2.0-java11-20240610
- notebook-spark/emr-7.2.0-java8-latest
- notebook-spark/emr-7.2.0-java8-20240610
- notebook-spark/emr-7.2.0-spark-rapids-java8-latest

- notebook-spark/emr-7.2.0-spark-rapids-java8-20240610
- notebook-python/emr-7.2.0-latest
- notebook-python/emr-7.2.0-20240610
- notebook-python/emr-7.2.0-spark-rapids-latest
- notebook-python/emr-7.2.0-spark-rapids-20240610
- notebook-python/emr-7.2.0-java11-latest
- notebook-python/emr-7.2.0-java11-20240610
- notebook-python/emr-7.2.0-java8-latest
- notebook-python/emr-7.2.0-java8-20240610
- notebook-python/emr-7.2.0-spark-rapids-java8-latest
- notebook-python/emr-7.2.0-spark-rapids-java8-20240610
- livy/emr-7.2.0-latest
- livy/emr-7.2.0-20240610
- livy/emr-7.2.0-java11-latest
- livy/emr-7.2.0-java11-20240610
- livy/emr-7.2.0-java8-latest
- livy/emr-7.2.0-java8-20240610

Catatan rilis

Catatan rilis untuk Amazon EMR di EKS 7.2.0

- Aplikasi yang didukung - AWS SDK untuk Java 2.23.18 and 1.12.705, Apache Spark 3.5.1-amzn-1, Apache Hudi 0.14.1-amzn-0, Apache Iceberg 1.5.0-amzn-0, Delta 3.1.0, Apache Spark RAPIDS 24.02.0-amzn-1, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.18.1-amzn-0, Flink Operator 1.8.0-amzn-1
- Komponen yang didukung - aws-sagemaker-spark-sdk,emr-ddb,emr-goodies,,emr-s3-select,emrfs,hadoop-client,hudi,hudi-spark,iceberg,spark-kubernetes.
- Klasifikasi konfigurasi yang didukung

Untuk digunakan dengan [StartJobRundan](#) [CreateManagedEndpoint](#)API:

Klasifikasi	Deskripsi
core-site	Ubah nilai dalam file <code>core-site.xml</code> Hadoop.
emrfs-site	Ubah pengaturan EMRFS.
spark-metrics	Ubah nilai dalam file <code>metrics.properties</code> Spark.
spark-defaults	Ubah nilai dalam file <code>spark-defaults.conf</code> Spark.
spark-env	Ubah nilai di lingkungan Spark.
spark-hive-site	Ubah nilai dalam file <code>hive-site.xml</code> Spark.
spark-log4j2	Ubah nilai dalam file <code>log4j2.properties</code> Spark.
emr-job-submitter	Konfigurasi untuk pod pengirim pekerjaan .

Untuk digunakan secara khusus dengan [CreateManagedEndpointAPI](#):

Klasifikasi	Deskripsi
jeg-config	Ubah nilai dalam file Jupyter Enterprise Gateway <code>jupyter_enterprise_gateway_config.py</code> .
jupyter-kernel-overrides	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi selengkapnya, lihat [Mengkonfigurasi Aplikasi](#).

Fitur penting

Fitur-fitur berikut disertakan dengan rilis 7.2.0 Amazon EMR di EKS.

- Peningkatan aplikasi - [Amazon EMR pada peningkatan aplikasi EKS 7.2.0 termasuk Spark 3.5.1, Flink 1.18.1, dan Flink Operator 1.8.0](#).
- [Autoscaler untuk pembaruan Flink](#) - Rilis 7.2.0 menggunakan konfigurasi open source `job.autoscaler.restart.time-tracking.enabled` untuk mengaktifkan estimasi waktu penskalaan ulang, jadi Anda tidak perlu lagi menetapkan nilai empiris secara manual untuk memulai ulang waktu. Jika Anda menjalankan 7.1.0 atau lebih rendah, Anda masih dapat menggunakan penskalaan otomatis Amazon EMR.
- [Apache Hudi integrasi Apache Flink di Amazon EMR](#) di EKS — Rilis ini menambahkan integrasi antara Apache Hudi dan Apache Flink, sehingga Anda dapat menggunakan operator Flink Kubernetes untuk menjalankan pekerjaan Hudi. Hudi memungkinkan Anda menggunakan operasi tingkat rekaman yang dapat Anda gunakan untuk menyederhanakan manajemen data dan pengembangan pipa data.
- [Integrasi Amazon S3 Express One Zone dengan Amazon EMR di EKS](#) - Dengan 7.2.0 dan lebih tinggi, Anda dapat mengunggah data ke S3 Express One Zone dengan Amazon EMR di EKS. S3 Express One Zone adalah kelas penyimpanan Amazon S3 zona tunggal berkinerja tinggi yang memberikan akses data milidetik satu digit yang konsisten untuk sebagian besar aplikasi yang sensitif terhadap latensi. Pada saat rilis, S3 Express One Zone memberikan latensi terendah dan penyimpanan objek cloud kinerja tertinggi di Amazon S3.
- [Support untuk konfigurasi default di operator Spark - Operator](#) Spark di Amazon EKS sekarang mendukung konfigurasi default yang sama dengan model start job run di Amazon EMR di EKS untuk 7.2.0 dan lebih tinggi. Ini berarti bahwa fitur seperti Amazon S3 dan EMRFS tidak lagi memerlukan konfigurasi manual dalam file `yaml`.

emr-7.2.0-terbaru

Catatan rilis: `emr-7.2.0-latest` saat ini menunjuk ke `emr-7.2.0-20240610`.

Wilayah: `emr-7.2.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.2.0:latest`

`emr-7.2.0-20240610`

Catatan rilis: `7.2.0-20240610` dirilis pada bulan Desember 2023. Ini adalah rilis awal Amazon EMR 7.2.0 (Spark).

Wilayah: `emr-7.2.0-20240610` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.2.0:20240610`

`emr-7.2.0-flink-terbaru`

Catatan rilis: `emr-7.2.0-flink-latest` saat ini menunjuk ke `emr-7.2.0-flink-20240610`.

Wilayah: `emr-7.2.0-flink-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.2.0-flink:latest`

`emr-7.2.0-batu pipi-20240610`

Catatan rilis: `7.2.0-flink-20240610` dirilis pada Desember 2023. Ini adalah rilis awal Amazon EMR 7.2.0 (Flink).

Wilayah: `emr-7.2.0-flink-20240610` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.2.0-flink:20240610`

Amazon EMR pada rilis EKS 7.1.0

Halaman ini menjelaskan fungsionalitas baru dan yang diperbarui untuk Amazon EMR yang khusus untuk EMR Amazon pada penerapan EKS. Untuk detail tentang Amazon EMR yang berjalan di Amazon EC2 dan tentang rilis Amazon EMR 7.1.0 secara umum, lihat Amazon EMR 7.1.0 di Panduan Rilis Amazon [EMR](#).

Amazon EMR pada rilis EKS 7.1

Rilis Amazon EMR 7.1.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis EMR-7.1.0-XXXX tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

Flink releases

Rilis Amazon EMR 7.1.0 berikut tersedia untuk Amazon EMR di EKS saat Anda menjalankan aplikasi Flink.

- [emr-7.1.0-flink-terbaru](#)
- [emr-7.1.0-batu pipi-20240321](#)

Spark releases

Rilis Amazon EMR 7.1.0 berikut tersedia untuk Amazon EMR di EKS saat Anda menjalankan aplikasi Spark.

- [emr-7.1.0-terbaru](#)
- [emr-7.1.0-20240321](#)
- emr-7.1.0-spark-rapids-latest
- emr-7.1.0-spark-rapids-20240321
- emr-7.1.0-java11-latest
- emr-7.1.0-java11-20240321
- emr-7.1.0-java8-latest
- emr-7.1.0-java8-20240321
- emr-7.1.0-spark-rapids-java8-latest
- emr-7.1.0-spark-rapids-java8-20240321
- notebook-spark/emr-7.1.0-latest
- notebook-spark/emr-7.1.0-20240321
- notebook-spark/emr-7.1.0-spark-rapids-latest
- notebook-spark/emr-7.1.0-spark-rapids-20240321
- notebook-spark/emr-7.1.0-java11-latest
- notebook-spark/emr-7.1.0-java11-20240321
- notebook-spark/emr-7.1.0-java8-latest

- notebook-spark/emr-7.1.0-java8-20240321
- notebook-spark/emr-7.1.0-spark-rapids-java8-latest
- notebook-spark/emr-7.1.0-spark-rapids-java8-20240321
- notebook-python/emr-7.1.0-latest
- notebook-python/emr-7.1.0-20240321
- notebook-python/emr-7.1.0-spark-rapids-latest
- notebook-python/emr-7.1.0-spark-rapids-20240321
- notebook-python/emr-7.1.0-java11-latest
- notebook-python/emr-7.1.0-java11-20240321
- notebook-python/emr-7.1.0-java8-latest
- notebook-python/emr-7.1.0-java8-20240321
- notebook-python/emr-7.1.0-spark-rapids-java8-latest
- notebook-python/emr-7.1.0-spark-rapids-java8-20240321
- livy/emr-7.1.0-latest
- livy/emr-7.1.0-20240321
- livy/emr-7.1.0-java11-latest
- livy/emr-7.1.0-java11-20240321
- livy/emr-7.1.0-java8-latest
- livy/emr-7.1.0-java8-20240321

Catatan rilis

Catatan rilis untuk Amazon EMR di EKS 7.1.0

- Aplikasi yang didukung - AWS SDK untuk Java 2.23.18 and 1.12.656, Apache Spark 3.5.0-amzn-1, Apache Hudi 0.14.1-amzn-0, Apache Iceberg 1.4.3-amzn-0, Delta 3.0.0, Apache Spark RAPIDS 23.10.0-amzn-1, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.18.1-amzn-0, Flink Operator 1.6.1-amzn-1
- Komponen yang didukung - aws-sagemaker-spark-sdk,emr-ddb,emr-goodies,,emr-s3-select,emrfs,hadoop-client,hudi,hudi-spark,iceberg,spark-kubernetes.
- Klasifikasi konfigurasi yang didukung

Untuk digunakan dengan [StartJobRundan](#) dan [CreateManagedEndpoint](#) API:

Klasifikasi	Deskripsi
core-site	Ubah nilai dalam file <code>core-site.xml</code> Hadoop.
emrfs-site	Ubah pengaturan EMRFS.
spark-metrics	Ubah nilai dalam file <code>metrics.properties</code> Spark.
spark-defaults	Ubah nilai dalam file <code>spark-defaults.conf</code> Spark.
spark-env	Ubah nilai di lingkungan Spark.
spark-hive-site	Ubah nilai dalam file <code>hive-site.xml</code> Spark.
spark-log4j2	Ubah nilai dalam file <code>log4j2.properties</code> Spark.
emr-job-submitter	Konfigurasi untuk pod pengirim pekerjaan .

Untuk digunakan secara khusus dengan [CreateManagedEndpointAPI](#):

Klasifikasi	Deskripsi
jeg-config	Ubah nilai dalam file Jupyter Enterprise Gateway <code>jupyter_enterprise_gateway_config.py</code> .
jupyter-kernel-overrides	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi selengkapnya, lihat [Mengkonfigurasi Aplikasi](#).

Fitur penting

Fitur-fitur berikut disertakan dengan rilis 7.1.0 Amazon EMR di EKS.

- [Dukungan Apache Livy untuk Amazon EMR di EKS - Dengan Amazon EMR pada EKS](#) rilis 7.1.0 dan yang lebih tinggi, Anda dapat menggunakan Apache Livy di kluster Amazon EKS untuk membuat antarmuka Apache Livy REST untuk mengirimkan pekerjaan Spark atau cuplikan kode Spark. Melakukannya memungkinkan Anda mengambil hasil secara sinkron dan asinkron, sambil tetap memanfaatkan Amazon EMR pada manfaat EKS, seperti runtime Amazon EMR-optimized Spark, titik akhir Livy, dan pengalaman penyiapan terprogram. SSL-enabled

emr-7.1.0-terbaru

Catatan rilis: `emr-7.1.0-latest` saat ini menunjuk ke `emr-7.1.0-20240321`.

Wilayah: `emr-7.1.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.1.0:latest`

emr-7.1.0-20240321

Catatan rilis: `7.1.0-20240321` dirilis pada bulan Desember 2023. Ini adalah rilis awal Amazon EMR 7.1.0 (Spark).

Wilayah: `emr-7.1.0-20240321` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.1.0:20240321`

emr-7.1.0-flink-terbaru

Catatan rilis: `emr-7.1.0-flink-latest` saat ini menunjuk ke `emr-7.1.0-flink-20240321`.

Wilayah: `emr-7.1.0-flink-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.1.0-flink:latest`

`emr-7.1.0-batu pipi-20240321`

Catatan rilis: `7.1.0-flink-20240321` dirilis pada Desember 2023. Ini adalah rilis awal Amazon EMR 7.1.0 (Flink).

Wilayah: `emr-7.1.0-flink-20240321` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.1.0-flink:20240321`

Amazon EMR pada rilis EKS 7.0.0

Halaman ini menjelaskan fungsionalitas baru dan yang diperbarui untuk Amazon EMR yang khusus untuk EMR Amazon pada penerapan EKS. Untuk detail tentang Amazon EMR yang berjalan di Amazon EC2 dan tentang rilis Amazon EMR 7.0.0 secara umum, lihat Amazon EMR 7.0.0 di Panduan [Rilis Amazon EMR](#).

Amazon EMR pada rilis EKS 7.0

Rilis Amazon EMR 7.0.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis `EMR-7.0.0-XXXX` tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

Flink releases

Rilis Amazon EMR 7.0.0 berikut tersedia untuk Amazon EMR di EKS saat Anda menjalankan aplikasi Flink.

- [emr-7.0.0-flink-terbaru](#)
- [emr-7.0.0-batu pipi-2024321](#)
- [emr-7.0.0-batu pipi-20231211](#)

Spark releases

Rilis Amazon EMR 7.0.0 berikut tersedia untuk Amazon EMR di EKS saat Anda menjalankan aplikasi Spark.

- [emr-7.0.0-terbaru](#)
- [emr-7.0.0-20231211](#)
- emr-7.0.0-spark-rapids-latest
- emr-7.0.0-spark-rapids-20231211
- emr-7.0.0-java11-latest
- emr-7.0.0-java11-20231211
- emr-7.0.0-java8-latest
- emr-7.0.0-java8-20231211
- emr-7.0.0-spark-rapids-java8-latest
- emr-7.0.0-spark-rapids-java8-20231211
- notebook-spark/emr-7.0.0-latest
- notebook-spark/emr-7.0.0-20231211
- notebook-spark/emr-7.0.0-spark-rapids-latest
- notebook-spark/emr-7.0.0-spark-rapids-20231211
- notebook-spark/emr-7.0.0-java11-latest
- notebook-spark/emr-7.0.0-java11-20231211
- notebook-spark/emr-7.0.0-java8-latest
- notebook-spark/emr-7.0.0-java8-20231211
- notebook-spark/emr-7.0.0-spark-rapids-java8-latest
- notebook-spark/emr-7.0.0-spark-rapids-java8-20231211
- notebook-python/emr-7.0.0-latest
- notebook-python/emr-7.0.0-20231211
- notebook-python/emr-7.0.0-spark-rapids-latest
- notebook-python/emr-7.0.0-spark-rapids-20231211
- notebook-python/emr-7.0.0-java11-latest
- notebook-python/emr-7.0.0-java11-20231211
- notebook-python/emr-7.0.0-java8-latest
- notebook-python/emr-7.0.0-java8-20231211
- notebook-python/emr-7.0.0-spark-rapids-java8-latest
- notebook-python/emr-7.0.0-spark-rapids-java8-20231211

Catatan rilis

Catatan rilis untuk Amazon EMR di EKS 7.0.0

- Aplikasi yang didukung - AWS SDK untuk Java 2.20.160-amzn-0 and 1.12.595, Apache Spark 3.5.0-amzn-0, Apache Flink 1.18.0-amzn-0, Flink Operator 1.6.1, Apache Hudi 0.14.0-amzn-1, Apache Iceberg 1.4.2-amzn-0, Delta 3.0.0, Apache Spark RAPIDS 23.10.0-amzn-0, Jupyter Enterprise Gateway 2.6.0
- Komponen yang didukung - `aws-sagemaker-spark-sdk`, `emr-ddb`, `emr-goodies`, `emr-s3-select`, `emrfs`, `hadoop-client`, `hudi`, `hudi-spark`, `iceberg`, `spark-kubernetes`.
- Klasifikasi konfigurasi yang didukung

Untuk digunakan dengan [StartJobRundan](#) dan [CreateManagedEndpointAPI](#):

Klasifikasi	Deskripsi
<code>core-site</code>	Ubah nilai dalam file <code>core-site.xml</code> Hadoop.
<code>emrfs-site</code>	Ubah pengaturan EMRFS.
<code>spark-metrics</code>	Ubah nilai dalam file <code>metrics.properties</code> Spark.
<code>spark-defaults</code>	Ubah nilai dalam file <code>spark-defaults.conf</code> Spark.
<code>spark-env</code>	Ubah nilai di lingkungan Spark.
<code>spark-hive-site</code>	Ubah nilai dalam file <code>hive-site.xml</code> Spark.
<code>spark-log4j</code>	Ubah nilai dalam file <code>log4j2.properties</code> Spark.
<code>emr-job-submitter</code>	Konfigurasi untuk pod pengirim pekerjaan .

Untuk digunakan secara khusus dengan [CreateManagedEndpointAPI](#):

Klasifikasi	Deskripsi
jeg-config	Ubah nilai dalam file Jupyter Enterprise Gateway <code>jupyter_enterprise_gateway_config.py</code> .
jupyter-kernel-overrides	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi selengkapnya, lihat [Mengkonfigurasi Aplikasi](#).

Fitur penting

Fitur-fitur berikut disertakan dengan rilis 7.0 Amazon EMR di EKS.

- Peningkatan aplikasi - [Amazon EMR pada peningkatan aplikasi EKS 7.0.0 termasuk Spark 3.5, Flink 1.18, dan Flink Operator 1.6.1](#).
- Penyetelan otomatis parameter Flink Autoscaler — Parameter default yang digunakan Flink Autoscaler untuk perhitungan penskalaannya mungkin bukan nilai optimal untuk pekerjaan tertentu. Amazon EMR di EKS 7.0.0 menggunakan tren historis metrik tertentu yang diambil untuk menghitung parameter optimal yang disesuaikan untuk pekerjaan tersebut.

Perubahan

Perubahan berikut disertakan dengan rilis 7.0 Amazon EMR di EKS.

- Amazon Linux 2023 - Dengan Amazon EMR di EKS 7.0.0 dan yang lebih tinggi, semua gambar kontainer didasarkan pada Amazon Linux 2023.
- Spark menggunakan Java 17 sebagai runtime default - Amazon EMR di EKS 7.0.0 Spark menggunakan Java 17 sebagai runtime default. Jika perlu, Anda dapat beralih untuk menggunakan Java 8 atau Java 11 dengan label rilis yang sesuai seperti yang disediakan dalam [Amazon EMR pada rilis EKS 7.0](#) daftar.

emr-7.0.0-terbaru

Catatan rilis: `emr-7.0.0-latest` saat ini menunjuk ke `emr-7.0.0-2024321`.

Wilayah: `emr-7.0.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.0.0:latest`

emr-7.0.0-2024321

Catatan rilis: `7.0.0-2024321` dirilis pada 11 Maret 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-7.0.0-2024321` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.0.0:2024321`

emr-7.0.0-20231211

Catatan rilis: `7.0.0-20231211` dirilis pada bulan Desember 2023. Ini adalah rilis awal Amazon EMR 7.0.0 (Spark).

Wilayah: `emr-7.0.0-20231211` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.0.0:20231211`

emr-7.0.0-flink-terbaru

Catatan rilis: `emr-7.0.0-flink-latest` saat ini menunjuk ke `emr-7.0.0-flink-2024321`.

Wilayah: `emr-7.0.0-flink-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.0.0-flink:latest`

emr-7.0.0-batu pipi-2024321

Catatan rilis: `7.0.0-flink-2024321` dirilis pada 11 Maret 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-7.0.0-flink-2024321` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.0.0-flink:2024321`

emr-7.0.0-batu pipi-20231211

Catatan rilis: `7.0.0-flink-20231211` dirilis pada Desember 2023. Ini adalah rilis awal Amazon EMR 7.0.0 (Flink).

Wilayah: `emr-7.0.0-flink-20231211` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-7.0.0-flink:20231211`

Amazon EMR pada rilis EKS 6.15.0

Halaman ini menjelaskan fungsionalitas baru dan yang diperbarui untuk Amazon EMR yang khusus untuk EMR Amazon pada penerapan EKS. Untuk detail tentang Amazon EMR yang berjalan di Amazon EC2 dan tentang rilis Amazon EMR 6.15.0 secara umum, lihat Amazon EMR 6.15.0 di Panduan [Rilis Amazon EMR](#).

Amazon EMR pada rilis EKS 6.15

Rilis Amazon EMR 6.15.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis `EMR-6.15.0-XXXX` tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

Flink releases

Rilis Amazon EMR 6.15.0 berikut tersedia untuk Amazon EMR di EKS saat Anda menjalankan aplikasi Flink.

- [emr-6.15.0-flink-terbaru](#)
- [emr-6.15.0-batu pipi-20240105](#)
- [emr-6.15.0-batu pipi-20231109](#)

Spark releases

Rilis Amazon EMR 6.15.0 berikut tersedia untuk Amazon EMR di EKS saat Anda menjalankan aplikasi Spark.

- [emr-6.15.0-terbaru](#)
- [emr-6.15.0-20231109](#)
- emr-6.15.0-spark-rapids-latest
- emr-6.15.0-spark-rapids-20231109
- emr-6.15.0-java11-latest
- emr-6.15.0-java11-20231109
- emr-6.15.0-java17-latest
- emr-6.15.0-java17-20231109
- emr-6.15.0-java17-al2023-latest
- emr-6.15.0-java17-al2023-20231109
- emr-6.15.0-spark-rapids-java17-latest
- emr-6.15.0-spark-rapids-java17-20231109
- emr-6.15.0-spark-rapids-java17-al2023-latest
- emr-6.15.0-spark-rapids-java17-al2023-20231109
- notebook-spark/emr-6.15.0-latest
- notebook-spark/emr-6.15.0-20231109
- notebook-spark/emr-6.15.0-spark-rapids-latest
- notebook-spark/emr-6.15.0-spark-rapids-20231109
- notebook-spark/emr-6.15.0-java11-latest
- notebook-spark/emr-6.15.0-java11-20231109
- notebook-spark/emr-6.15.0-java17-latest
- notebook-spark/emr-6.15.0-java17-20231109
- notebook-spark/emr-6.15.0-java17-al2023-latest
- notebook-spark/emr-6.15.0-java17-al2023-20231109
- notebook-python/emr-6.15.0-latest
- notebook-python/emr-6.15.0-20231109
- notebook-python/emr-6.15.0-spark-rapids-latest
- notebook-python/emr-6.15.0-spark-rapids-20231109
- notebook-python/emr-6.15.0-java11-latest

- notebook-python/emr-6.15.0-java11-20231109
- notebook-python/emr-6.15.0-java17-latest
- notebook-python/emr-6.15.0-java17-20231109
- notebook-python/emr-6.15.0-java17-al2023-latest
- notebook-python/emr-6.15.0-java17-al2023-20231109

Catatan rilis

Catatan rilis untuk Amazon EMR di EKS 6.15.0

- Aplikasi yang didukung - AWS SDK untuk Java 1.12.569, Apache Spark 3.4.1-amzn-2, Apache Flink 1.17.1-amzn-1, Apache Hudi 0.14.0-amzn-0, Apache Iceberg 1.4.0-amzn-0, Delta 2.4.0, Apache Spark RAPIDS 23.08.01-amzn-0, Jupyter Enterprise Gateway 2.6.0
- Komponen yang didukung - aws-sagemaker-spark-sdkemr-ddb,emr-goodies,,emr-s3-select,emrfs,hadoop-client,hudi,hudi-spark,iceberg,spark-kubernetes.
- Klasifikasi konfigurasi yang didukung

Untuk digunakan dengan [StartJobRundan](#) dan [CreateManagedEndpointAPI](#):

Klasifikasi	Deskripsi
core-site	Ubah nilai dalam file core-site.xml Hadoop.
emrfs-site	Ubah pengaturan EMRFS.
spark-metrics	Ubah nilai dalam file metrics.properties Spark.
spark-defaults	Ubah nilai dalam file spark-defaults.conf Spark.
spark-env	Ubah nilai di lingkungan Spark.
spark-hive-site	Ubah nilai dalam file hive-site.xml Spark.

Klasifikasi	Deskripsi
spark-log4j	Ubah nilai dalam file log4j2.properties Spark.
emr-job-submitter	Konfigurasi untuk pod pengirim pekerjaan .

Untuk digunakan secara khusus dengan [CreateManagedEndpoint](#) API:

Klasifikasi	Deskripsi
jeg-config	Ubah nilai dalam file Jupyter Enterprise Gateway <code>jupyter_enterprise_gateway_config.py</code> .
jupyter-kernel-overrides	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi selengkapnya, lihat [Mengkonfigurasi Aplikasi](#).

Fitur penting

Fitur-fitur berikut disertakan dengan rilis 6.15 Amazon EMR di EKS.

- [Amazon EMR di EKS dengan Apache Flink](#) - Dengan Amazon EMR di EKS 6.15.0, Anda dapat menjalankan aplikasi Apache Anda bersama dengan jenis Flink-based aplikasi lain di cluster Amazon EKS yang sama. Ini membantu meningkatkan pemanfaatan sumber daya dan menyederhanakan manajemen infrastruktur. Anda dapat memanfaatkan Instans Spot di aplikasi Flink dengan penonaktifan yang anggun, dan mencapai waktu restart yang lebih cepat dengan pemulihan halus dan pemulihan lokal tugas dengan Amazon EBS. Fitur aksesibilitas dan pemantauan mencakup kemampuan untuk meluncurkan aplikasi Flink dengan toples yang disimpan di Amazon S3, akses ke Katalog Data AWS Glue, pemantauan integrasi dengan Amazon S3 dan CloudWatch Amazon, dan rotasi log kontainer.

emr-6.15.0-terbaru

Catatan rilis: `emr-6.15.0-latest` saat ini menunjuk ke `emr-6.15.0-20240105`.

Wilayah: `emr-6.15.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.15.0:latest`

emr-6.15.0-20240105

Catatan rilis: `6.15.0-20240105` dirilis pada 17 Januari 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.15.0-20240105` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.15.0:20240105`

emr-6.15.0-20231109

Catatan rilis: `6.15.0-20231109` dirilis pada 17 November 2023. Ini adalah rilis awal Amazon EMR 6.15.0.

Wilayah: `emr-6.15.0-20231109` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.15.0:20231109`

emr-6.15.0-flink-terbaru

Catatan rilis: `emr-6.15.0-flink-latest` saat ini menunjuk ke `emr-6.15.0-flink-20240105`.

Wilayah: `emr-6.15.0-flink-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.15.0-flink:latest`

emr-6.15.0-batu pipi-20240105

Catatan rilis: 6.15.0-flink-20240105 dirilis pada 17 Januari 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: emr-6.15.0-flink-20240105 tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: emr-6.15.0-flink:20240105

emr-6.15.0-batu pipi-20231109

Catatan rilis: 6.15.0-flink-20231109 dirilis pada 17 November 2023. Ini adalah rilis awal Amazon EMR 6.15.0.

Wilayah: emr-6.15.0-flink-20231109 tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: emr-6.15.0-flink:20231109

Amazon EMR pada rilis EKS 6.14.0

Halaman ini menjelaskan fungsionalitas baru dan yang diperbarui untuk Amazon EMR yang khusus untuk EMR Amazon pada penerapan EKS. Untuk detail tentang Amazon EMR yang berjalan di Amazon EC2 dan tentang rilis Amazon EMR 6.14.0 secara umum, lihat Amazon EMR 6.14.0 di Panduan [Rilis Amazon EMR](#).

Amazon EMR pada rilis EKS 6.14

Rilis Amazon EMR 6.14.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis EMR-6.14.0-XXXX tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

- [emr-6.14.0-terbaru](#)
- [emr-6.14.0-20231005](#)
- emr-6.14.0-spark-rapids-latest
- emr-6.14.0-spark-rapids-20231005

- emr-6.14.0-java11-latest
- emr-6.14.0-java11-20231005
- emr-6.14.0-java17-latest
- emr-6.14.0-java17-20231005
- emr-6.14.0-java17-al2023-latest
- emr-6.14.0-java17-al2023-20231005
- emr-6.14.0-spark-rapids-java17-latest
- emr-6.14.0-spark-rapids-java17-20231005
- emr-6.14.0-spark-rapids-java17-al2023-latest
- emr-6.14.0-spark-rapids-java17-al2023-20231005
- notebook-spark/emr-6.14.0-latest
- notebook-spark/emr-6.14.0-20231005
- notebook-spark/emr-6.14.0-spark-rapids-latest
- notebook-spark/emr-6.14.0-spark-rapids-20231005
- notebook-spark/emr-6.14.0-java11-latest
- notebook-spark/emr-6.14.0-java11-20231005
- notebook-spark/emr-6.14.0-java17-latest
- notebook-spark/emr-6.14.0-java17-20231005
- notebook-spark/emr-6.14.0-java17-al2023-latest
- notebook-spark/emr-6.14.0-java17-al2023-20231005
- notebook-python/emr-6.14.0-latest
- notebook-python/emr-6.14.0-20231005
- notebook-python/emr-6.14.0-spark-rapids-latest
- notebook-python/emr-6.14.0-spark-rapids-20231005
- notebook-python/emr-6.14.0-java11-latest
- notebook-python/emr-6.14.0-java11-20231005
- notebook-python/emr-6.14.0-java17-latest
- notebook-python/emr-6.14.0-java17-20231005

- notebook-python/emr-6.14.0-java17-al2023-latest
- notebook-python/emr-6.14.0-java17-al2023-20231005

Catatan rilis

Catatan rilis untuk Amazon EMR di EKS 6.14.0

- Aplikasi yang didukung - AWS SDK untuk Java 1.12.543, Apache Spark 3.4.1-amzn-1, Apache Hudi 0.13.1-amzn-2, Apache Iceberg 1.3.0-amzn-0, Delta 2.4.0, Apache Spark RAPIDS 23.06.0-amzn-2, Jupyter Enterprise Gateway 2.7.0
- Komponen yang didukung - aws-sagemaker-spark-sdk, emr-ddb, emr-goodies, emr-s3-select, emrfs, hadoop-client, hudi, hudi-spark, iceberg, spark-kubernetes.
- Klasifikasi konfigurasi yang didukung

Untuk digunakan dengan [StartJobRundan](#) dan [CreateManagedEndpointAPI](#):

Klasifikasi	Deskripsi
core-site	Ubah nilai dalam file core-site.xml Hadoop.
emrfs-site	Ubah pengaturan EMRFS.
spark-metrics	Ubah nilai dalam file metrics.properties Spark.
spark-defaults	Ubah nilai dalam file spark-defaults.conf Spark.
spark-env	Ubah nilai di lingkungan Spark.
spark-hive-site	Ubah nilai dalam file hive-site.xml Spark.
spark-log4j	Ubah nilai dalam file log4j2.properties Spark.
emr-job-submitter	Konfigurasi untuk pod pengirim pekerjaan .

Untuk digunakan secara khusus dengan [CreateManagedEndpointAPI](#):

Klasifikasi	Deskripsi
jeg-config	Ubah nilai dalam file Jupyter Enterprise Gateway <code>jupyter_enterprise_gateway_config.py</code> .
jupyter-kernel-overrides	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi selengkapnya, lihat [Mengkonfigurasi Aplikasi](#).

Fitur penting

Fitur-fitur berikut disertakan dengan rilis 6.14 Amazon EMR di EKS.

- Dukungan [Apache Livy](#) - Amazon EMR di EKS sekarang mendukung Apache Livy dengan `spark-submit`

emr-6.14.0-terbaru

Catatan rilis: `emr-6.14.0-latest` saat ini menunjuk ke `emr-6.14.0-20231005`.

Wilayah: `emr-6.14.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.14.0:latest`

emr-6.14.0-20231005

Catatan rilis: `6.14.0-20231005` dirilis pada 17 Oktober 2023. Ini adalah rilis awal Amazon EMR 6.14.0.

Wilayah: `emr-6.14.0-20231005` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.14.0:20231005`

Amazon EMR pada rilis EKS 6.13.0

Halaman ini menjelaskan fungsionalitas baru dan yang diperbarui untuk Amazon EMR yang khusus untuk EMR Amazon pada penerapan EKS. Untuk detail tentang Amazon EMR yang berjalan di Amazon EC2 dan tentang rilis Amazon EMR 6.13.0 secara umum, lihat Amazon EMR 6.13.0 di Panduan [Rilis Amazon EMR](#).

Amazon EMR pada rilis EKS 6.13

Rilis Amazon EMR 6.13.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis EMR-6.13.0-XXXX tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

- [emr-6.13.0-terbaru](#)
- [emr-6.13.0-20230814](#)
- `emr-6.13.0-spark-rapids-latest`
- `emr-6.13.0-spark-rapids-20230814`
- `emr-6.13.0-java11-latest`
- `emr-6.13.0-java11-20230814`
- `emr-6.13.0-java17-latest`
- `emr-6.13.0-java17-20230814`
- `emr-6.13.0-java17-al2023-latest`
- `emr-6.13.0-java17-al2023-20230814`
- `emr-6.13.0-spark-rapids-java17-latest`
- `emr-6.13.0-spark-rapids-java17-20230814`
- `emr-6.13.0-spark-rapids-java17-al2023-latest`
- `emr-6.13.0-spark-rapids-java17-al2023-20230814`
- `notebook-spark/emr-6.13.0-latest`
- `notebook-spark/emr-6.13.0-20230814`

- notebook-spark/emr-6.13.0-spark-rapids-latest
- notebook-spark/emr-6.13.0-spark-rapids-20230814
- notebook-spark/emr-6.13.0-java11-latest
- notebook-spark/emr-6.13.0-java11-20230814
- notebook-spark/emr-6.13.0-java17-latest
- notebook-spark/emr-6.13.0-java17-20230814
- notebook-spark/emr-6.13.0-java17-al2023-latest
- notebook-spark/emr-6.13.0-java17-al2023-20230814
- notebook-python/emr-6.13.0-latest
- notebook-python/emr-6.13.0-20230814
- notebook-python/emr-6.13.0-spark-rapids-latest
- notebook-python/emr-6.13.0-spark-rapids-20230814
- notebook-python/emr-6.13.0-java11-latest
- notebook-python/emr-6.13.0-java11-20230814
- notebook-python/emr-6.13.0-java17-latest
- notebook-python/emr-6.13.0-java17-20230814
- notebook-python/emr-6.13.0-java17-al2023-latest
- notebook-python/emr-6.13.0-java17-al2023-20230814

Catatan rilis

Catatan rilis untuk Amazon EMR di EKS 6.13.0

- Aplikasi yang didukung - AWS SDK untuk Java 1.12.513, Apache Spark 3.4.1-amzn-0, Apache Hudi 0.13.1-amzn-0, Apache Iceberg 1.3.0-amzn-0, Delta 2.4.0, Apache Spark RAPIDS 23.06.0-amzn-1, Jupyter Enterprise Gateway 2.6.0.amzn
- Komponen yang didukung - aws-sagemaker-spark-sdk,emr-ddb,emr-goodies,,emr-s3-select,emrfs,hadoop-client,hudi,hudi-spark,iceberg,spark-kubernetes.
- Klasifikasi konfigurasi yang didukung

Untuk digunakan dengan [StartJobRundan](#) dan [CreateManagedEndpointAPI](#):

Klasifikasi	Deskripsi
core-site	Ubah nilai dalam file <code>core-site.xml</code> Hadoop.
emrfs-site	Ubah pengaturan EMRFS.
spark-metrics	Ubah nilai dalam file <code>metrics.properties</code> Spark.
spark-defaults	Ubah nilai dalam file <code>spark-defaults.conf</code> Spark.
spark-env	Ubah nilai di lingkungan Spark.
spark-hive-site	Ubah nilai dalam file <code>hive-site.xml</code> Spark.
spark-log4j	Ubah nilai dalam file <code>log4j2.properties</code> Spark.
emr-job-submitter	Konfigurasi untuk pod pengirim pekerjaan .

Untuk digunakan secara khusus dengan [CreateManagedEndpointAPI](#):

Klasifikasi	Deskripsi
jeg-config	Ubah nilai dalam file Jupyter Enterprise Gateway <code>jupyter_enterprise_gateway_config.py</code> .
jupyter-kernel-overrides	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi selengkapnya, lihat [Mengkonfigurasi Aplikasi](#).

Fitur penting

Fitur-fitur berikut disertakan dengan rilis 6.13 Amazon EMR di EKS.

- **Amazon Linux 2023** - Dengan Amazon EMR di EKS 6.13 dan lebih tinggi, Anda dapat meluncurkan Spark dengan AL2023 sebagai sistem operasi bersama dengan runtime Java 17. Untuk melakukan ini, gunakan label rilis `al2023` dengan namanya. Sebagai contoh: `emr-6.13.0-java17-al2023-latest`. Kami menyarankan Anda memvalidasi dan menjalankan pengujian kinerja sebelum memindahkan beban kerja produksi ke AL2023 dan Java 17.
- [Amazon EMR di EKS dengan Apache Flink](#) (pratinjau publik) - Amazon EMR di EKS merilis 6.13 dan dukungan yang lebih tinggi Apache Flink, tersedia di pratinjau publik. Dengan peluncuran ini, Anda dapat menjalankan Flink-based aplikasi Apache Anda bersama dengan jenis aplikasi lain di cluster Amazon EKS yang sama. Ini membantu meningkatkan pemanfaatan sumber daya dan menyederhanakan manajemen infrastruktur. Jika Anda sudah menjalankan kerangka kerja data besar di Amazon EKS, Anda sekarang dapat membiarkan Amazon EMR mengotomatiskan penyediaan dan manajemen Anda.

emr-6.13.0-terbaru

Catatan rilis: `emr-6.13.0-latest` saat ini menunjuk ke `emr-6.13.0-20230814`.

Wilayah: `emr-6.13.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.13.0:latest`

emr-6.13.0-20230814

Catatan rilis: `6.13.0-20230814` dirilis pada 7 September 2023. Ini adalah rilis awal Amazon EMR 6.13.0.

Wilayah: `emr-6.13.0-20230814` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.13.0:20230814`

Amazon EMR pada rilis EKS 6.12.0

Halaman ini menjelaskan fungsionalitas baru dan yang diperbarui untuk Amazon EMR yang khusus untuk EMR Amazon pada penerapan EKS. Untuk detail tentang Amazon EMR yang berjalan di Amazon EC2 dan tentang rilis Amazon EMR 6.12.0 secara umum, lihat Amazon EMR 6.12.0 di Panduan [Rilis Amazon EMR](#).

Amazon EMR pada rilis EKS 6.12

Rilis Amazon EMR 6.12.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis EMR-6.12.0-XXXX tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

- [emr-6.12.0-terbaru](#)
- [emr-6.12.0-20240321](#)
- [emr-6.12.0-20230701](#)
- `emr-6.12.0-spark-cepat-terbaru`
- `emr-6.12.0-spark-cepat-20230701`
- `emr-6.12.0-java11-terbaru`
- `emr-6.12.0-java11-20230701`
- `emr-6.12.0-java17-terbaru`
- `emr-6.12.0-java17-20230701`
- `emr-6.12.0-spark-rapids-java17-terbaru`
- `emr-6.12.0-spark-cepat-java17-20230701`
- `notebook- spark/emr -6.12.0-terbaru`
- `notebook- spark/emr -6.12.0-20230701`
- `notebook- spark/emr -6.12.0-spark-cepat-terbaru`
- `notebook- spark/emr -6.12.0-spark-rapids-20230701`
- `notebook- python/emr -6.12.0-terbaru`
- `notebook- python/emr -6.12.0-20230701`
- `notebook- python/emr -6.12.0-spark-cepat-terbaru`
- `notebook- python/emr -6.12.0-spark-rapids-20230701`

Catatan rilis

Catatan rilis untuk Amazon EMR di EKS 6.12.0

- Aplikasi yang didukung - AWS SDK untuk Java 1.12.490, Apache Spark 3.4.0-amzn-0, Apache Hudi 0.13.1-amzn-0, Apache Iceberg 1.3.0-amzn-0, Delta 2.4.0, Apache Spark RAPIDS 23.06.0-amzn-0, Jupyter Enterprise Gateway 2.6.0
- Komponen yang didukung - `aws-sagemaker-spark-sdk`, `emr-ddb`, `emr-goodies`, `emr-s3-select`, `emrfs`, `hadoop-client`, `hudi`, `hudi-spark`, `iceberg`, `spark-kubernetes`.
- Klasifikasi konfigurasi yang didukung

Untuk digunakan dengan [StartJobRundan](#) dan [CreateManagedEndpointAPI](#):

Klasifikasi	Deskripsi
<code>core-site</code>	Ubah nilai dalam file <code>core-site.xml</code> Hadoop.
<code>emrfs-site</code>	Ubah pengaturan EMRFS.
<code>spark-metrics</code>	Ubah nilai dalam file <code>metrics.properties</code> Spark.
<code>spark-defaults</code>	Ubah nilai dalam file <code>spark-defaults.conf</code> Spark.
<code>spark-env</code>	Ubah nilai di lingkungan Spark.
<code>spark-hive-site</code>	Ubah nilai dalam file <code>hive-site.xml</code> Spark.
<code>spark-log4j</code>	Ubah nilai dalam file <code>log4j2.properties</code> Spark.
<code>emr-job-submitter</code>	Konfigurasi untuk pod pengirim pekerjaan .

Untuk digunakan secara khusus dengan [CreateManagedEndpointAPI](#):

Klasifikasi	Deskripsi
jeg-config	Ubah nilai dalam file Jupyter Enterprise Gateway <code>jupyter_enterprise_gateway_config.py</code> .
jupyter-kernel-overrides	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi selengkapnya, lihat [Mengkonfigurasi Aplikasi](#).

Fitur penting

Fitur-fitur berikut disertakan dengan rilis 6.12 Amazon EMR di EKS.

- Java 17 - Dengan Amazon EMR di EKS 6.12 dan lebih tinggi, Anda dapat meluncurkan Spark dengan runtime Java 17. Untuk melakukan ini, berikan `emr-6.12.0-java17-latest` sebagai label rilis. Kami menyarankan Anda memvalidasi dan menjalankan pengujian kinerja sebelum memindahkan beban kerja produksi dari versi sebelumnya dari gambar Java ke gambar Java 17.

emr-6.12.0-terbaru

Catatan rilis: `emr-6.12.0-latest` saat ini menunjuk ke `emr-6.12.0-20240321`.

Wilayah: `emr-6.12.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.12.0:latest`

emr-6.12.0-20240321

Catatan rilis: `6.12.0-20240321` dirilis pada 11 Maret 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.12.0-20240321` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.12.0:20240321`

emr-6.12.0-20230701

Catatan rilis: `6.12.0-20230701` dirilis pada 1 Juli 2023. Ini adalah rilis awal Amazon EMR 6.12.0.

Wilayah: `emr-6.12.0-20230701` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.12.0:20230701`

Amazon EMR pada rilis EKS 6.11.0

Halaman ini menjelaskan fungsionalitas baru dan yang diperbarui untuk Amazon EMR yang khusus untuk EMR Amazon pada penerapan EKS. Untuk detail tentang Amazon EMR yang berjalan di Amazon EC2 dan tentang rilis Amazon EMR 6.11.0 secara umum, lihat Amazon EMR 6.11.0 di Panduan [Rilis Amazon EMR](#).

Amazon EMR pada rilis EKS 6.11

Rilis Amazon EMR 6.11.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis `EMR-6.11.0-XXXX` tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

- [emr-6.11.0-terbaru](#)
- [emr-6.11.0-20230905](#)
- [emr-6.11.0-20230509](#)

- `emr-6.11.0-spark-cepat-terbaru`
- `emr-6.11.0-spark-cepat-20230509`
- `emr-6.11.0-java11-terbaru`
- `emr-6.11.0-java11-20230509`
- `notebook- spark/emr -6.11.0-terbaru`
- `notebook- spark/emr -6.11.0-20230509`
- `notebook- python/emr -6.11.0-terbaru`

- notebook- python/emr -6.11.0-20230509

Catatan rilis

Catatan rilis untuk Amazon EMR di EKS 6.11.0

- Aplikasi yang didukung - AWS SDK untuk Java 1.12.446, Apache Spark 3.3.2-amzn-0, Apache Hudi 0.13.0-amzn-0, Apache Iceberg 1.2.0-amzn-0, Delta 2.2.0, Apache Spark RAPIDS 23.02.0-amzn-0, Jupyter Enterprise Gateway 2.6.0
- Komponen yang didukung - aws-sagemaker-spark-sdk,emr-ddb,emr-goodies,,emr-s3-select,emrfs,hadoop-client,hudi,hudi-spark,iceberg,spark-kubernetes.
- Klasifikasi konfigurasi yang didukung

Untuk digunakan dengan [StartJobRundan](#) [CreateManagedEndpoint](#)API:

Klasifikasi	Deskripsi
core-site	Ubah nilai dalam file core-site.xml Hadoop.
emrfs-site	Ubah pengaturan EMRFS.
spark-metrics	Ubah nilai dalam file metrics.properties Spark.
spark-defaults	Ubah nilai dalam file spark-defaults.conf Spark.
spark-env	Ubah nilai di lingkungan Spark.
spark-hive-site	Ubah nilai dalam file hive-site.xml Spark.
spark-log4j	Ubah nilai dalam file log4j.properties Spark.

Untuk digunakan secara khusus dengan [CreateManagedEndpoint](#)API:

Klasifikasi	Deskripsi
jeg-config	Ubah nilai dalam file Jupyter Enterprise Gateway <code>jupyter_enterprise_gateway_config.py</code> .
jupyter-kernel-overrides	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi selengkapnya, lihat [Mengkonfigurasi Aplikasi](#).

Fitur penting

Fitur-fitur berikut disertakan dengan rilis 6.11 Amazon EMR di EKS.

- [Amazon EMR pada gambar dasar EKS di Galeri Publik Amazon ECR](#) — Jika Anda menggunakan kemampuan [gambar khusus, gambar](#) dasar kami menyediakan toples, konfigurasi, dan pustaka penting untuk berinteraksi dengan Amazon EMR di EKS. Anda sekarang dapat menemukan gambar dasar di [Galeri Publik Amazon ECR](#).
- [Rotasi log kontainer percikan - Amazon EMR di EKS 6.11 mendukung rotasi log kontainer Spark](#). Anda dapat mengaktifkan kemampuan dengan `containerLogRotationConfiguration` dalam `MonitoringConfiguration` pengoperasian `StartJobRun API`. Anda dapat mengonfigurasi `rotationSize` dan `maxFilestoKeep` menentukan jumlah dan ukuran file log yang Anda inginkan Amazon EMR di EKS untuk disimpan di driver Spark dan pod pelaksana. Untuk informasi selengkapnya, lihat [Menggunakan rotasi log kontainer Spark](#).
- Dukungan gunung berapi di operator Spark dan `spark-submit` — [Amazon EMR di EKS 6.11 mendukung menjalankan pekerjaan Spark dengan Volcano sebagai penjadwal kustom Kubernetes di operator Spark dan `spark-submit`](#). Anda dapat menggunakan fitur seperti penjadwalan geng, manajemen antrian, preemption, dan penjadwalan berbagi adil untuk mencapai throughput penjadwalan yang tinggi dan kapasitas yang dioptimalkan. Untuk informasi selengkapnya, lihat [Menggunakan Volcano sebagai penjadwal khusus untuk Apache Spark di Amazon EMR di EKS](#).

emr-6.11.0-terbaru

Catatan rilis: `emr-6.11.0-latest` saat ini menunjuk ke `emr-20230905`.

Wilayah: `emr-6.11.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.11.0:latest`

emr-6.11.0-20230905

Catatan rilis: `6.11.0-20230905` dirilis pada 29 September 2023. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.11.0-20230509` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.11.0:20230509`

emr-6.11.0-20230509

Catatan rilis: `6.11.0-20230509` dirilis pada 9 Mei 2023. Ini adalah rilis awal Amazon EMR 6.11.0.

Wilayah: `emr-6.11.0-20230509` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.11.0:20230509`

Amazon EMR pada rilis EKS 6.10.0

Rilis Amazon EMR 6.10.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis `EMR-6.10.0-XXXX` tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

- [emr-6.10.0-terbaru](#)
- [emr-6.10.0-20230905](#)
- [emr-6.10.0-20230624](#)
- [emr-6.10.0-20230421](#)
- [emr-6.10.0-20230403](#)
- [emr-6.10.0-20230220](#)

- emr-6.10.0-spark-cepat-terbaru
- emr-6.10.0-spark-cepat-20230624
- emr-6.10.0-spark-cepat-20230220
- emr-6.10.0-java11-terbaru
- emr-6.10.0-java11-20230624
- emr-6.10.0-java11-20230220
- notebook- spark/emr -6.10.0-terbaru
- notebook- spark/emr -6.10.0-20230624
- notebook- spark/emr -6.10.0-20230220
- notebook- python/emr -6.10.0-terbaru
- notebook- python/emr -6.10.0-20230624
- notebook- python/emr -6.10.0-20230220

Catatan rilis untuk Amazon EMR 6.10.0

- Aplikasi yang didukung - AWS SDK untuk Java 1.12.397, Spark 3.3.1-amzn-0, Hudi 0.12.2-amzn-0, Iceberg 1.1.0-amzn-0, Delta 2.2.0.
- Komponen yang didukung - aws-sagemaker-spark-sdk, emr-ddb, emr-goodies, emr-s3-select, emrfs, hadoop-client, hudi, hudi-spark, iceberg, spark-kubernetes.
- Klasifikasi konfigurasi yang didukung:

Untuk digunakan dengan [StartJobRundan](#) dan [CreateManagedEndpointAPI](#):

Klasifikasi	Deskripsi
core-site	Ubah nilai dalam file Hadoop. core-site.xml
emrfs-site	Ubah pengaturan EMRFS.
spark-metrics	Ubah nilai dalam metrics.properties file Spark.
spark-defaults	Ubah nilai dalam spark-defaults.conf file Spark.

Klasifikasi	Deskripsi
spark-env	Ubah nilai di lingkungan Spark.
spark-hive-site	Ubah nilai dalam hive-site.xml file Spark.
spark-log4j	Ubah nilai dalam log4j.properties file Spark.

Untuk digunakan secara khusus dengan [CreateManagedEndpoint](#) API:

Klasifikasi	Deskripsi
jeg-config	Ubah nilai dalam file Jupyter Enterprise Gateway <code>jupyter_enterprise_gateway_config.py</code> .
jupyter-kernel-overrides	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi selengkapnya, lihat [Mengkonfigurasi Aplikasi](#).

Fitur penting

- Operator Spark - Dengan Amazon EMR di EKS 6.10.0 dan yang lebih tinggi, Anda dapat menggunakan operator Kubernetes untuk Apache Spark, atau operator Spark, untuk menyebarkan dan mengelola aplikasi Spark dengan runtime rilis Amazon EMR di kluster Amazon EKS Anda sendiri. Untuk informasi selengkapnya, lihat [Menjalankan pekerjaan Spark dengan operator Spark](#).
- Java 11 - Dengan Amazon EMR di EKS 6.10 dan lebih tinggi, Anda dapat meluncurkan Spark dengan runtime Java 11. Untuk melakukan ini, berikan `emr-6.10.0-java11-latest` sebagai label rilis. Sebaiknya Anda memvalidasi dan menjalankan pengujian kinerja sebelum memindahkan beban kerja produksi dari gambar Java 8 ke gambar Java 11.

- Untuk integrasi Amazon Redshift untuk Apache Spark, Amazon EMR di EKS 6.10.0 menghapus `dependensiminimal-jar.jar`, dan secara otomatis menambahkan jar terkait yang `spark-redshift` diperlukan ke jalur kelas pelaksana untuk Spark: `spark-redshift.jar` dan `spark-avro.jar` `RedshiftJDBC.jar`

Perubahan

- EMRFS S3-optimized committer sekarang diaktifkan secara default untuk `parquet`, `ORC`, dan format berbasis teks (termasuk `CSV` dan `JSON`).

emr-6.10.0-terbaru

Catatan rilis: `emr-6.10.0-latest` saat ini menunjuk ke `emr-6.10.0-20230905`.

Wilayah: `emr-6.10.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.10.0:latest`

emr-6.10.0-20230905

Catatan rilis: `6.10.0-20230905` dirilis pada 29 September 2023. Dibandingkan dengan rilis sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.10.0-20230905` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.10.0:20230905`

emr-6.10.0-20230624

Catatan rilis: `6.10.0-20230624` dirilis pada 7 Juli 2023. Dibandingkan dengan rilis sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.10.0-20230624` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.10.0:20230624`

emr-6.10.0-20230421

Catatan rilis: 6.10.0-20230421 dirilis pada 28 April 2023. Dibandingkan dengan rilis sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: emr-6.10.0-20230421 tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: emr-6.10.0:20230421

emr-6.10.0-20230403

Catatan rilis: 6.10.0-20230403 dirilis pada 12 April 2023. Dibandingkan dengan rilis sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: emr-6.10.0-20230403 tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: emr-6.10.0:20230403

emr-6.10.0-20230220

Catatan rilis: emr-6.10.0-20230220 dirilis pada 20 Februari 2023. Ini adalah rilis awal Amazon EMR 6.10.0.

Wilayah: emr-6.10.0-20230220 tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: emr-6.10.0:20230220

Amazon EMR pada rilis EKS 6.9.0

Rilis Amazon EMR 6.9.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis EMR-6.9.0-XXXX tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

- [emr-6.9.0-terbaru](#)
- [emr-6.9.0-20230905](#)
- [emr-6.9.0-20230624](#)
- [emr-6.9.0-20221108](#)

- `emr-6.9.0-spark-cepat-terbaru`
- `emr-6.9.0-spark-cepat-20230624`
- `emr-6.9.0-spark-cepat-20221108`
- `notebook- spark/emr -6.9.0-terbaru`
- `notebook- spark/emr -6.9.0-20230624`
- `notebook- spark/emr -6.9.0-20221108`
- `notebook- python/emr -6.9.0-terbaru`
- `notebook- python/emr -6.9.0-20230624`
- `notebook- python/emr -6.9.0-20221108`

Catatan rilis untuk Amazon EMR 6.9.0

- Aplikasi yang didukung - AWS SDK untuk Java 1.12.331, Spark 3.3.0-amzn-1, Hudi 0.12.1-amzn-0, Iceberg 0.14.1-amzn-0, Delta 2.1.0.
- Komponen yang didukung - `aws-sagemaker-spark-sdkemr-ddb`, `emr-goodies`, `emr-s3-select`, `emrfs`, `hadoop-client`, `hudi`, `hudi-spark`, `iceberg`, `spark-kubernetes`.
- Klasifikasi konfigurasi yang didukung:

Untuk digunakan dengan [StartJobRundan](#) dan [CreateManagedEndpointAPI](#):

Klasifikasi	Deskripsi
<code>core-site</code>	Ubah nilai dalam file <code>core-site.xml</code> Hadoop.
<code>emrfs-site</code>	Ubah pengaturan EMRFS.
<code>spark-metrics</code>	Ubah nilai dalam file <code>metrics.properties</code> Spark.
<code>spark-defaults</code>	Ubah nilai dalam file <code>spark-defaults.conf</code> Spark.
<code>spark-env</code>	Ubah nilai di lingkungan Spark.
<code>spark-hive-site</code>	Ubah nilai dalam file <code>hive-site.xml</code> Spark.
<code>spark-log4j</code>	Ubah nilai dalam file <code>log4j.properties</code> Spark.

Untuk digunakan secara khusus dengan [CreateManagedEndpoint](#) API:

Klasifikasi	Deskripsi
jeg-config	Ubah nilai dalam file Jupyter Enterprise Gateway <code>jupyter_enterprise_gateway_config.py</code> .
jupyter-kernel-overrides	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi selengkapnya, lihat [Mengkonfigurasi Aplikasi](#).

Fitur penting

- Akselerator Nvidia RAPIDS untuk Apache Spark - Amazon EMR di EKS untuk mempercepat Spark menggunakan tipe instans unit pemrosesan grafis (GPU) EC2. Untuk menggunakan gambar Spark dengan RAPIDS Accelerator, tentukan label rilis sebagai `emr-6.9.0-spark-rapids-latest`. Kunjungi [halaman dokumentasi](#) untuk mempelajari lebih lanjut.
- Spark-Redshift konektor - Integrasi Amazon Redshift untuk Apache Spark disertakan dalam rilis Amazon EMR 6.9.0 dan yang lebih baru. Sebelumnya alat open-source, integrasi asli adalah konektor Spark yang dapat Anda gunakan untuk membangun aplikasi Apache Spark yang membaca dan menulis ke data di Amazon Redshift dan Amazon Redshift Serverless. Untuk informasi selengkapnya, lihat [Menggunakan integrasi Amazon Redshift untuk Apache Spark di Amazon EMR di EKS](#).
- Delta Lake - [Delta Lake](#) adalah format penyimpanan sumber terbuka yang memungkinkan pembangunan danau data dengan konsistensi transaksional, definisi kumpulan data yang konsisten, perubahan evolusi skema, dan dukungan mutasi data. Kunjungi [Menggunakan Danau Delta](#) untuk mempelajari lebih lanjut.
- Ubah PySpark parameter - Titik akhir interaktif sekarang mendukung modifikasi parameter Spark yang terkait dengan PySpark sesi di Notebook EMR Studio Jupyter. Kunjungi [Memodifikasi parameter PySpark sesi](#) untuk mempelajari lebih lanjut.

Masalah terselesaikan

- Saat Anda menggunakan konektor DynamoDB dengan Spark di Amazon EMR versi 6.6.0, 6.7.0, dan 6.8.0, semua pembacaan dari tabel Anda mengembalikan hasil kosong, meskipun pemisahan input mereferensikan data yang tidak kosong. Amazon EMR rilis 6.9.0 memperbaiki masalah ini.
- [Amazon EMR di EKS 6.8.0 salah mengisi hash build dalam metadata file Parquet yang dihasilkan menggunakan Apache Spark](#). Masalah ini dapat menyebabkan alat yang mengurai string versi metadata dari file Parquet yang dihasilkan oleh Amazon EMR di EKS 6.8.0 gagal.

Masalah yang diketahui

- Jika Anda menggunakan integrasi Amazon Redshift untuk Apache Spark dan memiliki waktu, jadwal, stempel waktu, atau timestamptz dengan presisi mikrodetik dalam format Parquet, konektor membulatkan nilai waktu ke nilai milidetik terdekat. Sebagai solusinya, gunakan parameter format pembongkaran teks. `unload_s3_format`

emr-6.9.0-terbaru

Catatan rilis: `emr-6.9.0-latest` saat ini menunjuk ke `emr-6.9.0-20230905`.

Wilayah: `emr-6.9.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.9.0:latest`

emr-6.9.0-20230905

Catatan rilis: `emr-6.9.0-20230905`. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.9.0-20230905` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.9.0:20230905`

emr-6.9.0-20230624

Catatan rilis: `emr-6.9.0-20230624` dirilis pada 7 Juli 2023.

Wilayah: `emr-6.9.0-20230624` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.9.0:20230624`

emr-6.9.0-20221108

Catatan rilis: `emr-6.9.0-20221108` dirilis pada 08 Desember 2022. Ini adalah rilis awal Amazon EMR 6.9.0.

Wilayah: `emr-6.9.0-20221108` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.9.0:20221108`

Amazon EMR pada rilis EKS 6.8.0

Rilis Amazon EMR 6.8.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis EMR-6.8.0-XXXX tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

- [emr-6.8.0-terbaru](#)
- [emr-6.8.0-20230905](#)
- [emr-6.8.0-20230624](#)
- [emr-6.8.0-20221219](#)
- [emr-6.8.0-20220802](#)

Catatan rilis untuk Amazon EMR 6.8.0

- Aplikasi yang didukung - AWS SDK untuk Java 1.12.170, Spark 3.3.0-amzn-0, Hudi 0.11.1-amzn-0, Iceberg 0.14.0-amzn-0.
- Komponen yang didukung - `aws-sagemaker-spark-sdkemr-ddb`, `emr-goodies`, `emr-s3-select`, `emrfs`, `hadoop-client`, `hudi`, `hudi-spark`, `iceberg`, `spark-kubernetes`.
- Klasifikasi konfigurasi yang didukung:

Klasifikasi	Deskripsi
<code>core-site</code>	Ubah nilai dalam file <code>core-site.xml</code> Hadoop.

Klasifikasi	Deskripsi
<code>emrfs-site</code>	Ubah pengaturan EMRFS.
<code>spark-metrics</code>	Ubah nilai dalam file <code>metrics.properties</code> Spark.
<code>spark-defaults</code>	Ubah nilai dalam file <code>spark-defaults.conf</code> Spark.
<code>spark-env</code>	Ubah nilai di lingkungan Spark.
<code>spark-hive-site</code>	Ubah nilai dalam file <code>hive-site.xml</code> Spark.
<code>spark-log4j</code>	Ubah nilai dalam file <code>log4j.properties</code> Spark.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi selengkapnya, lihat [Mengkonfigurasi Aplikasi](#).

Fitur penting

- Spark3.3.0- Amazon EMR di EKS 6.8 menyertakan Spark 3.3.0, yang mendukung penggunaan label pemilih node terpisah untuk pod pelaksana driver Spark. Label baru ini memungkinkan Anda untuk menentukan tipe node untuk driver dan pod eksekutor secara terpisah di StartJobRun API, tanpa menggunakan templat pod.
 - Properti pemilih node driver: `spark.kubernetes.driver.node.selector`. [LabelKey]
 - Properti pemilih node pelaksana: `spark.kubernetes.executor.node.selector`. [LabelKey]
- Pesan kegagalan pekerjaan yang disempurnakan - Rilis ini memperkenalkan konfigurasi `spark.stage.extraDetailsOnFetchFailures.enabled` dan `spark.stage.extraDetailsOnFetchFailures.maxFailuresToInclude` melacak kegagalan tugas karena kode pengguna. Detail ini akan digunakan untuk menyempurnakan pesan kegagalan yang ditampilkan di log driver saat tahap dibatalkan karena kegagalan pengambilan acak.

Nama properti	Nilai default	Arti	Sejak versi
<code>spark.stage.extraDetailsOnFetchFailures.enabled</code>	false	<p>Jika disetel ke <code>true</code>, properti ini digunakan untuk menyempurnakan pesan kegagalan pekerjaan yang ditampilkan di log driver saat tahap dibatalkan karena Kegagalan Pengambilan Aduk. Secara default, 5 kegagalan tugas terakhir yang disebabkan oleh kode pengguna dilacak, dan pesan kesalahan kegagalan ditambahkan di Log Driver.</p> <p>Untuk meningkatkan jumlah kegagalan tugas dengan pengecualian pengguna untuk dilacak, lihat konfigurasi <code>spark.stage.extraDetailsOnFetchFailures.maxFa</code></p>	emr-6.8

Nama properti	Nilai default	Arti	Sejak versi
		iluresToInclude .	
spark.stage.extraDetailsOnFetchFailures.maxFailuresToInclude	5	Jumlah kegagalan tugas untuk melacak per tahap dan upaya. Properti ini digunakan untuk menyempurnakan pesan kegagalan pekerjaan dengan pengecualian pengguna yang ditampilkan di log driver saat tahap dibatalkan karena Kegagalan Pengambilan Aduk. Properti ini hanya berfungsi jika Config DetailsOnFetchFailures.enabled spark.stage.extra disetel ke true.	emr-6.8

Untuk informasi selengkapnya lihat dokumentasi [konfigurasi Apache Spark](#).

Masalah yang diketahui

- [Amazon EMR di EKS 6.8.0 salah mengisi hash build dalam metadata file Parquet yang dihasilkan menggunakan Apache Spark](#). Masalah ini dapat menyebabkan alat yang mengurai string versi metadata dari file Parquet yang dihasilkan oleh Amazon EMR di EKS 6.8.0 gagal. Pelanggan yang mengurai string versi dari metadata Parquet dan bergantung pada hash build harus beralih ke versi EMR Amazon yang berbeda dan menulis ulang file.

Masalah terselesaikan

- Kemampuan Kernel interupsi untuk kernel PySpark - Beban kerja interaktif yang sedang berlangsung yang dipicu oleh mengeksekusi sel di notebook dapat dihentikan dengan menggunakan kemampuan tersebut. `Interrupt Kernel` Perbaikan telah diperkenalkan sehingga fungsi ini berfungsi untuk kernel PySpark. Ini juga tersedia di open source di [Changes untuk menangani interupsi untuk PySpark Kubernetes](#) Kernel #1115.

emr-6.8.0-terbaru

Catatan rilis: `emr-6.8.0-latest` saat ini menunjuk ke `emr-6.8.0-20230624`.

Wilayah: `emr-6.8.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.8.0:latest`

emr-6.8.0-20230905

Catatan rilis: `emr-6.8.0-20230905` dirilis pada 29 September 2023. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.8.0-20230905` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.8.0:20230905`

emr-6.8.0-20230624

Catatan rilis: `emr-6.8.0-20230624` dirilis pada 7 Juli 2023. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.8.0-20230624` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.8.0:20230624`

emr-6.8.0-20221219

Catatan rilis: `emr-6.8.0-20221219` dirilis pada 19 Jan 2023. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.8.0-20221219` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.8.0:20221219`

emr-6.8.0-20220802

Catatan rilis: `emr-6.8.0-20220802` dirilis pada 27 Sep 2022. Ini adalah rilis awal Amazon EMR 6.8.0.

Wilayah: `emr-6.8.0-20220802` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.8.0:20220802`

Amazon EMR pada rilis EKS 6.7.0

Rilis Amazon EMR 6.7.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis `EMR-6.7.0-XXXX` tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

- [emr-6.7.0-terbaru](#)
- [emr-6.7.0-20240321](#)
- [emr-6.7.0-20230624](#)
- [emr-6.7.0-20221219](#)
- [emr-6.7.0-20220630](#)

Catatan rilis untuk Amazon EMR 6.7.0

- Aplikasi yang didukung - Spark 3.2.1-amzn-0, Jupyter Enterprise Gateway 2.6, Hudi 0.11-amzn-0, Iceberg 0.13.1.
- Komponen yang didukung - `aws-hm-client` (Konektor Glue), `aws-sagemaker-spark-sdk`, `emr-s3-select`, `emrfs`, `emr-ddb`, `hudi-spark`.

- Dengan upgrade ke JEG 2.6, manajemen kernel sekarang asinkron, yang berarti bahwa JEG tidak memblokir transaksi ketika peluncuran kernel sedang berlangsung. Ini sangat meningkatkan pengalaman pengguna dengan memberikan yang berikut:
 - kemampuan untuk menjalankan perintah di notebook yang sedang berjalan saat peluncuran kernel lainnya sedang berlangsung
 - kemampuan untuk meluncurkan beberapa kernel secara bersamaan tanpa memengaruhi kernel yang sudah berjalan
- Klasifikasi konfigurasi yang didukung:

Klasifikasi	Deskripsi
core-site	Ubah nilai dalam file Hadoopcore-site.xml .
emrfs-site	Ubah pengaturan EMRFS.
spark-metrics	Ubah nilai dalam metrics.properties file Spark.
spark-defaults	Ubah nilai dalam spark-defaults.conf file Spark.
spark-env	Ubah nilai di lingkungan Spark.
spark-hive-site	Ubah nilai dalam hive-site.xml file Spark.
spark-log4j	Ubah nilai dalam log4j.properties file Spark.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi lebih lanjut, lihat [Mengonfigurasi Aplikasi](#).

Masalah terselesaikan

- Amazon EMR di EKS 6.7 memperbaiki masalah di 6.6 saat menggunakan fungsionalitas templat pod Apache Spark dengan titik akhir interaktif. Masalah ini hadir di Amazon EMR pada rilis EKS 6.4, 6.5 dan 6.6. Anda sekarang dapat menggunakan templat pod untuk menentukan bagaimana driver Spark dan pod pelaksana Anda dimulai saat menggunakan titik akhir interaktif untuk menjalankan analitik interaktif.
- Di Amazon EMR sebelumnya pada rilis EKS, Jupyter Enterprise Gateway akan memblokir transaksi saat peluncuran kernel sedang berlangsung, dan ini menghambat pelaksanaan sesi notebook yang sedang berjalan. Anda sekarang dapat menjalankan perintah di notebook yang sedang berjalan saat peluncuran kernel lainnya sedang berlangsung. Anda juga dapat meluncurkan beberapa kernel secara bersamaan tanpa risiko kehilangan konektivitas ke kernel yang sudah berjalan.

emr-6.7.0-terbaru

Catatan rilis: `emr-6.7.0-latest` saat ini menunjuk ke `emr-6.7.0-20240321`.

Wilayah: `emr-6.7.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.7.0:latest`

emr-6.7.0-20240321

Catatan rilis: `emr-6.7.0-20240321` dirilis pada 11 Maret 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.7.0-20240321` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.7.0:20240321`

emr-6.7.0-20230624

Catatan rilis: `emr-6.7.0-20230624` dirilis pada 7 Juli 2023. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.7.0-20230624` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.7.0:20230624`

emr-6.7.0-20221219

Catatan rilis: `emr-6.7.0-20221219` dirilis pada 19 Januari 2023. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.7.0-20221219` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.7.0:20221219`

emr-6.7.0-20220630

Catatan rilis: `emr-6.7.0-20220630` dirilis pada 12 Juli 2022. Ini adalah rilis awal Amazon EMR 6.7.0.

Wilayah: `emr-6.7.0-20220630` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.7.0:20220630`

Amazon EMR pada rilis EKS 6.6.0

Rilis Amazon EMR 6.6.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis EMR-6.6.0-XXXX tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

- [emr-6.6.0-terbaru](#)
- [emr-6.6.0-20240321](#)
- [emr-6.6.0-20230624](#)
- [emr-6.6.0-20221219](#)
- [emr-6.6.0-20220411](#)

Catatan rilis untuk Amazon EMR 6.6.0

- Aplikasi yang didukung - Spark 3.2.0-amzn-0, Jupyter Enterprise Gateway (titik akhir, pratinjau publik), Hudi 0.10.1-amzn-0, Iceberg 0.13.1.

- Komponen yang didukung - `aws-hm-client` (Konektor Glue), `aws-sagemaker-spark-sdk`, `emr-s3-select`, `emrfs`, `emr-ddb`, `hudi-spark`.
- Klasifikasi konfigurasi yang didukung:

Klasifikasi	Deskripsi
<code>core-site</code>	Ubah nilai dalam file <code>core-site.xml</code> Hadoop.
<code>emrfs-site</code>	Ubah pengaturan EMRFS.
<code>spark-metrics</code>	Ubah nilai dalam file <code>metrics.properties</code> Spark.
<code>spark-defaults</code>	Ubah nilai dalam file <code>spark-defaults.conf</code> Spark.
<code>spark-env</code>	Ubah nilai di lingkungan Spark.
<code>spark-hive-site</code>	Ubah nilai dalam file <code>hive-site.xml</code> Spark.
<code>spark-log4j</code>	Ubah nilai dalam file <code>log4j.properties</code> Spark.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi lebih lanjut, lihat [Mengonfigurasi Aplikasi](#).

Masalah yang diketahui

- Fungsionalitas template Spark pod dengan titik akhir interaktif tidak berfungsi di Amazon EMR pada rilis EKS 6.4, 6.5, dan 6.6.

Masalah terselesaikan

- Log endpoint interaktif diunggah ke Cloudwatch dan S3.

emr-6.6.0-terbaru

Catatan rilis: `emr-6.6.0-latest` saat ini menunjuk ke `emr-6.6.0-20240321`.

Wilayah: `emr-6.6.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.6.0:latest`

`emr-6.6.0-20240321`

Catatan rilis: `emr-6.6.0-20240321` dirilis pada 11 Maret 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.6.0-20240321` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.6.0:20240321`

`emr-6.6.0-20230624`

Catatan rilis: `emr-6.6.0-20230624` dirilis pada 27 Jan 2023. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.6.0-20230624` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.6.0:20230624`

`emr-6.6.0-20221219`

Catatan rilis: `emr-6.6.0-20221219` dirilis pada 27 Jan 2023. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.6.0-20221219` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.6.0:20221219`

`emr-6.6.0-20220411`

Catatan rilis: `emr-6.6.0-20220411` dirilis pada 20 Mei 2022. Ini adalah rilis awal Amazon EMR 6.6.0.

Wilayah: `emr-6.6.0-20220411` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.6.0:20220411`

Amazon EMR pada rilis EKS 6.5.0

Rilis Amazon EMR 6.5.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis `EMR-6.5.0-XXXX` tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

- [emr-6.5.0-terbaru](#)
- [emr-6.5.0-20240321](#)
- [emr-6.5.0-20221219](#)
- [emr-6.5.0-20220802](#)
- [emr-6.5.0-20211119](#)

Catatan rilis untuk Amazon EMR 6.5.0

- Aplikasi yang didukung - Spark 3.1.2-amzn-1, Jupyter Enterprise Gateway (titik akhir, pratinjau publik).
- Komponen yang didukung - `aws-hm-client` (Konektor Glue), `aws-sagemaker-spark-sdk`, `emr-s3-select`, `emrfs`, `emr-ddb`, `hudi-spark`.
- Klasifikasi konfigurasi yang didukung:

Klasifikasi	Deskripsi
<code>core-site</code>	Ubah nilai dalam file <code>core-site.xml</code> Hadoop.
<code>emrfs-site</code>	Ubah pengaturan EMRFS.
<code>spark-metrics</code>	Ubah nilai dalam file <code>metrics.properties</code> Spark.
<code>spark-defaults</code>	Ubah nilai dalam file <code>spark-defaults.conf</code> Spark.
<code>spark-env</code>	Ubah nilai di lingkungan Spark.

Klasifikasi	Deskripsi
spark-hive-site	Ubah nilai dalam file hive-site.xml Spark.
spark-log4j	Ubah nilai dalam file log4j.properties Spark.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti spark-hive-site.xml. Untuk informasi lebih lanjut, lihat [Mengonfigurasi Aplikasi](#).

Masalah yang Diketahui

- Fungsionalitas template Spark pod dengan titik akhir interaktif tidak berfungsi di Amazon EMR pada rilis EKS 6.4 dan 6.5.

emr-6.5.0-terbaru

Catatan rilis: `emr-6.5.0-latest` saat ini menunjuk ke `emr-6.5.0-20240321`.

Wilayah: `emr-6.5.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.5.0:latest`

emr-6.5.0-20240321

Catatan rilis: `emr-6.5.0-20240321` dirilis pada 11 Maret 2024. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.5.0-20240321` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.5.0:20240321`

emr-6.5.0-20221219

Catatan rilis: emr-6.5.0-20221219 dirilis pada 19 Jan 2023. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: emr-6.5.0-20221219 tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: emr-6.5.0:20221219

emr-6.5.0-20220802

Catatan rilis: emr-6.5.0-20220802 dirilis pada 24 Agustus 2022. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui.

Wilayah: emr-6.5.0-20220802 tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: emr-6.5.0:20220802

emr-6.5.0-20211119

Catatan rilis: emr-6.5.0-20211119 dirilis pada 20 Jan 2022. Ini adalah rilis awal Amazon EMR 6.5.0.

Wilayah: emr-6.5.0-20211119 tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: emr-6.5.0:20211119

Amazon EMR pada rilis EKS 6.4.0

Rilis Amazon EMR 6.4.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis EMR-6.4.0-XXXX tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

- [emr-6.4.0-terbaru](#)
- [emr-6.4.0-20240321](#)
- [emr-6.4.0-20221219](#)
- [emr-6.4.0-20210830](#)

Catatan rilis untuk Amazon EMR 6.4.0

- Aplikasi yang didukung - Spark 3.1.2-amzn-0, Jupyter Enterprise Gateway (titik akhir, pratinjau publik).
- Komponen yang didukung - `aws-hm-client` (Konektor Glue), `aws-sagemaker-spark-sdk`, `emr-s3-select`, `emrfs`, `emr-ddb`, `hudi-spark`.
- Klasifikasi konfigurasi yang didukung:

Klasifikasi	Deskripsi
<code>core-site</code>	Ubah nilai dalam file <code>core-site.xml</code> Hadoop.
<code>emrfs-site</code>	Ubah pengaturan EMRFS.
<code>spark-metrics</code>	Ubah nilai dalam file <code>metrics.properties</code> Spark.
<code>spark-defaults</code>	Ubah nilai dalam file <code>spark-defaults.conf</code> Spark.
<code>spark-env</code>	Ubah nilai di lingkungan Spark.
<code>spark-hive-site</code>	Ubah nilai dalam file <code>hive-site.xml</code> Spark.
<code>spark-log4j</code>	Ubah nilai dalam file <code>log4j.properties</code> Spark.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi lebih lanjut, lihat [Mengonfigurasi Aplikasi](#).

Masalah yang diketahui

- Fungsionalitas template Spark pod dengan titik akhir interaktif tidak berfungsi di Amazon EMR pada rilis EKS 6.4.

emr-6.4.0-terbaru

Catatan rilis: `emr-6.4.0-latest` saat ini menunjuk ke `emr-6.4.0-20240321`.

Wilayah: `emr-6.4.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.4.0:latest`

`emr-6.4.0-20240321`

Catatan rilis: `emr-6.4.0-20240321` dirilis pada 11 Maret 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.4.0-20240321` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.4.0:20240321`

`emr-6.4.0-20221219`

Catatan rilis: `emr-6.4.0-20221219` dirilis pada 27 Jan 2023. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru saja ditambahkan.

Wilayah: `emr-6.4.0-20221219` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.4.0:20221219`

`emr-6.4.0-20210830`

Catatan rilis: `emr-6.4.0-20210830` dirilis pada 9 Desember 2021. Ini adalah rilis awal Amazon EMR 6.4.0.

Wilayah: `emr-6.4.0-20210830` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.4.0:20210830`

Amazon EMR pada rilis EKS 6.3.0

Rilis Amazon EMR 6.3.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis `EMR-6.3.0-XXXX` tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

- [emr-6.3.0-terbaru](#)
- [emr-6.3.0-20240321](#)
- [emr-6.3.0-20220802](#)
- [emr-6.3.0-20211008](#)
- [emr-6.3.0-20210802](#)
- [emr-6.3.0-20210429](#)

Catatan rilis untuk Amazon EMR 6.3.0

- Fitur baru - Dimulai dengan Amazon EMR 6.3.0 dalam seri rilis 6.x, Amazon EMR di EKS mendukung fitur templat pod Spark. Anda juga dapat mengaktifkan fitur rotasi Spark log peristiwa untuk Amazon EMR di EKS. Untuk informasi lebih lanjut, lihat [Menggunakan templat pod](#) dan [Menggunakan rotasi log peristiwa Spark](#).
- Aplikasi yang didukung - Spark 3.1.1-amzn-0, Jupyter Enterprise Gateway (titik akhir, pratinjau publik).
- Komponen yang didukung - `aws-hm-client` (Konektor Glue), `aws-sagemaker-spark-sdk`, `emr-s3-select`, `emrfs`, `emr-ddb`, `hudi-spark`.
- Klasifikasi konfigurasi yang didukung:

Klasifikasi	Deskripsi
<code>core-site</code>	Ubah nilai dalam file <code>core-site.xml</code> Hadoop.
<code>emrfs-site</code>	Ubah pengaturan EMRFS.
<code>spark-metrics</code>	Ubah nilai dalam file <code>metrics.properties</code> Spark.
<code>spark-defaults</code>	Ubah nilai dalam file <code>spark-defaults.conf</code> Spark.
<code>spark-env</code>	Ubah nilai di lingkungan Spark.
<code>spark-hive-site</code>	Ubah nilai dalam file <code>hive-site.xml</code> Spark.
<code>spark-log4j</code>	Ubah nilai dalam file <code>log4j.properties</code> Spark.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi lebih lanjut, lihat [Mengonfigurasi Aplikasi](#).

emr-6.3.0-terbaru

Catatan rilis: `emr-6.3.0-latest` saat ini menunjuk ke `emr-6.3.0-20240321`.

Wilayah: `emr-6.3.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.3.0:latest`

emr-6.3.0-20240321

Catatan rilis: `emr-6.3.0-20240321` dirilis pada 11 Maret 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.3.0-20240321` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.3.0:20240321`

emr-6.3.0-20220802

Catatan rilis: `emr-6.3.0-20220802` dirilis pada 27 Sep 2022. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui.

Wilayah: `emr-6.3.0-20220802` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.3.0:20220802`

emr-6.3.0-20211008

Catatan rilis: `emr-6.3.0-20211008` dirilis pada 9 Desember 2021. Dibandingkan dengan versi sebelumnya, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-6.3.0-20211008` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.3.0:20211008`

emr-6.3.0-20210802

Catatan rilis: `emr-6.3.0-20210802` dirilis pada 2 Agustus 2021. Dibandingkan dengan versi sebelumnya, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-6.3.0-20210802` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.3.0:20210802`

emr-6.3.0-20210429

Catatan rilis: `emr-6.3.0-20210429` dirilis pada tanggal 29 April 2021. Ini adalah rilis awal Amazon EMR 6.3.0.

Wilayah: `emr-6.3.0-20210429` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.3.0:20210429`

Amazon EMR pada rilis EKS 6.2.0

Rilis Amazon EMR 6.2.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis EMR-6.2.0-XXXX tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

- [emr-6.2.0-latest](#)
- [emr-6.2.0-20240321](#)
- [emr-6.2.0-20220802](#)
- [emr-6.2.0-20211008](#)
- [emr-6.2.0-20210802](#)
- [emr-6.2.0-20210615](#)
- [emr-6.2.0-20210129](#)
- [emr-6.2.0-20201218](#)

- [emr-6.2.0-20201201](#)

Catatan rilis untuk Amazon EMR 6.2.0

- Aplikasi yang didukung - Spark 3.0.1-amzn-0, Jupyter Enterprise Gateway (titik akhir, pratinjau publik).
- Komponen yang didukung - `aws-hm-client` (Konektor Glue), `aws-sagemaker-spark-sdk`, `emr-s3-select`, `emrfs`, `emr-ddb`, `hudi-spark`.
- Klasifikasi konfigurasi yang didukung:

Klasifikasi	Deskripsi
<code>core-site</code>	Ubah nilai dalam file <code>core-site.xml</code> Hadoop.
<code>emrfs-site</code>	Ubah pengaturan EMRFS.
<code>spark-metrics</code>	Ubah nilai dalam file <code>metrics.properties</code> Spark.
<code>spark-defaults</code>	Ubah nilai dalam file <code>spark-defaults.conf</code> Spark.
<code>spark-env</code>	Ubah nilai di lingkungan Spark.
<code>spark-hive-site</code>	Ubah nilai dalam file <code>hive-site.xml</code> Spark.
<code>spark-log4j</code>	Ubah nilai dalam file <code>log4j.properties</code> Spark.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi lebih lanjut, lihat [Mengonfigurasi Aplikasi](#).

emr-6.2.0-latest

Catatan rilis: `emr-6.2.0-latest` saat ini menunjuk ke `emr-6.2.0-20240321`.

Wilayah: `emr-6.2.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.2.0:20240321`

emr-6.2.0-20240321

Catatan rilis: `emr-6.2.0-20240321` dirilis pada 11 Maret 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.2.0-20240321` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.2.0:20240321`

emr-6.2.0-20220802

Catatan rilis: `emr-6.2.0-20220802` dirilis pada 27 Sep 2022. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui.

Wilayah: `emr-6.2.0-20220802` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-6.2.0:20220802`

emr-6.2.0-20211008

Catatan rilis: `emr-6.2.0-20211008` dirilis pada 9 Desember 2021. Dibandingkan dengan versi sebelumnya, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-6.2.0-20211008` tersedia di Wilayah Berikut: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Tanda gambar kontainer: `emr-6.2.0:20211008`

emr-6.2.0-20210802

Catatan rilis: `emr-6.2.0-20210802` dirilis pada 2 Agustus 2021. Dibandingkan dengan versi sebelumnya, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-6.2.0-20210802` tersedia di Wilayah Berikut: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Tanda gambar kontainer: `emr-6.2.0:20210802`

emr-6.2.0-20210615

Catatan rilis: `emr-6.2.0-20210615` dirilis pada 15 Juni 2021. Dibandingkan dengan versi sebelumnya, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-6.2.0-20210615` tersedia di Wilayah Berikut: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Tanda gambar kontainer: `emr-6.2.0:20210615`

emr-6.2.0-20210129

Catatan rilis: `emr-6.2.0-20210129` dirilis pada tanggal 29 Januari 2021. Dibandingkan dengan `emr-6.2.0-20201218`, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-6.2.0-20210129` tersedia di Wilayah berikut: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Tanda gambar kontainer: `emr-6.2.0-20210129`

emr-6.2.0-20201218

Catatan rilis: `emr-6.2.0-20201218` dirilis pada tanggal 18 Desember 2020. Dibandingkan dengan `emr-6.2.0-20201201`, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-6.2.0-20201218` tersedia di Wilayah berikut: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Tanda gambar kontainer: `emr-6.2.0-20201218`

emr-6.2.0-20201201

Catatan rilis: `emr-6.2.0-20201201` dirilis pada tanggal 1 Desember 2020. Ini adalah rilis awal Amazon EMR 6.2.0.

Wilayah: `emr-6.2.0-20201201` tersedia di Wilayah Berikut: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Tanda gambar kontainer: `emr-6.2.0-20201201`

Amazon EMR pada rilis EKS 5.36.0

Rilis Amazon EMR 5.36.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis EMR-5.36.0-XXXX tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

- [emr-5.36.0-terbaru](#)
- [emr-5.36.0-20240321](#)
- [emr-5.36.0-20221219](#)
- [emr-5.36.0-20220620](#)
- [emr-5.36.0-20220525](#)

Catatan rilis untuk Amazon EMR 5.36.0

- Memperbaiki masalah keamanan log4j2.
- Aplikasi yang didukung - Spark 2.4.8-amzn-2, Jupyter Enterprise Gateway (titik akhir, pratinjau publik; Kernel Scala tidak didukung), livy-0.7.1, fluentd-4.0.0.
- Komponen yang didukung - aws-hm-client, aws-sagemaker-spark-sdk, emr-ddb, emr-goodies, emr-kinesis, kerberos-server.
- Klasifikasi konfigurasi yang didukung:

Klasifikasi	Deskripsi
core-site	Ubah nilai dalam file core-site.xml Hadoop.
emrfs-site	Ubah pengaturan EMRFS.
spark-metrics	Ubah nilai dalam file metrics.properties Spark.
spark-defaults	Ubah nilai dalam file spark-defaults.conf Spark.
spark-env	Ubah nilai di lingkungan Spark.
spark-hive-site	Ubah nilai dalam file hive-site.xml Spark.
spark-log4j	Ubah nilai dalam file log4j.properties Spark.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi lebih lanjut, lihat [Mengonfigurasi Aplikasi](#).

emr-5.36.0-terbaru

Catatan rilis: `emr-5.36.0-latest` saat ini menunjuk ke `emr-5.36.0-20240321`.

Wilayah: `emr-5.36.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-5.36.0:latest`

emr-5.36.0-20240321

Catatan rilis: `emr-5.36.0-20240321` dirilis pada 11 Maret 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-5.36.0-20240321` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-5.36.0:20240321`

emr-5.36.0-20221219

Catatan rilis: `emr-5.36.0-20221219` dirilis pada 27 Jan 2023. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui.

Wilayah: `emr-5.36.0-20221219` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-5.36.0:20221219`

emr-5.36.0-20220620

Catatan rilis: `emr-5.36.0-20220620` dirilis pada 27 Juli 2022. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui.

Wilayah: `emr-5.36.0-20220620` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-5.36.0:20220620`

`emr-5.36.0-20220525`

Catatan rilis: `emr-5.36.0-20220525` dirilis pada 16 Juni 2022. Ini adalah rilis awal Amazon EMR 5.36.0.

Wilayah: `emr-5.36.0-20220525` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-5.36.0:20220525`

Amazon EMR pada rilis EKS 5.35.0

Rilis Amazon EMR 5.35.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis `EMR-5.35.0-XXXX` tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

- [emr-5.35.0-terbaru](#)
- [emr-5.35.0-20240321](#)
- [emr-5.35.0-20221219](#)
- [emr-5.35.0-20220802](#)
- [emr-5.35.0-20220307](#)

Catatan rilis untuk Amazon EMR 5.35.0

- Memperbaiki masalah keamanan `log4j2`.
- Aplikasi yang didukung - Spark 2.4.8-amzn-1, Hudi 0.9.0-amzn-2, Jupyter Enterprise Gateway (titik akhir, pratinjau publik; Kernel Scala tidak didukung).
- Komponen yang didukung - `aws-hm-client` (konektor Glue), `aws-sagemaker-spark-sdk`, `emr-s3-select`, `emrfs`, `emr-ddb`, `hudi-spark`.
- Klasifikasi konfigurasi yang didukung:

Klasifikasi	Deskripsi
core-site	Ubah nilai dalam file core-site.xml Hadoop.
emrfs-site	Ubah pengaturan EMRFS.
spark-metrics	Ubah nilai dalam file metrics.properties Spark.
spark-defaults	Ubah nilai dalam file spark-defaults.conf Spark.
spark-env	Ubah nilai di lingkungan Spark.
spark-hive-site	Ubah nilai dalam file hive-site.xml Spark.
spark-log4j	Ubah nilai dalam file log4j.properties Spark.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti spark-hive-site.xml. Untuk informasi lebih lanjut, lihat [Mengonfigurasi Aplikasi](#).

emr-5.35.0-terbaru

Catatan rilis: `emr-5.35.0-latest` saat ini menunjuk ke `emr-5.35.0-20240321`.

Wilayah: `emr-5.35.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-5.35.0:latest`

emr-5.35.0-20240321

Catatan rilis: `emr-5.35.0-20240321` dirilis pada 11 Maret 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-5.35.0-20240321` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-5.35.0:20240321`

`emr-5.35.0-20221219`

Catatan rilis: `emr-5.35.0-20221219` dirilis pada 27 Jan 2023. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui.

Wilayah: `emr-5.35.0-20221219` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-5.35.0:20221219`

`emr-5.35.0-20220802`

Catatan rilis: `emr-5.35.0-20220802` dirilis pada 27 Sep 2022. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui.

Wilayah: `emr-5.35.0-20220802` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-5.35.0:20220802`

`emr-5.35.0-20220307`

Catatan rilis: `emr-5.35.0-20220307` dirilis pada 30 Maret 2022. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui.

Wilayah: `emr-5.35.0-20220307` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-5.35.0:20220307`

Amazon EMR pada rilis EKS 5.34.0

Rilis Amazon EMR 5.34.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis `EMR-5.34.0-XXXX` tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

- [emr-5.34.0-terbaru](#)
- [emr-5.34.0-20240321](#)

- [emr-5.34.0-20220802](#)

Catatan rilis untuk Amazon EMR 5.34.0

- Aplikasi yang didukung - Spark 2.4.8-amzn-0, Jupyter Enterprise Gateway (titik akhir, pratinjau publik; Kernel Scala tidak didukung).
- Komponen yang didukung - `aws-hm-client` (Konektor Glue), `aws-sagemaker-spark-sdk`, `emr-s3-select`, `emrfs`, `emr-ddb`, `hudi-spark`.
- Klasifikasi konfigurasi yang didukung:

Klasifikasi	Deskripsi
<code>core-site</code>	Ubah nilai dalam file <code>core-site.xml</code> Hadoop.
<code>emrfs-site</code>	Ubah pengaturan EMRFS.
<code>spark-metrics</code>	Ubah nilai dalam file <code>metrics.properties</code> Spark.
<code>spark-defaults</code>	Ubah nilai dalam file <code>spark-defaults.conf</code> Spark.
<code>spark-env</code>	Ubah nilai di lingkungan Spark.
<code>spark-hive-site</code>	Ubah nilai dalam file <code>hive-site.xml</code> Spark.
<code>spark-log4j</code>	Ubah nilai dalam file <code>log4j.properties</code> Spark.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi lebih lanjut, lihat [Mengonfigurasi Aplikasi](#).

emr-5.34.0-terbaru

Catatan rilis: `emr-5.34.0-latest` saat ini menunjuk ke `emr-5.34.0-20220802`.

Wilayah: `emr-5.34.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-5.34.0:latest`

emr-5.34.0-20240321

Catatan rilis: `emr-5.34.0-20240321` dirilis pada 11 Maret 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-5.34.0-20240321` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-5.34.0:20240321`

emr-5.34.0-20220802

Catatan rilis: `emr-5.34.0-20220802` dirilis pada 24 Agustus 2022. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui.

Wilayah: `emr-5.34.0-20220802` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-5.34.0:20220802`

emr-5.34.0-20211208

Catatan rilis: `emr-5.34.0-20211208` dirilis pada 20 Jan 2022. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui.

Wilayah: `emr-5.34.0-20211208` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-5.34.0:20211208`

Amazon EMR pada rilis EKS 5.33.0

Rilis Amazon EMR 5.33.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis EMR-5.33.0-XXXX tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

- [emr-5.33.0-latest](#)

- [emr-5.33.0-20240321](#)
- [emr-5.33.0-20221219](#)
- [emr-5.33.0-20220802](#)
- [emr-5.33.0-20211008](#)
- [emr-5.33.0-20210802](#)
- [emr-5.33.0-20210615](#)
- [emr-5.33.0-20210323](#)

Catatan rilis untuk Amazon EMR 5.33.0

- Fitur baru - Dimulai dengan Amazon EMR 5.33.0 dalam seri rilis 5.x, Amazon EMR di EKS mendukung fitur templat pod Spark. Untuk informasi selengkapnya, lihat [Menggunakan templat pod](#).
- Aplikasi yang didukung - Spark 2.4.7-amzn-1, Jupyter Enterprise Gateway (titik akhir, pratinjau publik; Kernel Scala tidak didukung).
- Komponen yang didukung - `aws-hm-client` (Konektor Glue), `aws-sagemaker-spark-sdk`, `emr-s3-select`, `emrfs`, `emr-ddb`, `hudi-spark`.
- Klasifikasi konfigurasi yang didukung:

Klasifikasi	Deskripsi
<code>core-site</code>	Ubah nilai dalam file <code>core-site.xml</code> Hadoop.
<code>emrfs-site</code>	Ubah pengaturan EMRFS.
<code>spark-metrics</code>	Ubah nilai dalam file <code>metrics.properties</code> Spark.
<code>spark-defaults</code>	Ubah nilai dalam file <code>spark-defaults.conf</code> Spark.
<code>spark-env</code>	Ubah nilai di lingkungan Spark.
<code>spark-hive-site</code>	Ubah nilai dalam file <code>hive-site.xml</code> Spark.
<code>spark-log4j</code>	Ubah nilai dalam file <code>log4j.properties</code> Spark.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti `spark-hive-site.xml`. Untuk informasi lebih lanjut, lihat [Mengonfigurasi Aplikasi](#).

emr-5.33.0-latest

Catatan rilis: `emr-5.33.0-latest` saat ini menunjuk ke `emr-5.33.0-20240321`.

Wilayah: `emr-5.33.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-5.33.0:latest`

emr-5.33.0-20240321

Catatan rilis: `emr-5.33.0-20240321` dirilis pada 11 Maret 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-5.33.0-20240321` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-5.33.0:20240321`

emr-5.33.0-20221219

Catatan rilis: `emr-5.33.0-20221219` dirilis pada 19 Jan 2023. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-5.33.0-20221219` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-5.33.0:20221219`

emr-5.33.0-20220802

Catatan rilis: `emr-5.33.0-20220802` dirilis pada 24 Agustus 2022. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui.

Wilayah: `emr-5.33.0-20220802` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-5.33.0:20220802`

emr-5.33.0-20211008

Catatan rilis: `emr-5.33.0-20211008` dirilis pada 9 Desember 2021. Dibandingkan dengan versi sebelumnya, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-5.33.0-20211008` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-5.33.0:20211008`

emr-5.33.0-20210802

Catatan rilis: `emr-5.33.0-20210802` dirilis pada 2 Agustus 2021. Dibandingkan dengan versi sebelumnya, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-5.33.0-20210802` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-5.33.0:20210802`

emr-5.33.0-20210615

Catatan rilis: `emr-5.33.0-20210615` dirilis pada 15 Juni 2021. Dibandingkan dengan versi sebelumnya, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-5.33.0-20210615` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-5.33.0:20210615`

emr-5.33.0-20210323

Catatan rilis: `emr-5.33.0-20210323` dirilis pada tanggal 23 Maret 2021. Ini adalah rilis awal Amazon EMR 5.33.0.

Wilayah: `emr-5.33.0-20210323` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-5.33.0-20210323`

Amazon EMR pada rilis EKS 5.32.0

Rilis Amazon EMR 5.32.0 berikut tersedia untuk Amazon EMR di EKS. Pilih rilis EMR-5.32.0-XXXX tertentu untuk melihat detail selengkapnya seperti tag gambar kontainer terkait.

- [emr-5.32.0-latest](#)
- [emr-5.32.0-20240321](#)
- [emr-5.32.0-20220802](#)
- [emr-5.32.0-20211008](#)
- [emr-5.32.0-20210802](#)
- [emr-5.32.0-20210615](#)
- [emr-5.32.0-20210129](#)
- [emr-5.32.0-20201218](#)
- [emr-5.32.0-20201201](#)

Catatan rilis untuk Amazon EMR 5.32.0

- Aplikasi yang didukung - Spark 2.4.7-amzn-0, Jupyter Enterprise Gateway (titik akhir, pratinjau publik; Kernel Scala tidak didukung).
- Komponen yang didukung - `aws-hm-client` (Konektor Glue), `aws-sagemaker-spark-sdk`, `emr-s3-select`, `emrfs`, `emr-ddb`, `hudi-spark`.
- Klasifikasi konfigurasi yang didukung:

Klasifikasi	Deskripsi
<code>core-site</code>	Ubah nilai dalam file <code>core-site.xml</code> Hadoop.
<code>emrfs-site</code>	Ubah pengaturan EMRFS.
<code>spark-metrics</code>	Ubah nilai dalam file <code>metrics.properties</code> Spark.
<code>spark-defaults</code>	Ubah nilai dalam file <code>spark-defaults.conf</code> Spark.

Klasifikasi	Deskripsi
spark-env	Ubah nilai di lingkungan Spark.
spark-hive-site	Ubah nilai dalam file hive-site.xml Spark.
spark-log4j	Ubah nilai dalam file log4j.properties Spark.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering kali bersesuaian dengan file XML konfigurasi untuk aplikasi, seperti spark-hive-site.xml. Untuk informasi lebih lanjut, lihat [Mengonfigurasi Aplikasi](#).

emr-5.32.0-latest

Catatan rilis: `emr-5.32.0-latest` saat ini menunjuk ke `emr-5.32.0-20240321`.

Wilayah: `emr-5.32.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-5.32.0:latest`

emr-5.32.0-20240321

Catatan rilis: `emr-5.32.0-20240321` dirilis pada 11 Maret 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-5.32.0-20240321` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-5.32.0:20240321`

emr-5.32.0-20220802

Catatan rilis: `emr-5.32.0-20220802` dirilis pada 24 Agustus 2022. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui.

Wilayah: `emr-5.32.0-20220802` tersedia di semua Wilayah yang didukung oleh Amazon EMR di EKS. Untuk informasi selengkapnya, lihat [titik akhir layanan Amazon EMR di EKS](#).

Tanda gambar kontainer: `emr-5.32.0:20220802`

emr-5.32.0-20211008

Catatan rilis: `emr-5.32.0-20211008` dirilis pada 9 Desember 2021. Dibandingkan dengan versi sebelumnya, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-5.32.0-20211008` tersedia di Wilayah Berikut: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Tanda gambar kontainer: `emr-5.32.0:20211008`

emr-5.32.0-20210802

Catatan rilis: `emr-5.32.0-20210802` dirilis pada 2 Agustus 2021. Dibandingkan dengan versi sebelumnya, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-5.32.0-20210802` tersedia di Wilayah Berikut: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Tanda gambar kontainer: `emr-5.32.0:20210802`

emr-5.32.0-20210615

Catatan rilis: `emr-5.32.0-20210615` dirilis pada 15 Juni 2021. Dibandingkan dengan versi sebelumnya, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-5.32.0-20210615` tersedia di Wilayah Berikut: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Tanda gambar kontainer: `emr-5.32.0:20210615`

emr-5.32.0-20210129

Catatan rilis: `emr-5.32.0-20210129` dirilis pada tanggal 29 Januari 2021. Dibandingkan dengan `emr-5.32.0-20201218`, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-5.32.0-20210129` tersedia di Wilayah berikut: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Tanda gambar kontainer: `emr-5.32.0-20210129`

emr-5.32.0-20201218

Catatan rilis: 5.32.0-20201218 dirilis pada tanggal 18 Desember 2020. Dibandingkan dengan 5.32.0-20201201, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: emr-5.32.0-20201218 tersedia di Wilayah berikut: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Tanda gambar kontainer: emr-5.32.0-20201218

emr-5.32.0-20201201

Catatan rilis: 5.32.0-20201201 dirilis pada tanggal 1 Desember 2020. Ini adalah rilis awal Amazon EMR 5.32.0.

Wilayah: 5.32.0-20201201d tersedia di Wilayah berikut: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Tanda gambar kontainer: emr-5.32.0-20201201

Riwayat dokumen

Tabel berikut menjelaskan perubahan penting pada dokumentasi sejak rilis terakhir Amazon EMR di EKS. Untuk informasi lebih lanjut tentang pembaruan dokumentasi ini, Anda dapat berlangganan umpan RSS.

Ubah	Deskripsi	Date
Menambahkan informasi rilis emr-spark-8.0.0	Menambahkan informasi rilis emr-spark -8.0.0 (Apache Spark 4.0.2). Rilis GA pertama dari Spark 4.x di EMR di EKS.	Mei 2026
Menambahkan informasi rilis EMR 7.13.0	Menambahkan informasi rilis EMR 7.13.0 dengan Python 3.11 default untuk dan peningkatan aplikasi PySpark	April 2026
Menambahkan informasi rilis EMR 7.12.0	Menambahkan informasi rilis EMR 7.12.0 dengan Iceberg Materialized Views dan Hudi Full Table Access	November 2025
Menambahkan informasi rilis EMR 7.11.0	Menambahkan informasi rilis EMR 7.11.0 dengan integrasi Unified Studio SageMaker	November 2025
Perbarui konten	Kebijakan terkelola untuk Amazon EMR di EKS — Izin tambahan untuk AmazonEMRContainer sServiceRolePolicy	Februari 3, 2025
Rilis baru	Amazon EMR pada rilis EKS 7.6.0	Januari 10, 2025
Rilis baru	Amazon EMR pada rilis EKS 7.5.0	November 21, 2024
Rilis baru	Amazon EMR pada rilis EKS 7.4.0	November 13, 2024
Rilis baru	Amazon EMR pada rilis EKS 7.3.0	Oktober 16, 2024
Rilis baru	Amazon EMR pada rilis EKS 7.2.0	Juli 25, 2024

Ubah	Deskripsi	Date
Rilis baru	Amazon EMR pada rilis EKS 7.1.0	April 17, 2024
Rilis baru	Amazon EMR pada rilis EKS 7.0.0	22 Desember 2023
Rilis baru	Amazon EMR pada rilis EKS 6.15.0	17 November 2023
Rilis baru	Amazon EMR pada rilis EKS 6.14.0	17 Oktober 2023
Perbarui konten	Ubah nama “titik akhir terkelola” menjadi titik akhir interaktif ; Ketersediaan umum titik akhir interaktif	September 29, 2023
Rilis baru	Amazon EMR pada rilis EKS 6.13.0 , dan dokumen pratinjau publik untuk Menjalankan pekerjaan Flink dengan Amazon EMR di EKS	12 September 2023
Rilis baru	Amazon EMR pada rilis EKS 6.12.0	21 Juli 2023
Konten baru	Ditambahkan Menggunakan Volcano sebagai penjadwal khusus untuk Apache Spark di Amazon EMR di EKS	13 Juni 2023
Konten baru	Ditambahkan Menggunakan Volcano sebagai penjadwal khusus untuk Apache Spark di Amazon EMR di EKS	13 Juni 2023
Konten baru	Ditambahkan Menggunakan rotasi log kontainer Spark	12 Juni 2023
Perbarui konten	Memperbarui dokumentasi gambar khusus untuk menemukan informasi gambar dasar di Galeri Publik Amazon ECR.	8 Juni 2023
Rilis baru	Amazon EMR pada rilis EKS 6.11.0	8 Juni 2023

Ubah	Deskripsi	Date
Konten baru	Menambahkan Menjalankan pekerjaan Spark dengan operator Spark dan mengatur ulang bagian Job Runs di bawah Menjalankan pekerjaan Spark dengan Amazon EMR di EKS .	Juni 5, 2023
Konten baru	Ditambahkan dua bagian: Menggunakan penskalaan otomatis vertikal dengan pekerjaan Amazon EMR Spark dan Menggunakan notebook Jupyter yang dihosting sendiri	4 Mei 2023
Halaman riwayat dokumen	Membuat halaman riwayat dokumen untuk Amazon EMR di EKS.	13 Maret 2023
Halaman kebijakan terkelola	Membuat halaman kebijakan terkelola untuk Amazon EMR di EKS.	13 Maret 2023

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.