



Panduan Pengguna Tanpa Server Amazon EMR

Amazon EMR



Amazon EMR: Panduan Pengguna Tanpa Server Amazon EMR

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

Merek dagang dan tampilan dagang Amazon tidak boleh digunakan sehubungan dengan produk atau layanan apa pun yang bukan milik Amazon, dengan cara apa pun yang dapat menyebabkan kebingungan di antara pelanggan, atau dengan cara apa pun yang merendahkan atau mendiskreditkan Amazon. Semua merek dagang lain yang tidak dimiliki oleh Amazon merupakan hak milik masing-masing pemiliknya, yang mungkin atau mungkin tidak terafiliasi, terkait dengan, atau disponsori oleh Amazon.

Table of Contents

Apa itu Amazon EMR Tanpa Server?	1
Konsep	1
Versi rilis	1
Aplikasi	2
Tugas berjalan	2
Pekerja	3
Kapasitas pra-inisialisasi	3
Studio EMR	3
Prasyarat untuk memulai	5
Mendaftar untuk Akun AWS	5
Berikan izin	5
Memberikan akses programatis	7
Mengatur AWS CLI	9
Buka konsol	10
Memulai	11
Izin	11
Penyimpanan	11
Beban kerja interaktif	11
Buat peran runtime pekerjaan	12
Memulai dari konsol	18
Langkah 1: Buat aplikasi	18
Langkah 2: Kirimkan pekerjaan atau beban kerja interaktif	19
Langkah 3: Lihat UI aplikasi dan log	22
Langkah 4: Membersihkan	22
Memulai dari AWS CLI	23
Langkah 1: Buat aplikasi	23
Langkah 2: Kirimkan pekerjaan	24
Langkah 3: Tinjau keluaran	26
Langkah 4: Membersihkan	28
Berinteraksi dengan dan mengkonfigurasi aplikasi EMR Tanpa Server	30
Status aplikasi	30
Menggunakan konsol EMR Studio	31
Membuat aplikasi	32
Daftar aplikasi dari konsol EMR Studio	33

Mengelola aplikasi dari konsol EMR Studio	33
Menggunakan AWS CLI	33
Mengkonfigurasi aplikasi	34
Perilaku aplikasi	35
Pre-initialized kapasitas untuk bekerja dengan aplikasi di EMR Serverless	37
Konfigurasi aplikasi default	40
Menyesuaikan gambar	47
Prasyarat	36
Langkah 1: Buat gambar khusus dari gambar dasar EMR Serverless	48
Langkah 2: Validasi gambar secara lokal	49
Langkah 3: Unggah gambar ke repositori Amazon ECR Anda	50
Langkah 4: Buat atau perbarui aplikasi dengan gambar khusus	50
Langkah 5: Izinkan EMR Tanpa Server untuk mengakses repositori gambar kustom	52
Pertimbangan dan batasan	53
Mengkonfigurasi akses VPC untuk aplikasi EMR Tanpa Server untuk terhubung ke data	53
Buat aplikasi	54
Konfigurasi aplikasi	57
Praktik terbaik untuk perencanaan subnet	57
Pilihan arsitektur	59
Menggunakan arsitektur x86_64	59
Menggunakan arsitektur arm64 (Graviton)	60
Luncurkan aplikasi baru dengan Graviton	60
Konversi aplikasi yang ada ke Graviton	60
Pertimbangan-pertimbangan	61
Konkurensi dan antrian pekerjaan	62
Manfaat utama konkurensi dan antrian	62
Memulai dengan konkurensi dan antrian	62
Pertimbangan untuk konkurensi dan antrian	63
Mengunggah data	65
Prasyarat	65
Memulai dengan S3 Express One Zone	66
Menjalankan pekerjaan	68
Status tugas berjalan	68
Pembatalan Job run dengan masa tenggang	70
Masa Tenggang Untuk pekerjaan batch	70
Masa Tenggang Untuk Pekerjaan Streaming	72

Pertimbangan-pertimbangan	53
Menggunakan konsol EMR Studio	75
Mengirim tugas	75
Akses pekerjaan berjalan	78
Menggunakan AWS CLI	78
Eksekusi kebijakan IAM	80
Memulai	80
Contoh perintah CLI	80
Catatan Penting	82
Persimpangan Kebijakan	82
Menggunakan disk yang dioptimalkan dengan shuffle	85
Manfaat utama	85
Memulai	86
Menggunakan penyimpanan tanpa server	90
Manfaat utama	90
Memulai	91
Pertimbangan dan batasan	92
Didukung Wilayah AWS	93
Pekerjaan streaming untuk memproses data yang dialirkan secara terus menerus	94
Pertimbangan dan batasan	96
Memulai	96
Konektor streaming	97
Manajemen log	100
Menggunakan konfigurasi Spark saat Anda menjalankan pekerjaan EMR Tanpa Server	100
Parameter percikan	101
Properti percikan	104
Praktik terbaik konfigurasi sumber daya	110
Contoh percikan	111
Menggunakan konfigurasi Hive saat Anda menjalankan pekerjaan EMR Tanpa Server	112
Parameter sarang	112
Properti sarang	114
Contoh sarang	128
Ketahanan Job	129
Memantau pekerjaan dengan kebijakan coba lagi	132
Logging dengan kebijakan coba lagi	132
Konfigurasi metastore untuk EMR Tanpa Server	133

Menggunakan Katalog Data AWS Glue sebagai metastore	133
Menggunakan metastore Hive eksternal	138
Bekerja dengan hirarki multi-katalog AWS Glue di EMR Serverless	143
Pertimbangan saat menggunakan metastore eksternal	144
Cross-account Akses S3	145
Prasyarat	145
Menggunakan kebijakan bucket S3	145
Gunakan peran yang diasumsikan	147
Contoh peran yang diasumsikan	150
Memecahkan masalah kesalahan	154
Kesalahan: Job gagal karena akun telah mencapai batas layanan pada vCPU maksimum yang dapat digunakan secara bersamaan.	155
Kesalahan: Job gagal karena aplikasi telah melampaui pengaturan MaximumCapacity.	155
Kesalahan: Job gagal karena Worker tidak dapat dialokasikan karena aplikasi telah melebihi MaximumCapacity.	155
Kesalahan: Akses S3 ditolak. Silakan periksa izin akses S3 dari peran runtime pekerjaan pada sumber daya S3 yang diperlukan.	155
Kesalahan: ModuleNotFoundError: Tidak ada modul bernama<module>. Silakan merujuk ke panduan pengguna tentang cara menggunakan pustaka python dengan EMR Tanpa Server.	155
Kesalahan: Tidak dapat mengambil peran eksekusi <role name>karena tidak ada atau tidak diatur dengan hubungan kepercayaan yang diperlukan.	156
Alokasi Biaya Tingkat Pekerjaan	156
Perilaku default	156
Cara mengaktifkan atau menonaktifkan fitur	156
Pertimbangan dan batasan	157
Menjalankan Sesi Interaktif	158
Izin yang diperlukan	158
Bekerja dengan sesi interaktif	160
Pertimbangan dan batasan	164
Menjalankan beban kerja interaktif	166
Ikhtisar	166
Prasyarat	166
Izin	167
Konfigurasi	168
Pertimbangan-pertimbangan	168

Menjalankan beban kerja interaktif melalui titik akhir Apache Livy	170
Prasyarat	170
Izin yang diperlukan	170
Memulai	172
Pertimbangan-pertimbangan	178
Pencatatan log dan pemantauan	181
Menyimpan log	181
Penyimpanan terkelola	182
Amazon S3	183
Amazon CloudWatch	185
Memutar log	188
Mengkripsi log	190
Penyimpanan terkelola	190
Bucket Amazon S3	190
Amazon CloudWatch	190
Izin yang diperlukan	191
Mengkonfigurasi Log4j2	195
Log4j2 dan Spark	195
Memantau	199
Aplikasi dan pekerjaan	200
Metrik mesin percikan	209
Metrik penggunaan	214
Mengotomatisasi dengan EventBridge	215
Contoh acara EMR Tanpa Server EventBridge	216
Penandaan pada sumber daya	220
Apa itu tag?	220
Memberikan tag ke sumber daya	220
Batasan penandaan	221
Bekerja dengan tag	222
Tutorial	224
Menggunakan Java 17	224
JAVA_HOME	224
spark-defaults	225
Menggunakan Hudi	226
Menggunakan Iceberg	227
Menggunakan pustaka Python	228

Menggunakan fitur Python asli	228
Membangun lingkungan virtual Python	229
Mengkonfigurasi PySpark pekerjaan untuk menggunakan pustaka Python	230
Menggunakan versi Python yang berbeda	231
Menggunakan Delta Lake OSS	233
Amazon EMR versi 6.9.0 dan lebih tinggi	233
Amazon EMR versi 6.8.0 dan lebih rendah	234
Mengirimkan pekerjaan dari Airflow	235
Menggunakan fungsi yang ditentukan pengguna Hive	238
Menggunakan gambar kustom	239
Gunakan versi Python khusus	240
Gunakan versi Java kustom	240
Membangun citra ilmu data	241
Memproses data geospasial dengan Apache Sedona	241
Informasi lisensi untuk menggunakan gambar kustom	242
Menggunakan Spark di Amazon Redshift	242
Luncurkan aplikasi Spark	243
Otentikasi ke Amazon Redshift	244
Baca dan tulis ke Amazon Redshift	247
Pertimbangan-pertimbangan	249
Menghubungkan ke DynamoDB	250
Langkah 1: Unggah ke Amazon S3	250
Langkah 2: Buat tabel Hive	251
Langkah 3: Salin ke DynamoDB	252
Langkah 4: Query dari DynamoDB	254
Menyiapkan akses lintas akun	255
Pertimbangan-pertimbangan	257
Keamanan	260
Praktik terbaik keamanan	261
Terapkan prinsip hak istimewa paling rendah	261
Mengisolasi kode aplikasi yang tidak tepercaya	261
Izin kontrol akses berbasis peran (RBAC)	261
Perlindungan data	261
Enkripsi saat istirahat	263
Enkripsi saat bergerak	266
Identity and Access Management (IAM)	266

Audiens	267
Mengautentikasi dengan identitas	267
Mengelola akses menggunakan kebijakan	268
Bagaimana EMR Serverless bekerja dengan IAM	270
Menggunakan Peran Terkait Layanan	275
Peran runtime Job untuk Amazon EMR Tanpa Server	281
Kebijakan akses pengguna	284
Kebijakan untuk kendali akses berbasis tanda	288
Identity-based kebijakan	292
Pembaruan kebijakan	295
Pemecahan masalah	296
Propagasi Identitas Tepercaya	298
Ikhtisar	298
Fitur dan manfaat	299
Cara kerjanya	299
Memulai dengan Propagasi Identitas Tepercaya	300
Propagasi Identitas Tepercaya untuk beban kerja interaktif	304
Sesi latar belakang pengguna	304
Pertimbangan untuk integrasi EMR Tanpa Server Trusted-Identity-Propagation	309
Menggunakan Lake Formation dengan EMR Serverless	310
Ketersediaan fitur	310
Akses meja lengkap Lake Formation untuk EMR Tanpa Server	311
Lake Formation untuk FGAC	327
Enkripsi antar pekerja	357
Mengaktifkan enkripsi TLS timbal balik pada EMR Tanpa Server	358
Enkripsi Disk dengan KMS CMK	358
Menggunakan Konteks Enkripsi	359
Mengkonfigurasi Enkripsi Disk dengan Kunci yang Dikelola Pelanggan	359
Izin yang diperlukan untuk enkripsi disk	361
Pemantauan Penggunaan Kunci	364
Pelajari Lebih Lanjut	366
Secrets Manager untuk perlindungan data	367
Bagaimana rahasia bekerja	367
Buat rahasia	368
Tentukan referensi rahasia	368
Berikan akses ke rahasia	370

Putar rahasianya	372
Hibah Akses S3 untuk kontrol akses data	373
Ikhtisar	373
Meluncurkan aplikasi	373
Pertimbangan-pertimbangan	375
CloudTrail untuk penebangan	375
EMR Informasi tanpa server di CloudTrail	375
Memahami entri file log EMR Tanpa Server	376
Validasi kepatuhan	378
Ketahanan	379
Keamanan infrastruktur	379
Konfigurasi dan analisis kerentanan	380
Titik akhir dan kuota	381
Titik akhir layanan	381
Kuota layanan	386
Batas API	387
Pertimbangan lainnya	53
Versi rilis	391
Waktu proses AWS untuk Apache Spark (emr-spark-8.0.0)	392
AWS runtime untuk Apache Spark (emr-spark-8.0-preview)	393
EMR Tanpa Server 7.13.0	395
EMR Tanpa Server 7.12.0	395
EMR Tanpa Server 7.11.0	396
EMR Tanpa Server 7.10.0	397
EMR Tanpa Server 7.9.0	397
EMR Tanpa Server 7.8.0	398
EMR Tanpa Server 7.7.0	398
EMR Tanpa Server 7.6.0	398
EMR Tanpa Server 7.5.0	399
EMR Tanpa Server 7.4.0	399
EMR Tanpa Server 7.3.0	399
EMR Tanpa Server 7.2.0	400
EMR Tanpa Server 7.1.0	401
EMR Tanpa Server 7.0.0	401
EMR Tanpa Server 6.15.0	401
EMR Tanpa Server 6.14.0	402

EMR Tanpa Server 6.13.0	402
EMR Tanpa Server 6.12.0	402
EMR Tanpa Server 6.11.0	403
EMR Tanpa Server 6.10.0	403
EMR Tanpa Server 6.9.0	404
EMR Tanpa Server 6.8.0	405
EMR Tanpa Server 6.7.0	405
Engine-specific perubahan	405
EMR Tanpa Server 6.6.0	406
Riwayat dokumen	408
.....	cdx

Apa itu Amazon EMR Tanpa Server?

Amazon EMR Tanpa Server adalah opsi penerapan untuk Amazon EMR yang menyediakan lingkungan runtime tanpa server. Ini menyederhanakan pengoperasian aplikasi analitik yang menggunakan kerangka kerja open-source terbaru, seperti Apache Spark dan Apache Hive. Dengan EMR Tanpa Server, Anda tidak perlu mengonfigurasi, mengoptimalkan, mengamankan, atau mengoperasikan cluster untuk menjalankan aplikasi dengan kerangka kerja ini.

EMR Tanpa Server membantu Anda menghindari sumber daya yang berlebihan atau kurang penyediaan untuk pekerjaan pemrosesan data Anda. EMR Tanpa Server secara otomatis menentukan sumber daya yang dibutuhkan aplikasi, mendapatkan sumber daya ini untuk memproses pekerjaan Anda, dan melepaskan sumber daya saat pekerjaan selesai. Untuk kasus penggunaan di mana aplikasi memerlukan respons dalam hitungan detik, seperti analisis data interaktif, Anda dapat melakukan pra-inisialisasi sumber daya yang dibutuhkan aplikasi saat membuat aplikasi.

Dengan EMR Tanpa Server, Anda terus mendapatkan manfaat Amazon EMR, seperti kompatibilitas open source, konkurensi, dan kinerja runtime yang dioptimalkan untuk kerangka kerja populer.

EMR Serverless cocok untuk pelanggan yang menginginkan kemudahan dalam mengoperasikan aplikasi menggunakan kerangka kerja open source. Ini menawarkan startup pekerjaan cepat, manajemen kapasitas otomatis, dan kontrol biaya langsung.

Konsep

Pada bagian ini, kami membahas istilah dan konsep EMR Tanpa Server yang muncul di seluruh Panduan Pengguna EMR Tanpa Server kami.

Versi rilis

Rilis EMR Amazon adalah seperangkat aplikasi open-source dari ekosistem big data. Setiap rilis mencakup berbagai aplikasi big data, komponen, dan fitur yang Anda pilih untuk EMR Serverless untuk disebar dan dikonfigurasi sehingga mereka dapat menjalankan aplikasi Anda. Saat Anda membuat aplikasi, tentukan versi rilisnya. Pilih versi rilis Amazon EMR dan versi kerangka kerja sumber terbuka yang ingin Anda gunakan dalam aplikasi Anda. Untuk mempelajari lebih lanjut tentang versi pra-rilis, lihat. [Amazon EMR Versi rilis tanpa server](#)

Aplikasi

Dengan EMR Tanpa Server, Anda dapat membuat satu atau lebih aplikasi EMR Tanpa Server yang menggunakan kerangka kerja analitik open source. Untuk membuat aplikasi, tentukan atribut berikut:

- Versi rilis Amazon EMR untuk versi kerangka open source yang ingin Anda gunakan. Untuk menentukan versi rilis Anda, lihat [Amazon EMR Versi rilis tanpa server](#).
- Runtime spesifik yang Anda ingin aplikasi Anda gunakan, seperti Apache Spark atau Apache Hive.

Setelah Anda membuat aplikasi, kirimkan pekerjaan pemrosesan data atau permintaan interaktif ke aplikasi Anda.

Setiap aplikasi EMR Tanpa Server berjalan pada Amazon Virtual Private Cloud (VPC) yang aman terpisah dari aplikasi lain. Selain itu, gunakan kebijakan AWS Identity and Access Management (IAM) untuk menentukan pengguna dan peran mana yang dapat mengakses aplikasi. Anda juga dapat menentukan batasan untuk mengontrol dan melacak biaya penggunaan yang dikeluarkan oleh aplikasi.

Pertimbangkan untuk membuat beberapa aplikasi ketika Anda harus melakukan hal berikut:

- Gunakan kerangka kerja open source yang berbeda
- Gunakan versi kerangka kerja open source yang berbeda untuk kasus penggunaan yang berbeda
- Lakukan A/B pengujian saat memutakhirkan dari satu versi ke versi lainnya
- Pertahankan lingkungan logis yang terpisah untuk skenario pengujian dan produksi
- Menyediakan lingkungan logis terpisah untuk tim yang berbeda dengan kontrol biaya independen dan pelacakan penggunaan
- Pisahkan line-of-business aplikasi yang berbeda

EMR Tanpa Server adalah layanan Regional yang menyederhanakan bagaimana beban kerja berjalan di beberapa Availability Zone di suatu Wilayah. Untuk mempelajari lebih lanjut tentang cara menggunakan aplikasi dengan EMR Tanpa Server, lihat [Berinteraksi dengan dan mengkonfigurasi aplikasi EMR Tanpa Server](#)

Tugas berjalan

Job run adalah permintaan yang dikirimkan ke aplikasi EMR Tanpa Server yang dijalankan dan dilacak oleh aplikasi secara asinkron hingga selesai. Contoh pekerjaan termasuk kueri HiveQL

yang Anda kirimkan ke aplikasi Apache Hive, atau PySpark skrip pemrosesan data yang Anda kirimkan ke aplikasi Apache Spark. Saat mengirimkan pekerjaan, Anda harus menentukan peran runtime, yang ditulis di IAM, yang digunakan pekerjaan untuk mengakses AWS sumber daya, seperti objek Amazon S3. Anda dapat mengirimkan beberapa permintaan job run ke aplikasi, dan setiap job run dapat menggunakan peran runtime yang berbeda untuk mengakses AWS sumber daya. Aplikasi EMR Tanpa Server mulai mengeksekusi pekerjaan segera setelah menerimanya dan menjalankan beberapa permintaan pekerjaan secara bersamaan. Untuk mempelajari lebih lanjut tentang bagaimana EMR Serverless menjalankan pekerjaan, lihat. [Menjalankan pekerjaan](#)

Pekerja

Aplikasi EMR Tanpa Server secara internal menggunakan pekerja untuk mengeksekusi beban kerja Anda. Ukuran default pekerja ini didasarkan pada jenis aplikasi Anda dan versi rilis Amazon EMR. Saat Anda menjadwalkan pekerjaan, ganti ukuran ini.

Saat Anda mengirimkan pekerjaan, EMR Serverless menghitung sumber daya yang dibutuhkan aplikasi untuk pekerjaan dan menjadwalkan pekerja. EMR Tanpa Server memecah beban kerja Anda menjadi tugas, mengunduh gambar, ketentuan, dan menyiapkan pekerja, dan menonaktifkannya saat pekerjaan selesai. EMR Tanpa Server secara otomatis meningkatkan atau menurunkan pekerja berdasarkan beban kerja dan paralelisme yang diperlukan pada setiap tahap pekerjaan. Penskalaan otomatis ini menghilangkan kebutuhan Anda untuk memperkirakan jumlah pekerja yang dibutuhkan aplikasi untuk menjalankan beban kerja Anda.

Kapasitas pra-inisialisasi

EMR Tanpa Server menyediakan fitur kapasitas pra-inisialisasi yang membuat pekerja diinisialisasi dan siap merespons dalam hitungan detik. Kapasitas ini secara efektif menciptakan kumpulan pekerja yang hangat untuk suatu aplikasi. Untuk mengkonfigurasi fitur ini untuk setiap aplikasi, atur `initial-capacity` parameter aplikasi. Saat Anda mengonfigurasi kapasitas pra-inisialisasi, pekerjaan dapat segera dimulai sehingga Anda dapat menerapkan aplikasi berulang dan pekerjaan yang sensitif terhadap waktu. Untuk mempelajari lebih lanjut tentang pekerja pra-inisialisasi, lihat. [Mengkonfigurasi aplikasi saat bekerja dengan EMR Serverless](#)

Studio EMR

EMR Studio adalah konsol pengguna untuk mengelola aplikasi EMR Tanpa Server Anda. Jika EMR Studio tidak ada di akun Anda saat Anda membuat aplikasi EMR Tanpa Server pertama Anda, kami secara otomatis membuatnya untuk Anda. Akses EMR Studio baik dari konsol EMR Amazon, atau

aktifkan akses federasi dari penyedia identitas (iDP) Anda melalui IAM atau IAM Identity Center. Saat Anda melakukan ini, pengguna dapat mengakses Studio dan mengelola aplikasi EMR Tanpa Server tanpa akses langsung ke konsol EMR Amazon. Untuk mempelajari lebih lanjut tentang cara kerja aplikasi EMR Tanpa Server dengan EMR Studio, lihat dan. [Membuat aplikasi EMR Tanpa Server dari konsol EMR Studio](#) [Menjalankan pekerjaan dari konsol EMR Studio](#)

Prasyarat untuk memulai dengan EMR Serverless

Bagian ini menjelaskan prasyarat administratif untuk menjalankan EMR Tanpa Server. Ini termasuk konfigurasi akun dan manajemen izin.

Topik

- [Mendaftar untuk Akun AWS](#)
- [Berikan izin](#)
- [Instal dan konfigurasi AWS CLI](#)
- [Buka konsol](#)

Mendaftar untuk Akun AWS

Untuk memulai AWS, Anda membutuhkan Akun AWS. Untuk informasi tentang membuat Akun AWS, lihat [Memulai dengan Akun AWS](#) di Panduan AWS Account Management Referensi.

Berikan izin

Di lingkungan produksi, kami menyarankan Anda menggunakan kebijakan yang lebih halus. Untuk contoh kebijakan tersebut, lihat [Contoh kebijakan akses pengguna untuk EMR Tanpa Server](#). Untuk mempelajari lebih lanjut tentang manajemen akses, lihat [Manajemen akses untuk AWS sumber daya](#) di Panduan Pengguna IAM.

Untuk pengguna yang perlu memulai EMR Tanpa Server di lingkungan kotak pasir, gunakan kebijakan yang mirip dengan berikut ini:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EMRStudioCreate",
      "Effect": "Allow",
      "Action": [
        "elasticmapreduce:CreateStudioPresignedUrl",
        "elasticmapreduce:DescribeStudio",
```

```

    "elasticmapreduce:CreateStudio",
    "elasticmapreduce:ListStudios"
  ],
  "Resource": [
    "*"
  ]
},
{
  "Sid": "EMRServerlessFullAccess",
  "Effect": "Allow",
  "Action": [
    "emr-serverless:*"
  ],
  "Resource": [
    "*"
  ]
},
{
  "Sid": "AllowEC2ENICreationWithEMRTags",
  "Effect": "Allow",
  "Action": [
    "ec2:CreateNetworkInterface"
  ],
  "Resource": [
    "arn:aws:ec2:*:*:network-interface/*"
  ],
  "Condition": {
    "StringEquals": {
      "aws:CalledViaLast": "ops.emr-serverless.amazonaws.com"
    }
  }
},
{
  "Sid": "AllowEMRServerlessServiceLinkedRoleCreation",
  "Effect": "Allow",
  "Action": [
    "iam:CreateServiceLinkedRole"
  ],
  "Resource": [
    "arn:aws:iam:*:*:role/aws-service-role/*"
  ]
}
]

```

}

Untuk memberikan akses dan menambahkan izin bagi pengguna, grup, atau peran Anda:

- Pengguna dan grup di AWS IAM Identity Center:

Buat rangkaian izin. Ikuti instruksi di [Buat rangkaian izin](#) dalam Panduan Pengguna AWS IAM Identity Center .

- Pengguna yang dikelola di IAM melalui penyedia identitas:

Buat peran untuk federasi identitas. Ikuti instruksi dalam [Buat peran untuk penyedia identitas pihak ketiga \(federasi\)](#) dalam Panduan Pengguna IAM.

- Pengguna IAM:

- Buat peran yang dapat diambil pengguna Anda. Ikuti instruksi dalam [Buat peran untuk pengguna IAM](#) dalam Panduan Pengguna IAM.
- (Tidak disarankan) Lampirkan kebijakan langsung ke pengguna atau tambahkan pengguna ke grup pengguna. Ikuti petunjuk dalam [Menambahkan izin ke pengguna \(konsol\)](#) dalam Panduan Pengguna IAM.

Memberikan akses programatis

Pengguna membutuhkan akses terprogram jika mereka ingin berinteraksi dengan AWS luar. Konsol Manajemen AWS Cara untuk memberikan akses terprogram tergantung pada jenis pengguna yang mengakses AWS.

Untuk memberi pengguna akses programatis, pilih salah satu opsi berikut.

Pengguna mana yang membutuhkan akses programatis?	Untuk	Oleh
IAM	(Disarankan) Gunakan kredensial konsol sebagai kredensial sementara untuk menandatangani permintaan	Mengikuti petunjuk untuk antarmuka yang ingin Anda gunakan. <ul style="list-style-type: none"> • Untuk itu AWS CLI, lihat Login untuk pengembangan

Pegguna mana yang membutuhkan akses programatis?	Untuk	Oleh
	terprogram ke, SDK AWS CLI, AWS atau API. AWS	<p>AWS lokal di Panduan AWS Command Line Interface Pengguna.</p> <ul style="list-style-type: none"> • Untuk AWS SDK, lihat Login untuk pengembangan AWS lokal di AWS SDK dan Panduan Referensi Alat.
Identitas tenaga kerja (Pengguna yang dikelola di Pusat Identitas IAM)	Gunakan kredensial sementara untuk menandatangani permintaan terprogram ke AWS CLI, AWS SDK, atau API. AWS	<p>Mengikuti petunjuk untuk antarmuka yang ingin Anda gunakan.</p> <ul style="list-style-type: none"> • Untuk AWS CLI, lihat Mengkonfigurasi yang akan AWS CLI digunakan AWS IAM Identity Center dalam Panduan AWS Command Line Interface Pengguna. • Untuk AWS SDK, alat, dan AWS API, lihat otentikasi Pusat Identitas IAM di Panduan Referensi AWS SDK dan Alat.
IAM	Gunakan kredensial sementara untuk menandatangani permintaan terprogram ke AWS CLI, AWS SDK, atau API. AWS	<p>Mengikuti petunjuk dalam Menggunakan kredensial sementara dengan AWS sumber daya di Panduan Pengguna IAM.</p>

Pengguna mana yang membutuhkan akses programatis?	Untuk	Oleh
IAM	(Tidak direkomendasikan) Gunakan kredensial jangka panjang untuk menandatangani permintaan terprogram ke AWS CLI, AWS SDK, atau API. AWS	<p>Mengikuti petunjuk untuk antarmuka yang ingin Anda gunakan.</p> <ul style="list-style-type: none"> • Untuk mengetahui AWS CLI, lihat Mengautentikasi menggunakan kredensial pengguna IAM di Panduan Pengguna. AWS Command Line Interface • Untuk AWS SDK dan alat bantu, lihat Mengautentikasi menggunakan kredensial jangka panjang di Panduan Referensi AWS SDK dan Alat. • Untuk AWS API, lihat Mengelola kunci akses untuk pengguna IAM di Panduan Pengguna IAM.

Instal dan konfigurasi AWS CLI

Jika Anda ingin menggunakan EMR Serverless API, instal versi terbaru dari (). AWS Command Line Interface AWS CLI Anda tidak AWS CLI perlu menggunakan EMR Tanpa Server dari konsol EMR Studio, dan memulai tanpa CLI dengan mengikuti langkah-langkahnya. [Memulai dengan EMR Serverless dari konsol](#)

Untuk mengatur AWS CLI

1. Untuk menginstal versi terbaru dari AWS CLI macOS, Linux, atau Windows, lihat [Menginstal atau memperbarui versi terbaru](#). AWS CLI

2. [Untuk mengonfigurasi AWS CLI dan mengamankan pengaturan akses Anda Layanan AWS, termasuk EMR Tanpa Server, lihat Konfigurasi cepat dengan. aws configure](#)
3. Untuk memverifikasi pengaturan, masukkan DataBrew perintah berikut pada prompt perintah.

```
aws emr-serverless help
```

AWS CLI perintah menggunakan default Wilayah AWS dari konfigurasi Anda, kecuali Anda mengaturnya dengan parameter atau profil. Untuk mengatur parameter Anda Wilayah AWS , tambahkan `--region` parameter ke setiap perintah.

Untuk mengatur profil Anda Wilayah AWS , pertama-tama tambahkan profil bernama dalam `~/.aws/config` file atau `%UserProfile%/.aws/config` file (untuk Microsoft Windows). Ikuti langkah-langkah di [Profil bernama untuk AWS CLI](#). Selanjutnya, atur pengaturan Anda Wilayah AWS dan lainnya dengan perintah yang mirip dengan yang ada di contoh berikut.

```
[profile emr-serverless]
aws_access_key_id = ACCESS-KEY-ID-OF-IAM-USER
aws_secret_access_key = SECRET-ACCESS-KEY-ID-OF-IAM-USER
region = us-east-1
output = text
```

Buka konsol

Sebagian besar topik berorientasi konsol di bagian ini dimulai dari konsol [EMR](#) Amazon. Jika Anda belum masuk Akun AWS, masuk, lalu buka [konsol Amazon EMR](#) dan lanjutkan ke bagian berikutnya untuk melanjutkan memulai dengan Amazon EMR.

Memulai dengan Amazon EMR Tanpa Server

Tutorial ini membantu Anda memulai dengan EMR Serverless saat Anda menerapkan contoh beban kerja Spark atau Hive. Anda akan membuat, menjalankan, dan men-debug aplikasi Anda sendiri. Kami menampilkan opsi default di sebagian besar bagian tutorial ini.

Sebelum Anda meluncurkan aplikasi EMR Tanpa Server, selesaikan tugas-tugas berikut.

Topik

- [Berikan izin untuk menggunakan EMR Tanpa Server](#)
- [Siapkan penyimpanan untuk EMR Tanpa Server](#)
- [Buat EMR Studio untuk menjalankan beban kerja interaktif](#)
- [Buat peran runtime pekerjaan](#)
- [Memulai dengan EMR Serverless dari konsol](#)
- [Memulai dari AWS CLI](#)

Berikan izin untuk menggunakan EMR Tanpa Server

Untuk menggunakan EMR Tanpa Server, Anda memerlukan peran pengguna atau IAM dengan kebijakan terlampir yang memberikan izin untuk EMR Tanpa Server. Untuk membuat pengguna dan melampirkan kebijakan yang sesuai kepada pengguna tersebut, ikuti instruksi di [Berikan izin](#).

Siapkan penyimpanan untuk EMR Tanpa Server

Dalam tutorial ini, Anda akan menggunakan bucket S3 untuk menyimpan file output dan log dari sampel beban kerja Spark atau Hive yang akan Anda jalankan menggunakan aplikasi EMR Tanpa Server. Untuk membuat bucket, ikuti petunjuk dalam [Membuat bucket](#) di Panduan Pengguna Amazon Simple Storage Service Console. Ganti referensi lebih lanjut *amzn-s3-demo-bucket* dengan nama bucket yang baru dibuat.

Buat EMR Studio untuk menjalankan beban kerja interaktif

Jika Anda ingin menggunakan EMR Tanpa Server untuk menjalankan kueri interaktif melalui notebook yang di-host di EMR Studio, Anda perlu menentukan bucket S3 dan [peran layanan](#) minimum untuk EMR Serverless untuk membuat Workspace. Untuk langkah-langkah penyiapan,

lihat [Menyiapkan Studio EMR di Panduan](#) Manajemen EMR Amazon. Untuk informasi selengkapnya tentang beban kerja interaktif, lihat [Jalankan beban kerja interaktif dengan EMR Serverless melalui EMR Studio](#).

Buat peran runtime pekerjaan

Job run di EMR Serverless menggunakan peran runtime yang memberikan izin terperinci untuk spesifik dan sumber daya saat runtime. Layanan AWS Dalam tutorial ini, bucket S3 publik menghosting data dan skrip. Ember *amzn-s3-demo-bucket* menyimpan output.

Untuk menyiapkan peran runtime pekerjaan, pertama-tama buat peran runtime dengan kebijakan kepercayaan sehingga EMR Tanpa Server dapat menggunakan peran baru tersebut. Selanjutnya, lampirkan kebijakan akses S3 yang diperlukan ke peran itu. Langkah-langkah berikut memandu Anda melalui proses tersebut.

Console

1. Arahkan ke konsol IAM di <https://console.aws.amazon.com/iam/>.
2. Di panel navigasi di sebelah kiri, pilih Kebijakan.
3. Pilih Buat Kebijakan.
4. Halaman Buat kebijakan terbuka di tab baru. Pilih editor Kebijakan sebagai Json dan Tempel kebijakan JSON di bawah ini.

Important

Ganti *amzn-s3-demo-bucket* dalam kebijakan di bawah ini dengan nama bucket aktual yang dibuat di [Siapkan penyimpanan untuk EMR Tanpa Server](#). Ini adalah kebijakan dasar untuk akses S3. Untuk contoh peran runtime pekerjaan lainnya, lihat [Peran runtime Job untuk Amazon EMR Tanpa Server](#).

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadAccessForEMRSamples",
```

```

    "Effect": "Allow",
    "Action": [
      "s3:GetObject",
      "s3:ListBucket"
    ],
    "Resource": [
      "arn:aws:s3::*.elasticmapreduce",
      "arn:aws:s3::*.elasticmapreduce/*"
    ]
  },
  {
    "Sid": "FullAccessToOutputBucket",
    "Effect": "Allow",
    "Action": [
      "s3:PutObject",
      "s3:GetObject",
      "s3:ListBucket",
      "s3:DeleteObject"
    ],
    "Resource": [
      "arn:aws:s3::amzn-s3-demo-bucket",
      "arn:aws:s3:::amzn-s3-demo-bucket/*"
    ]
  },
  {
    "Sid": "GlueCreateAndReadDataCatalog",
    "Effect": "Allow",
    "Action": [
      "glue:GetDatabase",
      "glue:CreateDatabase",
      "glue:GetDataBases",
      "glue:CreateTable",
      "glue:GetTable",
      "glue:UpdateTable",
      "glue:DeleteTable",
      "glue:GetTables",
      "glue:GetPartition",
      "glue:GetPartitions",
      "glue:CreatePartition",
      "glue:BatchCreatePartition",
      "glue:GetUserDefinedFunctions"
    ],
    "Resource": [
      "*"
    ]
  }
}

```

```

    ]
  }
]
}

```

5. Pilih Berikutnya untuk memasukkan nama kebijakan Anda, seperti `EMRServerlessS3AndGlueAccessPolicy` dan Buat kebijakan
6. Di panel navigasi kiri konsol IAM, pilih Peran.
7. Pilih Buat peran.
8. Untuk jenis peran, pilih Kebijakan kepercayaan khusus dan tempel kebijakan kepercayaan berikut. Ini memungkinkan pekerjaan yang dikirimkan ke aplikasi Amazon EMR Tanpa Server Anda untuk mengakses orang lain Layanan AWS atas nama Anda.

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowSTSAssumerole",
      "Effect": "Allow",
      "Principal": {
        "Service": "emr-serverless.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        }
      }
    }
  ]
}

```

9. Pilih Berikutnya untuk menavigasi ke halaman Tambahkan izin, lalu pilih `EMRServerlessAndGlueAccessPolicyS3`.
10. Di halaman Nama, tinjau, dan buat, untuk nama Peran, masukkan nama untuk peran Anda, misalnya, `EMRServerlessS3RuntimeRole`. Untuk membuat peran IAM ini, pilih Buat peran.

CLI

1. Buat file bernama `emr-serverless-trust-policy.json` yang berisi kebijakan kepercayaan yang akan digunakan untuk peran IAM. File harus berisi kebijakan berikut.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EMRServerlessTrustPolicy",
      "Effect": "Allow",
      "Principal": {
        "Service": "emr-serverless.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        }
      }
    }
  ]
}
```

2. Buat IAM role bernama `EMRServerlessS3RuntimeRole`. Gunakan kebijakan kepercayaan yang Anda buat di langkah sebelumnya.

```
aws iam create-role \
  --role-name EMRServerlessS3RuntimeRole \
  --assume-role-policy-document file://emr-serverless-trust-policy.json
```

Perhatikan ARN di output. Anda menggunakan ARN dari peran baru selama pengajuan pekerjaan, yang disebut setelah ini sebagai *job-role-arn*

3. Buat file bernama `emr-sample-access-policy.json` yang mendefinisikan kebijakan IAM untuk beban kerja Anda. Ini menyediakan akses baca ke skrip dan data yang disimpan di bucket S3 publik dan akses baca-tulis ke *amzn-s3-demo-bucket*

⚠ Important

Ganti *amzn-s3-demo-bucket* dalam kebijakan di bawah ini dengan nama bucket sebenarnya yang dibuat di [Siapkan penyimpanan untuk EMR Tanpa Server](#).. Ini adalah kebijakan dasar untuk akses AWS Glue dan S3. Untuk contoh peran runtime pekerjaan lainnya, lihat [Peran runtime Job untuk Amazon EMR Tanpa Server](#).

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadAccessForEMRSamples",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3::*.elasticmapreduce",
        "arn:aws:s3::*.elasticmapreduce/*"
      ]
    },
    {
      "Sid": "FullAccessToOutputBucket",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:ListBucket",
        "s3:DeleteObject"
      ],
      "Resource": [
        "arn:aws:s3::amzn-s3-demo-bucket",
        "arn:aws:s3::amzn-s3-demo-bucket/*"
      ]
    }
  ]
}
```

```

    "Sid": "GlueCreateAndReadDataCatalog",
    "Effect": "Allow",
    "Action": [
        "glue:GetDatabase",
        "glue:CreateDatabase",
        "glue:GetDataBases",
        "glue:CreateTable",
        "glue:GetTable",
        "glue:UpdateTable",
        "glue>DeleteTable",
        "glue:GetTables",
        "glue:GetPartition",
        "glue:GetPartitions",
        "glue:CreatePartition",
        "glue:BatchCreatePartition",
        "glue:GetUserDefinedFunctions"
    ],
    "Resource": [
        "*"
    ]
}
]
}

```

4. Buat kebijakan IAM bernama `EMRServerlessS3AndGlueAccessPolicy` dengan file kebijakan yang Anda buat di Langkah 3. Catat ARN di output, karena Anda akan menggunakan ARN dari kebijakan baru di langkah berikutnya.

```

aws iam create-policy \
  --policy-name EMRServerlessS3AndGlueAccessPolicy \
  --policy-document file://emr-sample-access-policy.json

```

Perhatikan ARN kebijakan baru di output. Anda akan menggantinya *policy-arn* di langkah berikutnya.

5. Lampirkan kebijakan IAM `EMRServerlessS3AndGlueAccessPolicy` ke peran runtime pekerjaan. `EMRServerlessS3RuntimeRole`

```

aws iam attach-role-policy \
  --role-name EMRServerlessS3RuntimeRole \
  --policy-arn policy-arn

```

Memulai dengan EMR Serverless dari konsol

Bagian ini menjelaskan bekerja dengan EMR Tanpa Server, termasuk membuat EMR Studio. Ini juga menjelaskan cara mengirimkan pekerjaan berjalan dan melihat log.

Langkah-langkah untuk menyelesaikan

- [Langkah 1: Buat aplikasi EMR Tanpa Server](#)
- [Langkah 2: Kirimkan pekerjaan atau beban kerja interaktif](#)
- [Langkah 3: Lihat UI aplikasi dan log](#)
- [Langkah 4: Membersihkan](#)

Langkah 1: Buat aplikasi EMR Tanpa Server

Buat aplikasi baru dengan EMR Serverless sebagai berikut.

1. [Masuk ke Konsol Manajemen AWS dan buka konsol EMR Amazon di https://console.aws.amazon.com/emr](https://console.aws.amazon.com/emr).
2. Di panel navigasi kiri, pilih EMR Tanpa Server untuk menavigasi ke halaman arahan EMR Tanpa Server.
3. Untuk membuat atau mengelola aplikasi EMR Tanpa Server, Anda memerlukan EMR Studio UI.
 - Jika Anda sudah memiliki EMR Studio di Wilayah AWS tempat Anda ingin membuat aplikasi, lalu pilih Kelola aplikasi untuk menavigasi ke EMR Studio Anda, atau pilih studio yang ingin Anda gunakan.
 - Jika Anda tidak memiliki EMR Studio di Wilayah AWS tempat Anda ingin membuat aplikasi, pilih Mulai lalu Pilih Buat dan luncurkan Studio. EMR Serverless membuat EMR Studio untuk Anda sehingga Anda dapat membuat dan mengelola aplikasi.
4. Di UI Buat studio yang terbuka di tab baru, masukkan nama, jenis, dan versi rilis untuk aplikasi Anda. Jika Anda hanya ingin menjalankan pekerjaan batch, pilih Gunakan pengaturan default untuk pekerjaan batch saja. Untuk beban kerja interaktif, pilih Gunakan pengaturan default untuk beban kerja interaktif. Anda juga dapat menjalankan pekerjaan batch pada aplikasi yang diaktifkan interaktif dengan opsi ini. Jika perlu, Anda dapat mengubah pengaturan ini nanti.

Untuk informasi selengkapnya, lihat [Membuat studio](#).

5. Pilih Buat aplikasi untuk membuat aplikasi pertama Anda.

Lanjutkan ke bagian berikutnya [Langkah 2: Kirimkan pekerjaan atau beban kerja interaktif](#) untuk mengirimkan pekerjaan atau beban kerja interaktif.

Langkah 2: Kirimkan pekerjaan atau beban kerja interaktif

Spark job run

Dalam tutorial ini, kita menggunakan PySpark script untuk menghitung jumlah kemunculan kata-kata unik di beberapa file teks. Bucket S3 publik dan hanya-baca menyimpan skrip dan kumpulan data.

Untuk menjalankan pekerjaan Spark

1. Unggah skrip sampel `wordcount.py` ke bucket baru Anda dengan perintah berikut.

```
aws s3 cp s3://us-east-1.elasticmapreduce/emr-containers/samples/wordcount/scripts/wordcount.py s3://amzn-s3-demo-bucket/scripts/
```

2. Menyelesaikan [Langkah 1: Buat aplikasi EMR Tanpa Server](#) membawa Anda ke halaman detail Aplikasi di EMR Studio. Di sana, pilih opsi Kirim pekerjaan.
3. Pada halaman Kirim pekerjaan, lengkapi yang berikut ini.
 - Di bidang Nama, masukkan nama yang ingin Anda panggil job run.
 - Di bidang peran Runtime, masukkan nama peran yang Anda buat. [Buat peran runtime pekerjaan](#)
 - Di bidang lokasi Script, masukkan `s3://amzn-s3-demo-bucket/scripts/wordcount.py` sebagai URI S3.
 - Di bidang argumen Script, masukkan `["s3://amzn-s3-demo-bucket/emr-serverless-spark/output"]`.
 - Di bagian properti Spark, pilih Edit sebagai teks dan masukkan konfigurasi berikut.

```
--conf spark.executor.cores=1 --conf spark.executor.memory=4g --  
conf spark.driver.cores=1 --conf spark.driver.memory=4g --conf  
spark.executor.instances=1
```

4. Untuk memulai pekerjaan, pilih Kirim pekerjaan.
5. Di tab Job runs, Anda akan melihat pekerjaan baru Anda berjalan dengan status Running.

Hive job run

Di bagian tutorial ini, kita membuat tabel, menyisipkan beberapa catatan, dan menjalankan kueri agregasi hitungan. Untuk menjalankan pekerjaan Hive, pertama-tama buat file yang berisi semua kueri Hive untuk dijalankan sebagai bagian dari pekerjaan tunggal, unggah file ke S3, dan tentukan jalur S3 ini saat memulai pekerjaan Hive.

Untuk menjalankan pekerjaan Hive

1. Buat file bernama `hive-query.sql` yang berisi semua kueri yang ingin Anda jalankan dalam pekerjaan Hive Anda.

```
create database if not exists emrserverless;
use emrserverless;
create table if not exists test_table(id int);
drop table if exists Values__Tmp__Table__1;
insert into test_table values (1),(2),(2),(3),(3),(3);
select id, count(id) from test_table group by id order by id desc;
```

2. Unggah `hive-query.sql` ke bucket S3 Anda dengan perintah berikut.

```
aws s3 cp hive-query.sql s3://amzn-s3-demo-bucket/emr-serverless-hive/query/hive-query.sql
```

3. Menyelesaikan [Langkah 1: Buat aplikasi EMR Tanpa Server](#) membawa Anda ke halaman detail Aplikasi di EMR Studio. Di sana, pilih opsi Kirim pekerjaan.
4. Pada halaman Kirim pekerjaan, lengkapi yang berikut ini.
 - Di bidang Nama, masukkan nama yang ingin Anda panggil job run.
 - Di bidang peran Runtime, masukkan nama peran yang Anda buat. [Buat peran runtime pekerjaan](#)
 - Di bidang lokasi Script, masukkan `s3://amzn-s3-demo-bucket/emr-serverless-hive/query/hive-query.sql` sebagai URI S3.
 - Di bagian Properti sarang, pilih Edit sebagai teks, dan masukkan konfigurasi berikut.

```
--hiveconf hive.log.explain.output=false
```

- Di bagian konfigurasi Job, pilih Edit sebagai JSON, dan masukkan JSON berikut.

```
{
```

```
"applicationConfiguration":
  [{
    "classification": "hive-site",
    "properties": {
      "hive.exec.scratchdir": "s3://amzn-s3-demo-bucket/emr-
serverless-hive/hive/scratch",
      "hive.metastore.warehouse.dir": "s3://amzn-s3-demo-bucket/emr-
serverless-hive/hive/warehouse",
      "hive.driver.cores": "2",
      "hive.driver.memory": "4g",
      "hive.tez.container.size": "4096",
      "hive.tez.cpu.vcores": "1"
    }
  ]
}
```

5. Untuk memulai pekerjaan, pilih Kirim pekerjaan.
6. Di tab Job runs, Anda akan melihat pekerjaan baru Anda berjalan dengan status Running.

Interactive workload

Dengan Amazon EMR 6.14.0 dan yang lebih tinggi, Anda dapat menggunakan notebook yang di-host di EMR Studio untuk menjalankan beban kerja interaktif untuk Spark di EMR Tanpa Server. Untuk informasi selengkapnya termasuk izin dan prasyarat, lihat [Jalankan beban kerja interaktif dengan EMR Serverless melalui EMR Studio](#)

Setelah Anda membuat aplikasi dan menyiapkan izin yang diperlukan, gunakan langkah-langkah berikut untuk menjalankan notebook interaktif dengan EMR Studio:

1. Arahkan ke tab Workspaces di EMR Studio. Jika Anda masih perlu mengonfigurasi lokasi penyimpanan Amazon S3 dan [peran layanan EMR Studio](#), pilih tombol Configure studio di spanduk di bagian atas layar.
2. Untuk mengakses buku catatan, pilih Workspace atau buat Workspace baru. Gunakan Quick launch untuk membuka Workspace Anda di tab baru.
3. Buka tab yang baru dibuka. Pilih ikon Compute dari navigasi kiri. Pilih EMR Tanpa Server sebagai tipe Compute.
4. Pilih aplikasi berkemampuan interaktif yang Anda buat di bagian sebelumnya.
5. Di bidang peran Runtime, masukkan nama peran IAM yang dapat diasumsikan oleh aplikasi EMR Tanpa Server Anda untuk menjalankan pekerjaan. Untuk mempelajari lebih lanjut

tentang peran runtime, lihat Peran [runtime Job](#) di Panduan Pengguna Tanpa Server Amazon EMR.

6. Pilih Lampirkan. Ini mungkin memakan waktu hingga satu menit. Halaman akan disegarkan saat dilampirkan.
7. Pilih kernel dan mulai notebook. Anda juga dapat menelusuri contoh notebook di EMR Serverless dan menyalinnya ke Workspace Anda. Untuk mengakses contoh buku catatan, navigasikan ke {...} menu di navigasi kiri dan telusuri buku catatan yang ada serverless di nama file notebook.
8. Di buku catatan, Anda dapat mengakses tautan log driver dan tautan ke Apache Spark UI, antarmuka waktu nyata yang menyediakan metrik untuk memantau pekerjaan Anda. Untuk informasi selengkapnya, lihat [Memantau aplikasi dan pekerjaan EMR Tanpa Server di Panduan Pengguna](#) Tanpa Server Amazon EMR.

Saat Anda melampirkan aplikasi ke ruang kerja Studio, aplikasi mulai terpicu secara otomatis jika aplikasi tersebut belum berjalan. Anda juga dapat memulai aplikasi terlebih dahulu dan menyiapkannya sebelum Anda melampirkannya ke ruang kerja.

Langkah 3: Lihat UI aplikasi dan log

Untuk melihat UI aplikasi, pertama-tama identifikasi pekerjaan yang dijalankan. Opsi untuk Spark UI atau Hive Tez UI tersedia di baris pertama opsi untuk pekerjaan itu, berdasarkan jenis pekerjaan. Pilih opsi yang sesuai.

Jika Anda memilih UI Spark, pilih tab Executors untuk melihat log driver dan pelaksana. Jika Anda memilih Hive Tez UI, pilih tab Semua Tugas untuk melihat log.

Setelah status job run ditampilkan sebagai Sukses, Anda dapat melihat output pekerjaan di bucket S3 Anda.

Langkah 4: Membersihkan

Meskipun aplikasi yang Anda buat harus berhenti otomatis setelah 15 menit tidak aktif, kami tetap menyarankan Anda merilis sumber daya yang tidak ingin Anda gunakan lagi.

Untuk menghapus aplikasi, navigasikan ke halaman Daftar aplikasi. Pilih aplikasi yang Anda buat dan pilih Tindakan → Berhenti untuk menghentikan aplikasi. Setelah aplikasi dalam STOPPED keadaan, pilih aplikasi yang sama dan pilih Tindakan → Hapus.

Untuk lebih banyak contoh menjalankan pekerjaan Spark dan Hive, lihat [Menggunakan konfigurasi Spark saat Anda menjalankan pekerjaan EMR Tanpa Server](#) dan [Menggunakan konfigurasi Hive saat Anda menjalankan pekerjaan EMR Tanpa Server](#)

Memulai dari AWS CLI

Memulai EMR Serverless dari perintah AWS CLI untuk membuat aplikasi, menjalankan pekerjaan, memeriksa output job run, dan menghapus sumber daya Anda.

Langkah 1: Buat aplikasi EMR Tanpa Server

Gunakan [emr-serverless create-application](#) perintah untuk membuat aplikasi EMR Serverless pertama Anda. Anda perlu menentukan jenis aplikasi dan label rilis Amazon EMR yang terkait dengan versi aplikasi yang ingin Anda gunakan. Nama aplikasi adalah opsional.

Spark

Untuk membuat aplikasi Spark, jalankan perintah berikut.

```
aws emr-serverless create-application \  
  --release-label emr-6.6.0 \  
  --type "SPARK" \  
  --name my-application
```

Hive

Untuk membuat aplikasi Hive, jalankan perintah berikut.

```
aws emr-serverless create-application \  
  --release-label emr-6.6.0 \  
  --type "HIVE" \  
  --name my-application
```

Perhatikan ID aplikasi yang dikembalikan dalam output. Anda akan menggunakan ID untuk memulai aplikasi dan selama pengiriman pekerjaan, yang disebut setelah ini sebagai *application-id*

Sebelum Anda melanjutkan ke [Langkah 2: Kirimkan pekerjaan ke aplikasi EMR Tanpa Server Anda](#), pastikan bahwa aplikasi Anda telah mencapai CREATED status dengan [get-application](#) API.

```
aws emr-serverless get-application \  
  --application-id application-id
```

```
--application-id application-id
```

EMR Tanpa Server menciptakan pekerja untuk mengakomodasi pekerjaan yang Anda minta. Secara default, ini dibuat sesuai permintaan, tetapi Anda juga dapat menentukan kapasitas pra-inisialisasi dengan mengatur `initialCapacity` parameter saat Anda membuat aplikasi. Anda juga dapat membatasi total kapasitas maksimum yang dapat digunakan aplikasi dengan `maximumCapacity` parameter tersebut. Untuk mempelajari selengkapnya tentang opsi ini, lihat [Mengkonfigurasi aplikasi saat bekerja dengan EMR Serverless](#).

Langkah 2: Kirimkan pekerjaan ke aplikasi EMR Tanpa Server Anda

Sekarang aplikasi EMR Tanpa Server Anda siap menjalankan pekerjaan.

Spark

Pada langkah ini, kami menggunakan PySpark skrip untuk menghitung jumlah kemunculan kata-kata unik di beberapa file teks. Bucket S3 publik dan hanya-baca menyimpan skrip dan kumpulan data. Aplikasi mengirimkan file output dan data log dari runtime Spark ke `/output` dan `/logs` direktori di bucket S3 yang Anda buat.

Untuk menjalankan pekerjaan Spark

1. Gunakan perintah berikut untuk menyalin skrip sampel yang akan kami jalankan ke bucket baru Anda.

```
aws s3 cp s3://us-east-1.elasticmapreduce/emr-containers/samples/wordcount/scripts/wordcount.py s3://amzn-s3-demo-bucket/scripts/
```

2. Dalam perintah berikut, ganti *application-id* dengan ID aplikasi Anda. Gantikan *job-role-arn* dengan peran runtime ARN yang Anda buat. [Buat peran runtime pekerjaan](#) Gantilah *job-run-name* dengan nama yang ingin Anda sebut pekerjaan Anda berjalan. Ganti semua *amzn-s3-demo-bucket* string dengan bucket Amazon S3 yang Anda buat, dan `/output` tambahkan ke path. Ini membuat folder baru di bucket Anda di mana EMR Serverless dapat menyalin file keluaran aplikasi Anda.

```
aws emr-serverless start-job-run \  
  --application-id application-id \  
  --execution-role-arn job-role-arn \  
  --name job-run-name \  
  --job-driver '{
```

```

    "sparkSubmit": {
      "entryPoint": "s3://amzn-s3-demo-bucket/scripts/wordcount.py",
      "entryPointArguments": ["s3://amzn-s3-demo-bucket/emr-serverless-
spark/output"],
      "sparkSubmitParameters": "--conf spark.executor.cores=1
--conf spark.executor.memory=4g --conf spark.driver.cores=1 --conf
spark.driver.memory=4g --conf spark.executor.instances=1"
    }
  }
}'

```

- Perhatikan ID job run yang dikembalikan dalam output. Ganti *job-run-id* dengan ID ini dalam langkah-langkah berikut.

Hive

Dalam tutorial ini, kita membuat tabel, menyisipkan beberapa catatan, dan menjalankan kueri agregasi hitungan. Untuk menjalankan pekerjaan Hive, pertama-tama buat file yang berisi semua kueri Hive untuk dijalankan sebagai bagian dari pekerjaan tunggal, unggah file ke S3, dan tentukan jalur S3 ini saat Anda memulai pekerjaan Hive.

Untuk menjalankan pekerjaan Hive

- Buat file bernama `hive-query.sql` yang berisi semua kueri yang ingin Anda jalankan dalam pekerjaan Hive Anda.

```

create database if not exists emrserverless;
use emrserverless;
create table if not exists test_table(id int);
drop table if exists Values__Tmp__Table__1;
insert into test_table values (1),(2),(2),(3),(3),(3);
select id, count(id) from test_table group by id order by id desc;

```

- Unggah `hive-query.sql` ke bucket S3 Anda dengan perintah berikut.

```

aws s3 cp hive-query.sql s3://amzn-s3-demo-bucket/emr-serverless-hive/query/hive-
query.sql

```

- Dalam perintah berikut, ganti *application-id* dengan ID aplikasi Anda sendiri. Gantikan *job-role-arn* dengan peran runtime ARN yang Anda buat. [Buat peran runtime pekerjaan](#) Ganti semua *amzn-s3-demo-bucket* string dengan bucket Amazon S3 yang Anda buat,

lalu `/output` tambahkan `/logs` dan ke path. Ini membuat folder baru di bucket Anda, di mana EMR Serverless dapat menyalin file output dan log aplikasi Anda.

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "hive": {
      "query": "s3://amzn-s3-demo-bucket/emr-serverless-hive/query/hive-
query.q1",
      "parameters": "--hiveconf hive.log.explain.output=false"
    }
  }' \
  --configuration-overrides '{
    "applicationConfiguration": [{
      "classification": "hive-site",
      "properties": {
        "hive.exec.scratchdir": "s3://amzn-s3-demo-bucket/emr-serverless-
hive/hive/scratch",
        "hive.metastore.warehouse.dir": "s3://amzn-s3-demo-bucket/emr-
serverless-hive/hive/warehouse",
        "hive.driver.cores": "2",
        "hive.driver.memory": "4g",
        "hive.tez.container.size": "4096",
        "hive.tez.cpu.vcores": "1"
      }
    }],
    "monitoringConfiguration": {
      "s3MonitoringConfiguration": {
        "logUri": "s3://amzn-s3-demo-bucket/emr-serverless-hive/logs"
      }
    }
  }'
```

- Perhatikan ID job run yang dikembalikan dalam output. Ganti *job-run-id* dengan ID ini dalam langkah-langkah berikut.

Langkah 3: Tinjau output pekerjaan Anda

Jalankan pekerjaan biasanya membutuhkan waktu 3-5 menit untuk menyelesaikannya.

Spark

Anda dapat memeriksa status pekerjaan Spark Anda dengan perintah berikut.

```
aws emr-serverless get-job-run \  
  --application-id application-id \  
  --job-run-id job-run-id
```

Dengan tujuan log Anda disetel ke `s3://amzn-s3-demo-bucket/emr-serverless-spark/logs`, Anda dapat menemukan log untuk pekerjaan khusus ini berjalan di bawah `s3://amzn-s3-demo-bucket/emr-serverless-spark/logs/applications/application-id/jobs/job-run-id`.

Untuk aplikasi Spark, EMR Serverless mendorong log peristiwa setiap 30 detik ke `sparklogs` folder di tujuan log S3 Anda. Ketika pekerjaan Anda selesai, log runtime Spark untuk driver dan pelaksana mengunggah ke folder yang diberi nama sesuai dengan jenis pekerja, seperti `driver` atau `executor`. Output dari PySpark pekerjaan diunggah ke `s3://amzn-s3-demo-bucket/output/`.

Hive

Anda dapat memeriksa status pekerjaan Hive Anda dengan perintah berikut.

```
aws emr-serverless get-job-run \  
  --application-id application-id \  
  --job-run-id job-run-id
```

Dengan tujuan log Anda disetel ke `s3://amzn-s3-demo-bucket/emr-serverless-hive/logs`, Anda dapat menemukan log untuk pekerjaan khusus ini berjalan di bawah `s3://amzn-s3-demo-bucket/emr-serverless-hive/logs/applications/application-id/jobs/job-run-id`.

Untuk aplikasi Hive, EMR Serverless terus mengunggah driver Hive ke folder, dan tugas Tez log ke folder, `HIVE_DRIVER` dari tujuan log S3 Anda dan `TEZ_TASK`. Setelah pekerjaan berjalan mencapai `SUCCEEDED` status, output kueri Hive Anda akan tersedia di lokasi Amazon S3 yang Anda tentukan `monitoringConfiguration` di bidang `configurationOverrides`

Langkah 4: Membersihkan

Setelah selesai mengerjakan tutorial ini, pertimbangkan untuk menghapus sumber daya yang Anda buat. Kami menyarankan Anda merilis sumber daya yang tidak ingin Anda gunakan lagi.

Hapus aplikasi Anda

Untuk menghapus aplikasi, gunakan perintah berikut.

```
aws emr-serverless delete-application \  
  --application-id application-id
```

Hapus bucket log S3 Anda

Untuk menghapus bucket logging dan output S3 Anda, gunakan perintah berikut. Ganti *amzn-s3-demo-bucket* dengan nama sebenarnya dari bucket S3 yang dibuat di.. [Siapkan penyimpanan untuk EMR Tanpa Server](#)

```
aws s3 rm s3://amzn-s3-demo-bucket --recursive  
aws s3api delete-bucket --bucket amzn-s3-demo-bucket
```

Hapus peran runtime pekerjaan Anda

Untuk menghapus peran runtime, lepaskan kebijakan dari peran. Anda kemudian dapat menghapus peran dan kebijakan.

```
aws iam detach-role-policy \  
  --role-name EMRServerlessS3RuntimeRole \  
  --policy-arn policy-arn
```

Untuk menghapus peran, gunakan perintah berikut.

```
aws iam delete-role \  
  --role-name EMRServerlessS3RuntimeRole
```

Untuk menghapus kebijakan yang dilampirkan pada peran, gunakan perintah berikut.

```
aws iam delete-policy \  
  --policy-arn policy-arn
```

Untuk lebih banyak contoh menjalankan pekerjaan Spark dan Hive, lihat [Menggunakan konfigurasi Spark saat Anda menjalankan pekerjaan EMR Tanpa Server](#) dan [Menggunakan konfigurasi Hive saat Anda menjalankan pekerjaan EMR Tanpa Server](#)

Berinteraksi dengan dan mengkonfigurasi aplikasi EMR Tanpa Server

Bagian ini mencakup cara berinteraksi dengan aplikasi Amazon EMR Tanpa Server Anda dengan aplikasi. AWS CLI Ini juga menjelaskan konfigurasi aplikasi, melakukan penyesuaian, dan default untuk mesin Spark dan Hive.

Topik

- [Status aplikasi](#)
- [Membuat aplikasi EMR Tanpa Server dari konsol EMR Studio](#)
- [Berinteraksi dengan aplikasi EMR Tanpa Server Anda di AWS CLI](#)
- [Mengkonfigurasi aplikasi saat bekerja dengan EMR Serverless](#)
- [Menyesuaikan gambar EMR Tanpa Server](#)
- [Mengkonfigurasi akses VPC untuk aplikasi EMR Tanpa Server untuk terhubung ke data](#)
- [Amazon EMR Opsi arsitektur tanpa server](#)
- [Konkurensi pekerjaan dan antrian untuk aplikasi EMR Tanpa Server](#)

Status aplikasi

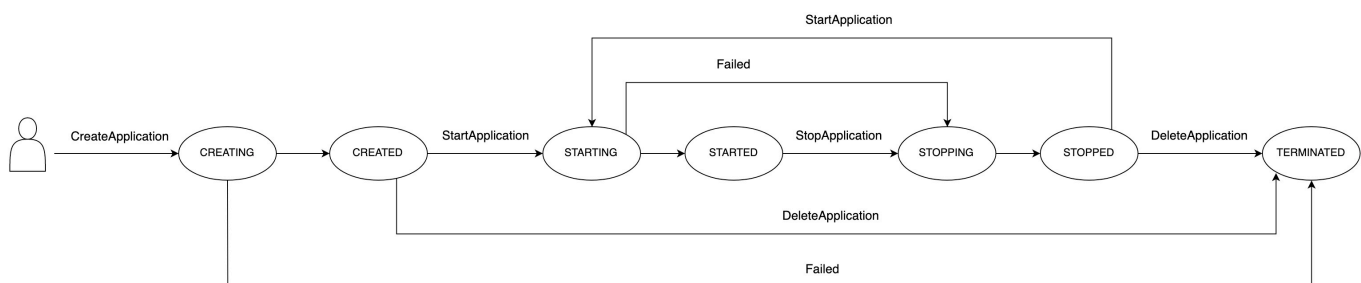
Saat Anda membuat aplikasi dengan EMR Serverless, aplikasi yang dijalankan memasuki status. CREATING Kemudian melewati status-status berikut sampai berhasil (keluar dengan kode 0) atau gagal (keluar dengan kode bukan nol).

Aplikasi dapat memiliki status berikut:

Status	Deskripsi
Creating	Aplikasi sedang dipersiapkan dan belum siap digunakan.
Dibuat	Aplikasi telah dibuat tetapi belum menyediakan kapasitas. Anda dapat memodifikasi aplikasi untuk mengubah konfigurasi kapasitas awalnya.

Status	Deskripsi
Starting	Aplikasi dimulai dan menyediakan kapasitas.
Dimulai	Aplikasi siap menerima pekerjaan baru. Aplikasi hanya menerima pekerjaan ketika berada di negara bagian ini.
Stopping	Semua pekerjaan telah selesai dan aplikasi merilis kapasitasnya.
Stopped	Aplikasi dihentikan dan tidak ada sumber daya yang berjalan pada aplikasi. Anda dapat memodifikasi aplikasi untuk mengubah konfigurasi kapasitas awalnya.
Diakhiri	Aplikasi telah dihentikan dan tidak muncul di daftar aplikasi Anda.

Diagram berikut menggambarkan lintasan status aplikasi EMR Tanpa Server.



Membuat aplikasi EMR Tanpa Server dari konsol EMR Studio

Dari konsol EMR Studio, buat, akses, dan kelola aplikasi EMR Tanpa Server. Untuk menavigasi ke konsol EMR Studio, ikuti petunjuk di [Memulai dari konsol](#).

Membuat aplikasi

Dengan halaman Buat aplikasi, buat aplikasi EMR Tanpa Server dengan mengikuti langkah-langkah ini.

1. Di bidang Nama, masukkan nama yang ingin Anda panggil aplikasi Anda.
2. Di bidang Type, pilih Spark atau Hive sebagai jenis aplikasi.
3. Di bidang Versi rilis, pilih nomor rilis EMR.
4. Dalam opsi Arsitektur, pilih arsitektur set instruksi yang akan digunakan. Untuk informasi lebih lanjut, lihat [Amazon EMR Opsi arsitektur tanpa server](#).
 - arm64 - arsitektur ARM 64-bit; untuk menggunakan prosesor Graviton
 - x86_64 - arsitektur x86 64-bit; untuk menggunakan prosesor berbasis x86
5. Ada dua opsi pengaturan aplikasi untuk bidang yang tersisa: pengaturan default dan pengaturan khusus. Bidang-bidang ini opsional.

Pengaturan default - Pengaturan default memungkinkan Anda membuat aplikasi dengan cepat dengan kapasitas pra-inisialisasi. Ini termasuk satu driver dan satu eksekutor untuk Spark, dan satu driver dan satu Tez Task for Hive. Pengaturan default tidak mengaktifkan konektivitas jaringan ke Anda VPCs. Aplikasi dikonfigurasi untuk berhenti jika idle selama 15 menit, dan otomatis dimulai pada pengiriman pekerjaan.

Pengaturan kustom - Pengaturan kustom memungkinkan Anda untuk memodifikasi properti berikut.

- Kapasitas pra-inisialisasi — Jumlah driver dan pelaksana atau pekerja Hive Tez Task, dan ukuran setiap pekerja.
- Batas aplikasi — Kapasitas maksimum aplikasi.
- Perilaku aplikasi — Perilaku auto-start dan auto-stop aplikasi.
- Koneksi jaringan — Konektivitas jaringan ke sumber daya VPC.
- Tag - Tag kustom yang menetapkan ke aplikasi.

Untuk informasi lebih lanjut tentang kapasitas pra-inisialisasi, batas aplikasi, dan perilaku aplikasi, lihat. [Mengkonfigurasi aplikasi saat bekerja dengan EMR Serverless](#) Untuk informasi lebih lanjut tentang konektivitas jaringan, lihat [Mengkonfigurasi akses VPC untuk aplikasi EMR Tanpa Server untuk terhubung ke data](#).

6. Untuk membuat aplikasi, pilih Buat aplikasi.

Daftar aplikasi dari konsol EMR Studio

Anda dapat mengakses semua aplikasi EMR Tanpa Server yang ada dari halaman Daftar aplikasi. Anda dapat memilih nama aplikasi untuk menavigasi ke halaman Detail untuk aplikasi itu.

Mengelola aplikasi dari konsol EMR Studio

Anda dapat melakukan tindakan berikut pada aplikasi baik dari halaman Daftar aplikasi atau dari halaman Detail aplikasi tertentu.

Mulai aplikasi

Pilih opsi ini untuk memulai aplikasi secara manual.

Hentikan aplikasi

Pilih opsi ini untuk menghentikan aplikasi secara manual. Aplikasi harus tidak memiliki pekerjaan yang berjalan untuk dihentikan. Untuk mempelajari lebih lanjut tentang transisi status aplikasi, lihat [Status aplikasi](#)

Konfigurasi aplikasi

Edit pengaturan opsional untuk aplikasi dari halaman Konfigurasi aplikasi. Anda dapat mengubah sebagian besar pengaturan aplikasi. Misalnya, ubah label rilis untuk aplikasi untuk memutakhirkannya ke versi EMR Amazon yang berbeda, atau alihkan arsitektur dari x86_64 ke arm64. Pengaturan opsional lainnya sama dengan yang ada di bagian Pengaturan kustom di halaman Buat aplikasi. Untuk informasi lebih lanjut tentang pengaturan aplikasi, lihat [Membuat aplikasi](#).

Hapus aplikasi

Pilih opsi ini untuk menghapus aplikasi secara manual. Anda harus menghentikan aplikasi untuk menghapusnya. Untuk mempelajari lebih lanjut tentang transisi status aplikasi, lihat [Status aplikasi](#)

Berinteraksi dengan aplikasi EMR Tanpa Server Anda di AWS CLI

Dari AWS CLI, buat, jelaskan, dan hapus aplikasi individual. Anda juga dapat membuat daftar semua aplikasi Anda sehingga mengaksesnya sekilas. Bagian ini menjelaskan cara melakukan

tindakan ini. Untuk operasi aplikasi lainnya, seperti memulai, menghentikan, dan pembaruan aplikasi, lihat Referensi API [EMR Tanpa Server](#). Untuk contoh cara menggunakan EMR Serverless API menggunakan AWS SDK untuk Java, lihat [contoh Java di repositori kami](#). GitHub Untuk contoh cara menggunakan EMR Serverless API menggunakan, AWS SDK for Python (Boto) lihat contoh [Python](#) di repositori kami. GitHub

Untuk membuat aplikasi, gunakan `create-application`. Anda harus menentukan SPARK atau HIVE sebagai `application-type`. Perintah ini mengembalikan ARN, nama, dan ID aplikasi.

```
aws emr-serverless create-application \  
--name my-application-name \  
--type 'application-type' \  
--release-label release-version
```

Untuk mendeskripsikan aplikasi, gunakan `get-application` dan sediakan `application-id`. Perintah ini mengembalikan status dan konfigurasi terkait kapasitas untuk aplikasi Anda.

```
aws emr-serverless get-application \  
--application-id application-id
```

Untuk membuat daftar semua aplikasi Anda, hubungi `list-applications`. Perintah ini mengembalikan properti yang sama seperti `get-application` tetapi mencakup semua aplikasi Anda.

```
aws emr-serverless list-applications
```

Untuk menghapus aplikasi Anda, hubungi `delete-application` dan sediakan aplikasi Anda `application-id`.

```
aws emr-serverless delete-application \  
--application-id application-id
```

Mengkonfigurasi aplikasi saat bekerja dengan EMR Serverless

Dengan EMR Tanpa Server, konfigurasi aplikasi yang Anda gunakan. Misalnya, atur kapasitas maksimum yang dapat ditingkatkan oleh aplikasi, konfigurasi kapasitas pra-inisialisasi agar pengemudi dan pekerja siap merespons, dan menentukan serangkaian konfigurasi runtime dan

pemantauan umum di tingkat aplikasi. Halaman berikut menjelaskan cara mengkonfigurasi aplikasi saat Anda menggunakan EMR Tanpa Server.

Topik

- [Memahami perilaku aplikasi di EMR Tanpa Server](#)
- [Pre-initialized kapasitas untuk bekerja dengan aplikasi di EMR Serverless](#)
- [Konfigurasi aplikasi default untuk EMR Serverless](#)

Memahami perilaku aplikasi di EMR Tanpa Server

Bagian ini menjelaskan perilaku pengiriman pekerjaan, konfigurasi kapasitas untuk penskalaan, dan pengaturan konfigurasi pekerja untuk EMR Tanpa Server.

Perilaku aplikasi default

Auto-start— Aplikasi secara default dikonfigurasi untuk memulai otomatis pada pengiriman pekerjaan. Anda dapat menonaktifkan fitur ini.

Auto-stop— Aplikasi secara default dikonfigurasi untuk berhenti otomatis saat idle selama 15 menit. Ketika aplikasi berubah ke STOPPED status, ia melepaskan kapasitas pra-diinisialisasi yang dikonfigurasi. Anda dapat mengubah jumlah waktu idle sebelum aplikasi berhenti otomatis, atau Anda dapat menonaktifkan fitur ini.

Kapasitas maksimum

Anda dapat mengonfigurasi kapasitas maksimum yang dapat ditingkatkan oleh aplikasi. Anda dapat menentukan kapasitas maksimum Anda dalam hal CPU, memori (GB), dan disk (GB).

Note

Ini adalah praktik terbaik untuk mengonfigurasi kapasitas maksimum Anda agar proporsional dengan ukuran pekerja Anda yang didukung dengan mengalikan jumlah pekerja dengan ukurannya. Misalnya, jika Anda ingin membatasi aplikasi Anda hingga 50 pekerja dengan 2 vCPU, 16 GB untuk memori, dan 20 GB untuk disk, atur kapasitas maksimum Anda menjadi 100 vCPU, 800 GB untuk memori, dan 1000 GB untuk disk.

Konfigurasi pekerja yang didukung

Tabel berikut mencantumkan konfigurasi dan ukuran pekerja yang didukung yang dapat ditentukan untuk EMR Tanpa Server. Konfigurasikan ukuran yang berbeda untuk driver dan pelaksana berdasarkan kebutuhan beban kerja Anda.

Konfigurasi dan ukuran pekerja

CPU	Memori	Penyimpanan fana default
1 vCPU	Minimum 2 GB, maksimum 8 GB, dengan peningkatan 1 GB	20 GB - 200 GB
2 vCPU	Minimum 4 GB, maksimum 16 GB, dengan peningkatan 1 GB	20 GB - 200 GB
4 vCPU	Minimum 8 GB, maksimum 30 GB, dengan peningkatan 1 GB	20 GB - 200 GB
8 vCPU	Minimum 16 GB, maksimum 60 GB, dengan peningkatan 4 GB	20 GB - 200 GB
16 vCPU	Minimum 32 GB, maksimum 120 GB, dengan peningkatan 8 GB	20 GB - 200 GB

CPU — Setiap pekerja dapat memiliki 1, 2, 4, 8, atau 16 vCPU.

Memori — Setiap pekerja memiliki memori, ditentukan dalam GB, dalam batas yang tercantum dalam tabel sebelumnya. Pekerjaan percikan memiliki overhead memori, yang berarti bahwa memori yang mereka gunakan lebih dari ukuran wadah yang ditentukan. Overhead ini ditentukan dengan properti `spark.driver.memoryOverhead` dan `spark.executor.memoryOverhead`. Overhead memiliki nilai default 10% dari memori kontainer, dengan minimal 384 MB. Anda harus mempertimbangkan overhead ini ketika Anda memilih ukuran pekerja.

Misalnya, Jika Anda memilih 4 vCPU untuk instans pekerja Anda, dan kapasitas penyimpanan pra-inisialisasi 30 GB, maka tetapkan nilai sekitar 27 GB sebagai memori pelaksana untuk pekerjaan

Spark Anda. Ini memaksimalkan pemanfaatan kapasitas pra-inisialisasi Anda. Memori yang dapat digunakan adalah 27 GB, ditambah 10% dari 27 GB (2,7 GB), dengan total 29,7 GB.

Disk — Anda dapat mengonfigurasi setiap pekerja dengan disk penyimpanan sementara dengan ukuran minimum 20 GB dan maksimum 200 GB. Anda hanya membayar penyimpanan tambahan melebihi 20 GB yang Anda konfigurasi per pekerja.

Pre-initialized kapasitas untuk bekerja dengan aplikasi di EMR Serverless

EMR Tanpa Server menyediakan fitur opsional yang membuat pengemudi dan pekerja pra-diinisialisasi dan siap merespons dalam hitungan detik. Ini secara efektif menciptakan kumpulan pekerja yang hangat untuk aplikasi. Fitur ini disebut kapasitas pra-inisialisasi. Untuk mengonfigurasi fitur ini, atur `initialCapacity` parameter aplikasi ke jumlah pekerja yang ingin diinisialisasi sebelumnya. Dengan kapasitas pekerja pra-inisialisasi, pekerjaan segera dimulai. Ini sangat ideal ketika Anda ingin menerapkan aplikasi berulang dan pekerjaan yang sensitif terhadap waktu.

Pre-initialized kapasitas membuat kumpulan pekerja yang hangat siap untuk pekerjaan dan sesi untuk memulai dalam hitungan detik. Anda akan membayar untuk pekerja pra-inisialisasi yang disediakan bahkan ketika aplikasi tidak digunakan, oleh karena itu kami sarankan untuk mengaktifkannya untuk kasus penggunaan yang mendapat manfaat dari waktu start-up yang cepat dan ukurannya untuk pemanfaatan sumber daya yang optimal. Aplikasi EMR Tanpa Server secara otomatis mati saat idle. Kami menyarankan agar fitur ini tetap aktif saat menggunakan pekerja yang telah diinisialisasi sebelumnya untuk menghindari biaya yang tidak terduga.

Ketika Anda mengirimkan pekerjaan, jika pekerja dari `initialCapacity` tersedia, pekerjaan menggunakan sumber daya tersebut untuk memulai operasinya. Jika pekerja tersebut sudah digunakan oleh pekerjaan lain, atau jika pekerjaan itu membutuhkan lebih banyak sumber daya daripada yang tersedia `initialCapacity`, maka aplikasi meminta dan mendapatkan pekerja tambahan, hingga batas maksimum sumber daya yang ditetapkan untuk aplikasi. Ketika pekerjaan selesai dijalankan, ia melepaskan pekerja yang digunakannya, dan jumlah sumber daya yang tersedia untuk aplikasi kembali ke `initialCapacity`. Aplikasi mempertahankan sumber `initialCapacity` daya bahkan setelah pekerjaan selesai berjalan. Aplikasi melepaskan kelebihan sumber daya di luar `initialCapacity` ketika pekerjaan tidak lagi membutuhkannya untuk dijalankan.

Pre-initialized kapasitas tersedia dan siap digunakan ketika aplikasi telah dimulai. Kapasitas pra-inisialisasi menjadi tidak aktif ketika aplikasi dihentikan. Aplikasi pindah ke `STARTED` status hanya jika kapasitas pra-inisialisasi yang diminta telah dibuat dan siap digunakan. Sepanjang waktu aplikasi dalam `STARTED` keadaan, EMR Serverless menjaga kapasitas pra-inisialisasi tersedia

untuk digunakan atau digunakan oleh pekerjaan atau beban kerja interaktif. Fitur ini mengembalikan kapasitas untuk kontainer yang dirilis atau gagal. Ini mempertahankan jumlah pekerja yang ditentukan `InitialCapacity` parameter. Keadaan aplikasi tanpa kapasitas pra-inisialisasi dapat segera berubah dari `CREATED` ke `STARTED`

Anda dapat mengonfigurasi aplikasi untuk melepaskan kapasitas pra-inisialisasi jika tidak digunakan untuk jangka waktu tertentu, dengan default 15 menit. Aplikasi yang dihentikan dimulai secara otomatis saat Anda mengirimkan pekerjaan baru. Anda dapat mengatur konfigurasi mulai dan berhenti otomatis ini saat Anda membuat aplikasi, atau mengubahnya saat aplikasi dalam `STOPPED` status `CREATED` atau.

Anda dapat mengubah `InitialCapacity` hitungan, dan menentukan konfigurasi komputasi seperti CPU, memori, dan disk, untuk setiap pekerja. Karena Anda tidak dapat membuat modifikasi sebagian, tentukan semua konfigurasi komputasi saat Anda mengubah nilai. Anda hanya dapat mengubah konfigurasi saat aplikasi dalam `STOPPED` status `CREATED` atau.

Note

Untuk mengoptimalkan penggunaan sumber daya aplikasi Anda, kami sarankan untuk menyelaraskan ukuran kontainer Anda dengan ukuran pekerja kapasitas yang telah diinisialisasi sebelumnya. Misalnya, jika Anda mengonfigurasi ukuran pelaksana Spark Anda menjadi 2 CPU dan memori Anda menjadi 8 GB, tetapi ukuran pekerja kapasitas pra-inisialisasi Anda adalah 4 CPU dengan memori 16 GB, maka pelaksana Spark hanya menggunakan setengah dari sumber daya pekerja ketika mereka ditugaskan untuk pekerjaan ini.

Menyesuaikan kapasitas pra-inisialisasi untuk Spark dan Hive

Anda dapat lebih lanjut menyesuaikan kapasitas pra-inisialisasi untuk beban kerja yang berjalan pada kerangka kerja data besar tertentu. Misalnya, ketika beban kerja berjalan di Apache Spark, tentukan berapa banyak pekerja yang memulai sebagai driver dan berapa banyak yang memulai sebagai pelaksana. Demikian pula, ketika Anda menggunakan Apache Hive, tentukan berapa banyak pekerja yang memulai sebagai driver Hive, dan berapa banyak yang harus menjalankan tugas Tez.

Mengonfigurasi aplikasi yang menjalankan Apache Hive dengan kapasitas pra-inisialisasi

Permintaan API berikut membuat aplikasi yang menjalankan Apache Hive berdasarkan rilis EMR Amazon `emr-6.6.0`. Aplikasi dimulai dengan 5 driver Hive pra-inisialisasi, masing-masing dengan 2

vCPU dan 4 GB memori, dan 50 pekerja tugas Tez pra-inisialisasi, masing-masing dengan 4 vCPU dan 8 GB memori. Ketika Hive query berjalan pada aplikasi ini, mereka pertama kali menggunakan pekerja pra-inisialisasi dan mulai mengeksekusi segera. Jika semua pekerja pra-inisialisasi sibuk dan lebih banyak pekerjaan Hive diajukan, aplikasi dapat menskalakan ke total 400 vCPU dan 1024 GB memori. Anda dapat secara opsional menghilangkan kapasitas untuk pekerja DRIVER atau pekerja TEZ_TASK

```
aws emr-serverless create-application \  
  --type "HIVE" \  
  --name my-application-name \  
  --release-label emr-6.6.0 \  
  --initial-capacity '{  
    "DRIVER": {  
      "workerCount": 5,  
      "workerConfiguration": {  
        "cpu": "2vCPU",  
        "memory": "4GB"  
      }  
    },  
    "TEZ_TASK": {  
      "workerCount": 50,  
      "workerConfiguration": {  
        "cpu": "4vCPU",  
        "memory": "8GB"  
      }  
    }  
  }' \  
  --maximum-capacity '{  
    "cpu": "400vCPU",  
    "memory": "1024GB"  
  }'
```

Mengkonfigurasi aplikasi yang menjalankan Apache Spark dengan kapasitas pra-inisialisasi

Permintaan API berikut membuat aplikasi yang menjalankan Apache Spark 3.2.0 berdasarkan Amazon EMR rilis 6.6.0. Aplikasi ini dimulai dengan 5 driver Spark pra-inisialisasi, masing-masing dengan 2 vCPU dan 4 GB memori, dan 50 pelaksana pra-inisialisasi, masing-masing dengan 4 vCPU dan 8 GB memori. Ketika pekerjaan Spark dijalankan pada aplikasi ini, mereka pertama kali menggunakan pekerja pra-inisialisasi dan mulai mengeksekusi segera. Jika semua pekerja pra-inisialisasi sibuk dan lebih banyak pekerjaan Spark diajukan, aplikasi dapat menskalakan ke total 400

vCPU dan 1024 GB memori. Anda dapat secara opsional menghilangkan kapasitas untuk DRIVER atau EXECUTOR

Note

Spark menambahkan overhead memori yang dapat dikonfigurasi, dengan nilai default 10%, ke memori yang diminta untuk driver dan pelaksana. Agar pekerjaan dapat menggunakan pekerja pra-inisialisasi, konfigurasi memori kapasitas awal harus lebih besar daripada memori yang diminta pekerjaan dan overhead.

```
aws emr-serverless create-application \  
--type "SPARK" \  
--name my-application-name \  
--release-label emr-6.6.0 \  
--initial-capacity '{  
  "DRIVER": {  
    "workerCount": 5,  
    "workerConfiguration": {  
      "cpu": "2vCPU",  
      "memory": "4GB"  
    }  
  },  
  "EXECUTOR": {  
    "workerCount": 50,  
    "workerConfiguration": {  
      "cpu": "4vCPU",  
      "memory": "8GB"  
    }  
  }  
}' \  
--maximum-capacity '{  
  "cpu": "400vCPU",  
  "memory": "1024GB"  
}'
```

Konfigurasi aplikasi default untuk EMR Serverless

Anda dapat menentukan satu set umum runtime dan konfigurasi pemantauan di tingkat aplikasi untuk semua pekerjaan yang Anda kirimkan di bawah aplikasi yang sama. Ini mengurangi overhead

tambahan yang terkait dengan kebutuhan untuk mengirimkan konfigurasi yang sama untuk setiap pekerjaan.

Anda dapat memodifikasi konfigurasi pada titik-titik waktu berikut:

- [Deklarasikan konfigurasi tingkat aplikasi saat pengajuan pekerjaan.](#)
- [Ganti konfigurasi default selama menjalankan pekerjaan.](#)

Bagian berikut memberikan rincian lebih lanjut dan contoh untuk konteks lebih lanjut.

Mendeklarasikan konfigurasi di tingkat aplikasi

Anda dapat menentukan pencatatan tingkat aplikasi dan properti konfigurasi runtime untuk pekerjaan yang Anda kirimkan di bawah aplikasi.

monitoringConfiguration

Untuk menentukan konfigurasi log untuk pekerjaan yang Anda kirimkan dengan aplikasi, gunakan [monitoringConfiguration](#) bidang. Untuk informasi lebih lanjut tentang pencatatan untuk EMR Tanpa Server, lihat. [Menyimpan log](#)

runtimeConfiguration

Untuk menentukan properti konfigurasi runtime seperti `spark-defaults`, berikan objek konfigurasi di `runtimeConfiguration` bidang. Ini memengaruhi konfigurasi default untuk semua pekerjaan yang Anda kirimkan dengan aplikasi. Untuk informasi lebih lanjut, lihat [Parameter penggantian konfigurasi sarang](#) dan [Parameter penggantian konfigurasi percikan](#).

Klasifikasi konfigurasi yang tersedia bervariasi menurut rilis EMR Tanpa Server tertentu. Misalnya, klasifikasi untuk Log4j kustom `spark-driver-log4j2` dan hanya `spark-executor-log4j2` tersedia dengan rilis 6.8.0 dan yang lebih tinggi. Untuk daftar properti khusus aplikasi, lihat dan. [Properti pekerjaan percikan](#) [Properti pekerjaan sarang](#)

Anda juga dapat mengonfigurasi [properti Apache Log4j2](#), [AWS Secrets Manager untuk perlindungan data](#), dan [runtime Java 17](#) di tingkat aplikasi.

Untuk meneruskan rahasia Secrets Manager di tingkat aplikasi, lampirkan kebijakan berikut kepada pengguna dan peran yang perlu membuat atau memperbarui aplikasi EMR Tanpa Server dengan rahasia.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SecretsManagerPolicy",
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue",
        "secretsmanager:DescribeSecret"
      ],
      "Resource": [
        "arn:aws:secretsmanager:us-east-1:123456789012:secret:my-secret-
name-123abc"
      ]
    },
    {
      "Sid": "KMSDecryptPolicy",
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": [
        "arn:aws:kms:us-
east-1:123456789012:key/12345678-1234-1234-1234-123456789012"
      ]
    }
  ]
}
```

Untuk informasi selengkapnya tentang membuat kebijakan khusus untuk rahasia, lihat [contoh kebijakan AWS Secrets Manager Izin](#) di Panduan AWS Secrets Manager Pengguna.

Note

runtimeConfigurationYang Anda tentukan di peta tingkat aplikasi ke applicationConfiguration dalam [StartJobRunAPI](#).

Contoh deklarasi

Contoh berikut menunjukkan bagaimana untuk mendeklarasikan konfigurasi default dengan `create-application`

```
aws emr-serverless create-application \
  --release-label release-version \
  --type SPARK \
  --name my-application-name \
  --runtime-configuration '[
    {
      "classification": "spark-defaults",
      "properties": {
        "spark.driver.cores": "4",
        "spark.executor.cores": "2",
        "spark.driver.memory": "8G",
        "spark.executor.memory": "8G",
        "spark.executor.instances": "2",

        "spark.hadoop.javax.jdo.option.ConnectionDriverName": "org.mariadb.jdbc.Driver",
        "spark.hadoop.javax.jdo.option.ConnectionURL": "jdbc:mysql://db-host:db-
port/db-name",
        "spark.hadoop.javax.jdo.option.ConnectionUserName": "connection-user-
name",
        "spark.hadoop.javax.jdo.option.ConnectionPassword":
"EMR.secret@SecretID"
      }
    },
    {
      "classification": "spark-driver-log4j2",
      "properties": {
        "rootLogger.level": "error",
        "logger.IdentifierForClass.name": "classpathForSettingLogger",
        "logger.IdentifierForClass.level": "info"
      }
    }
  ]' \
  --monitoring-configuration '{
    "s3MonitoringConfiguration": {
      "logUri": "s3://amzn-s3-demo-logging-bucket/logs/app-level"
    },
    "managedPersistenceMonitoringConfiguration": {
      "enabled": false
    }
  }
```

```
}'
```

Mengganti konfigurasi selama menjalankan pekerjaan

Anda dapat menentukan penggantian konfigurasi untuk konfigurasi aplikasi dan konfigurasi pemantauan dengan API. [StartJobRun](#) EMR Serverless kemudian menggabungkan konfigurasi yang Anda tentukan di tingkat aplikasi dan tingkat pekerjaan untuk menentukan konfigurasi untuk pelaksanaan pekerjaan.

Tingkat granularitas saat penggabungan terjadi adalah sebagai berikut:

- [ApplicationConfiguration](#)- Jenis klasifikasi, misalnya `spark-defaults`.
- [MonitoringConfiguration](#)- Jenis konfigurasi, misalnya `3MonitoringConfiguration`.

Note

Prioritas konfigurasi yang Anda berikan [StartJobRun](#) menggantikan konfigurasi yang Anda berikan di tingkat aplikasi.

Untuk informasi lebih lanjut peringkat prioritas, lihat [Parameter penggantian konfigurasi sarang](#) dan [Parameter penggantian konfigurasi percikan](#).

Ketika Anda memulai pekerjaan, jika Anda tidak menentukan konfigurasi tertentu, itu akan diwarisi dari aplikasi. Jika Anda mendeklarasikan konfigurasi di tingkat pekerjaan, Anda dapat melakukan operasi berikut:

- Ganti konfigurasi yang ada - Berikan parameter konfigurasi yang sama dalam `StartJobRun` permintaan dengan nilai override Anda.
- Tambahkan konfigurasi tambahan - Tambahkan parameter konfigurasi baru dalam `StartJobRun` permintaan dengan nilai yang ingin Anda tentukan.
- Hapus konfigurasi yang ada - Untuk menghapus konfigurasi runtime aplikasi, berikan kunci untuk konfigurasi yang ingin Anda hapus, dan berikan deklarasi kosong `{}` untuk konfigurasi. Kami tidak menyarankan untuk menghapus klasifikasi apa pun yang berisi parameter yang diperlukan untuk menjalankan pekerjaan. Misalnya, jika Anda mencoba menghapus [properti yang diperlukan untuk pekerjaan Hive, pekerjaan](#) itu akan gagal.

Untuk menghapus konfigurasi pemantauan aplikasi, gunakan metode yang sesuai untuk jenis konfigurasi yang relevan:

- **cloudWatchLoggingConfiguration**- Untuk menghapus `cloudWatchLogging`, berikan bendera yang diaktifkan sebagai `false`.
- **managedPersistenceMonitoringConfiguration**- Untuk menghapus pengaturan persistensi terkelola dan kembali ke status default yang diaktifkan, berikan deklarasi kosong `{}` untuk konfigurasi.
- **s3MonitoringConfiguration**- Untuk menghapus `s3MonitoringConfiguration`, berikan deklarasi kosong `{}` untuk konfigurasi.

Contoh mengesampingkan

Contoh berikut menunjukkan operasi yang berbeda yang dapat Anda lakukan selama pengajuan pekerjaan `distart-job-run`.

```
aws emr-serverless start-job-run \
  --application-id your-application-id \
  --execution-role-arn your-job-role-arn \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "s3://us-east-1.elasticmapreduce/emr-containers/samples/
wordcount/scripts/wordcount.py",
      "entryPointArguments": ["s3://amzn-s3-demo-destination-bucket1/
wordcount_output"]
    }
  }' \
  --configuration-overrides '{
    "applicationConfiguration": [
      {
        // Override existing configuration for spark-defaults in the
application
        "classification": "spark-defaults",
        "properties": {
          "spark.driver.cores": "2",
          "spark.executor.cores": "1",
          "spark.driver.memory": "4G",
          "spark.executor.memory": "4G"
        }
      }
    ],
  }
```

```
        // Add configuration for spark-executor-log4j2
        "classification": "spark-executor-log4j2",
        "properties": {
            "rootLogger.level": "error",
            "logger.IdentifierForClass.name": "classpathForSettingLogger",
            "logger.IdentifierForClass.level": "info"
        }
    },
    {
        // Remove existing configuration for spark-driver-log4j2 from the
application
        "classification": "spark-driver-log4j2",
        "properties": {}
    }
],
"monitoringConfiguration": {
    "managedPersistenceMonitoringConfiguration": {
        // Override existing configuration for managed persistence
        "enabled": true
    },
    "s3MonitoringConfiguration": {
        // Remove configuration of S3 monitoring
    },
    "cloudWatchLoggingConfiguration": {
        // Add configuration for CloudWatch logging
        "enabled": true
    }
}
}'
```

Pada saat pelaksanaan pekerjaan, klasifikasi dan konfigurasi berikut akan berlaku berdasarkan peringkat penggantian prioritas yang dijelaskan dalam dan. [Parameter penggantian konfigurasi sarang](#) [Parameter penggantian konfigurasi percikan](#)

- Klasifikasi spark-defaults akan diperbarui dengan properti yang ditentukan di tingkat pekerjaan. Hanya properti yang termasuk dalam StartJobRun dipertimbangkan untuk klasifikasi ini.
- Klasifikasi spark-executor-log4j2 akan ditambahkan dalam daftar klasifikasi yang ada.
- Klasifikasi spark-driver-log4j2 akan dihapus.
- Konfigurasi untuk managedPersistenceMonitoringConfiguration akan diperbarui dengan konfigurasi di tingkat pekerjaan.

- Konfigurasi untuk `s3MonitoringConfiguration` akan dihapus.
- Konfigurasi untuk `cloudWatchLoggingConfiguration` akan ditambahkan ke konfigurasi pemantauan yang ada.

Menyesuaikan gambar EMR Tanpa Server

Dimulai dengan Amazon EMR 6.9.0, gunakan gambar khusus untuk mengemas dependensi aplikasi dan lingkungan runtime ke dalam satu wadah dengan Amazon EMR Tanpa Server. Ini menyederhanakan cara Anda mengelola dependensi beban kerja dan membuat paket Anda lebih portabel. Saat Anda menyesuaikan gambar EMR Tanpa Server Anda, ini memberikan manfaat berikut:

- Menginstal dan mengonfigurasi paket yang dioptimalkan untuk beban kerja Anda. Paket-paket ini tidak tersedia secara luas di distribusi publik lingkungan runtime Amazon EMR.
- Mengintegrasikan EMR Tanpa Server dengan proses pembuatan, pengujian, dan penerapan yang sudah ada saat ini dalam organisasi Anda, termasuk pengembangan dan pengujian lokal.
- Menerapkan proses keamanan yang telah ditetapkan, seperti pemindaian gambar, yang memenuhi persyaratan kepatuhan dan tata kelola dalam organisasi Anda.
- Memungkinkan Anda menggunakan versi JDK dan Python Anda sendiri untuk aplikasi Anda.

EMR Serverless menyediakan gambar yang digunakan sebagai basis Anda saat Anda membuat gambar Anda sendiri. Gambar dasar menyediakan stoples, konfigurasi, dan pustaka penting bagi gambar untuk berinteraksi dengan EMR Tanpa Server. Anda dapat menemukan gambar dasar di [Galeri Publik Amazon ECR](#). Gunakan gambar yang cocok dengan jenis aplikasi Anda (Spark atau Hive) dan versi rilis. Misalnya, jika Anda membuat aplikasi di Amazon EMR rilis 6.9.0, gunakan gambar berikut.

Tipe	Citra
Spark	<code>public.ecr.aws/emr-serverless/spark/emr-6.9.0:latest</code>
Hive	<code>public.ecr.aws/emr-serverless/hive/emr-6.9.0:latest</code>

Prasyarat

Sebelum Anda membuat gambar khusus EMR Tanpa Server, lengkapi prasyarat ini.

1. Buat repositori Amazon ECR sama dengan Wilayah AWS yang Anda gunakan untuk meluncurkan aplikasi EMR Tanpa Server. Untuk membuat repositori pribadi Amazon ECR, lihat [Membuat repositori pribadi](#).
2. Untuk memberi pengguna akses ke repositori Amazon ECR Anda, tambahkan kebijakan berikut ke pengguna dan peran yang membuat atau memperbarui aplikasi EMR Tanpa Server dengan gambar dari repositori ini.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ECRRepositoryListGetPolicy",
      "Effect": "Allow",
      "Action": [
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "ecr:DescribeImages"
      ],
      "Resource": [
        "arn:aws:ecr:*:123456789012:repository/my-repo"
      ]
    }
  ]
}
```

Untuk lebih banyak contoh kebijakan berbasis identitas Amazon ECR, lihat contoh kebijakan berbasis identitas [Amazon Elastic Container Registry](#).

Langkah 1: Buat gambar khusus dari gambar dasar EMR Serverless

Pertama, buat [Dockerfile](#) yang dimulai dengan FROM instruksi yang menggunakan gambar dasar pilihan Anda. Setelah FROM instruksi, sertakan modifikasi apa pun yang ingin Anda buat pada gambar. Gambar dasar secara otomatis menyetel USER ke hadoop. Pengaturan ini tidak memiliki izin untuk semua modifikasi yang Anda sertakan. Sebagai solusinya, atur USER ke root, ubah

gambar Anda, lalu atur kembali keUSER. hadoop :hadoop Untuk merujuk ke sampel untuk kasus penggunaan umum, lihat[Menggunakan gambar kustom dengan EMR Serverless](#).

```
# Dockerfile
FROM public.ecr.aws/emr-serverless/spark/emr-6.9.0:latest

USER root
# MODIFICATIONS GO HERE

# EMRS runs the image as hadoop
USER hadoop:hadoop
```

Setelah Anda memiliki Dockerfile, buat gambar dengan perintah berikut.

```
# build the docker image
docker build . -t aws-account-id.dkr.ecr.region.amazonaws.com/my-repository[:tag]or[@digest]
```

Langkah 2: Validasi gambar secara lokal

EMR Tanpa Server menyediakan alat offline yang dapat memeriksa gambar kustom Anda secara statis untuk memvalidasi file dasar, variabel lingkungan, dan konfigurasi gambar yang benar. Untuk informasi tentang cara menginstal dan menjalankan alat, lihat CLI [Gambar Tanpa Server EMR Amazon](#). GitHub

Setelah Anda menginstal alat, jalankan perintah berikut untuk memvalidasi gambar:

```
amazon-emr-serverless-image \
validate-image -r emr-6.9.0 -t spark \
-i aws-account-id.dkr.ecr.region.amazonaws.com/my-repository:tag/@digest
```

Outputnya tampak mirip dengan yang berikut ini.

```
Amazon EMR Serverless - Image CLI
Version: 0.0.1
... Checking if docker cli is installed
... Checking Image Manifest
[INFO] Image ID: 9e2f4359cf5beb466a8a2ed047ab61c9d37786c555655fc122272758f761b41a
[INFO] Created On: 2022-12-02T07:46:42.586249984Z
[INFO] Default User Set to hadoop:hadoop : PASS
[INFO] Working Directory Set to : PASS
```

```
[INFO] Entrypoint Set to /usr/bin/entrypoint.sh : PASS
[INFO] HADOOP_HOME is set with value: /usr/lib/hadoop : PASS
[INFO] HADOOP_LIBEXEC_DIR is set with value: /usr/lib/hadoop/libexec : PASS
[INFO] HADOOP_USER_HOME is set with value: /home/hadoop : PASS
[INFO] HADOOP_YARN_HOME is set with value: /usr/lib/hadoop-yarn : PASS
[INFO] HIVE_HOME is set with value: /usr/lib/hive : PASS
[INFO] JAVA_HOME is set with value: /etc/alternatives/jre : PASS
[INFO] TEZ_HOME is set with value: /usr/lib/tez : PASS
[INFO] YARN_HOME is set with value: /usr/lib/hadoop-yarn : 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 hadoop-yarn-jars in /usr/lib/hadoop-yarn: PASS
[INFO] File Structure Test for hive-bin-files in /usr/bin: PASS
[INFO] File Structure Test for hive-jars in /usr/lib/hive/lib: PASS
[INFO] File Structure Test for java-bin in /etc/alternatives/jre/bin: PASS
[INFO] File Structure Test for tez-jars in /usr/lib/tez: PASS
-----
Overall Custom Image Validation Succeeded.
-----
```

Langkah 3: Unggah gambar ke repositori Amazon ECR Anda

Dorong gambar Amazon ECR Anda ke repositori Amazon ECR Anda dengan perintah berikut. Pastikan Anda memiliki izin IAM yang benar untuk mendorong gambar ke repositori Anda. Untuk informasi selengkapnya, lihat [Mendorong gambar](#) di Panduan Pengguna Amazon ECR.

```
# login to ECR repo
aws ecr get-login-password --region region | docker login --username AWS --password-
stdin aws-account-id.dkr.ecr.region.amazonaws.com

# push the docker image
docker push aws-account-id.dkr.ecr.region.amazonaws.com/my-repository:tag/@digest
```

Langkah 4: Buat atau perbarui aplikasi dengan gambar khusus

Pilih Konsol Manajemen AWS tab atau AWS CLI tab sesuai dengan bagaimana Anda ingin meluncurkan aplikasi Anda, lalu selesaikan langkah-langkah berikut.

Console

1. [Masuk ke konsol EMR Studio di https://console.aws.amazon.com/emr](https://console.aws.amazon.com/emr). Arahkan ke aplikasi Anda, atau buat aplikasi baru dengan instruksi di [Buat aplikasi](#).

2. Untuk menentukan gambar kustom saat Anda membuat atau memperbarui aplikasi EMR Tanpa Server, pilih Pengaturan khusus di opsi pengaturan aplikasi.
3. Di bagian Pengaturan gambar kustom, pilih kotak centang Gunakan gambar khusus dengan aplikasi ini.
4. Tempelkan URI image Amazon ECR ke bidang Image URI. EMR Tanpa Server menggunakan gambar ini untuk semua jenis pekerja untuk aplikasi. Atau, Anda dapat memilih Gambar kustom yang berbeda dan menempelkan gambar ECR Amazon yang berbeda URIs untuk setiap jenis pekerja.

CLI

- Buat aplikasi dengan `image-configuration` parameter. EMR Tanpa Server menerapkan pengaturan ini ke semua jenis pekerja.

```
aws emr-serverless create-application \  
--release-label emr-6.9.0 \  
--type SPARK \  
--image-configuration '{  
  "imageUri": "aws-account-id.dkr.ecr.region.amazonaws.com/my-repository:tag/  
@digest"  
}'
```

Untuk membuat aplikasi dengan pengaturan gambar yang berbeda untuk setiap jenis pekerja, gunakan `worker-type-specifications` parameter.

```
aws emr-serverless create-application \  
--release-label emr-6.9.0 \  
--type SPARK \  
--worker-type-specifications '{  
  "Driver": {  
    "imageConfiguration": {  
      "imageUri": "aws-account-id.dkr.ecr.region.amazonaws.com/my-  
repository:tag/@digest"  
    }  
  },  
  "Executor" : {  
    "imageConfiguration": {  
      "imageUri": "aws-account-id.dkr.ecr.region.amazonaws.com/my-  
repository:tag/@digest"  
    }  
  }  
}'
```

```
}
}'
```

Untuk memperbarui aplikasi, gunakan `image-configuration` parameter. EMR Tanpa Server menerapkan pengaturan ini ke semua jenis pekerja.

```
aws emr-serverless update-application \
--application-id application-id \
--image-configuration '{
  "imageUri": "aws-account-id.dkr.ecr.region.amazonaws.com/my-repository:tag/
@digest"
}'
```

Langkah 5: Izinkan EMR Tanpa Server untuk mengakses repositori gambar kustom

Tambahkan kebijakan sumber daya berikut ke repositori Amazon ECR untuk mengizinkan prinsipal layanan EMR Tanpa Server menggunakan `get`, `describe` dan permintaan dari repositori ini. `download`

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EmrServerlessCustomImageSupport",
      "Effect": "Allow",
      "Principal": {
        "Service": "emr-serverless.amazonaws.com"
      },
      "Action": [
        "ecr:BatchGetImage",
        "ecr:DescribeImages",
        "ecr:GetDownloadUrlForLayer"
      ],
      "Resource": "arn:aws:ecr:*:123456789012:repository/my-repo",
      "Condition": {
        "ArnLike": {
```

```
    "aws:SourceArn": "arn:aws:emr-serverless:*:123456789012:/applications/
  "*"
    }
  }
}
]
```

Sebagai praktik terbaik keamanan, tambahkan kunci `aws:SourceArn` kondisi ke kebijakan repositori. Kunci kondisi global IAM `aws:SourceArn` memastikan bahwa EMR Tanpa Server menggunakan repositori hanya untuk ARN aplikasi. Untuk informasi selengkapnya tentang kebijakan repositori Amazon ECR, lihat [Membuat](#) repositori pribadi.

Pertimbangan dan batasan

Saat Anda bekerja dengan gambar khusus, pertimbangkan hal berikut:

- Gunakan gambar dasar yang benar yang cocok dengan jenis (Spark atau Hive) dan label rilis (misalnya, `emr-6.9.0`) untuk aplikasi Anda.
- EMR Tanpa Server mengabaikan `[CMD]` atau `[ENTRYPOINT]` instruksi dalam file Docker. Gunakan instruksi umum dalam file Docker, seperti `[COPY]`, `[RUN]`, dan `[WORKDIR]`.
- Jangan memodifikasi variabel lingkungan `JAVA_HOMESPARK_HOME`, `HIVE_HOME`, `TEZ_HOME` ketika Anda membuat gambar kustom.
- Gambar kustom tidak boleh melebihi 10 GB dalam ukuran.
- Jika Anda memodifikasi binari atau toples di gambar dasar Amazon EMR, ini dapat menyebabkan kegagalan aplikasi atau peluncuran pekerjaan.
- Repositori Amazon ECR harus sama dengan Wilayah AWS yang Anda gunakan untuk meluncurkan aplikasi EMR Tanpa Server.

Mengkonfigurasi akses VPC untuk aplikasi EMR Tanpa Server untuk terhubung ke data

Anda dapat mengonfigurasi aplikasi EMR Tanpa Server untuk terhubung ke penyimpanan data dalam VPC Anda, seperti kluster Amazon Redshift, database Amazon RDS, atau bucket Amazon S3 dengan titik akhir VPC. Aplikasi EMR Tanpa Server Anda memiliki konektivitas keluar ke

penyimpanan data dalam VPC Anda. Secara default, EMR Tanpa Server memblokir akses masuk ke aplikasi Anda dan akses internet keluar untuk meningkatkan keamanan.

Note

Anda harus mengkonfigurasi akses VPC jika Anda ingin menggunakan database metastore Hive eksternal untuk aplikasi Anda. [Untuk informasi tentang cara mengkonfigurasi metastore Hive eksternal, lihat konfigurasi Metastore.](#)

Buat aplikasi

Pada halaman Buat aplikasi, pilih pengaturan khusus dan tentukan VPC, subnet, dan grup keamanan yang dapat digunakan aplikasi EMR Tanpa Server.

VPCs

Pilih nama virtual private cloud (VPC) yang berisi penyimpanan data Anda. Halaman Buat aplikasi mencantumkan semua VPCs untuk pilihan Anda Wilayah AWS.

Subnet

Pilih subnet dalam VPC yang berisi penyimpanan data Anda. Halaman Buat aplikasi mencantumkan semua subnet untuk penyimpanan data di VPC Anda. Subnet publik dan pribadi didukung. Anda dapat meneruskan subnet pribadi atau publik ke aplikasi Anda. Pilihan apakah akan memiliki subnet publik atau pribadi memiliki beberapa pertimbangan terkait yang harus diperhatikan.

Untuk subnet pribadi:

- Tabel rute terkait tidak boleh memiliki gateway internet.
- Untuk konektivitas keluar ke internet, jika diperlukan, konfigurasi rute keluar menggunakan NAT Gateway. Untuk mengonfigurasi Gateway NAT, lihat gateway [NAT](#).
- Untuk konektivitas Amazon S3, konfigurasi NAT Gateway atau titik akhir VPC. Untuk mengonfigurasi titik akhir VPC S3, lihat [Buat titik akhir gateway](#).
- Jika Anda mengonfigurasi titik akhir VPC S3 dan melampirkan kebijakan titik akhir untuk mengontrol akses, ikuti petunjuk di [Logging for EMR Serverless dengan penyimpanan terkelola untuk memberikan izin bagi EMR Tanpa Server untuk menyimpan](#) dan menyajikan log aplikasi.

- Untuk konektivitas ke yang lain Layanan AWS di luar VPC, seperti ke Amazon DynamoDB, konfigurasi titik akhir VPC atau gateway NAT. Untuk mengonfigurasi titik akhir VPC Layanan AWS, lihat Bekerja [dengan titik akhir VPC](#).

Note

Saat Anda menyiapkan aplikasi Amazon EMR Tanpa Server di subnet pribadi, kami sarankan Anda juga menyiapkan titik akhir VPC untuk Amazon S3. Jika aplikasi EMR Tanpa Server Anda berada dalam subnet pribadi tanpa titik akhir VPC untuk Amazon S3, Anda dikenakan biaya gateway NAT tambahan yang terkait dengan lalu lintas S3. Ini karena lalu lintas antara aplikasi EMR Anda dan Amazon S3 tidak akan tetap berada dalam VPC Anda ketika titik akhir VPC tidak dikonfigurasi.

Untuk subnet publik:

- Ini memiliki rute ke Internet Gateway.
- Anda harus memastikan konfigurasi grup keamanan yang tepat untuk mengontrol lalu lintas keluar.

Pekerja dapat terhubung ke penyimpanan data dalam VPC Anda melalui lalu lintas keluar. Secara default, EMR Tanpa Server memblokir akses masuk ke pekerja. Ini untuk meningkatkan keamanan.

Saat Anda menggunakan AWS Config, EMR Serverless membuat catatan item elastic network interface untuk setiap pekerja. Untuk menghindari biaya yang terkait dengan sumber daya ini, pertimbangkan untuk mematikan `AWS::EC2::NetworkInterface` AWS Config.

Note

Kami menyarankan Anda memilih beberapa subnet di beberapa Availability Zone. Ini karena subnet yang Anda pilih menentukan Availability Zones yang tersedia untuk aplikasi EMR Tanpa Server untuk diluncurkan. Setiap pekerja menggunakan alamat IP pada subnet tempat ia diluncurkan. Pastikan bahwa subnet yang ditentukan memiliki alamat IP yang cukup untuk jumlah pekerja yang Anda rencanakan untuk diluncurkan. Untuk informasi lebih lanjut tentang perencanaan subnet, lihat. [the section called “Praktik terbaik untuk perencanaan subnet”](#)

Pertimbangan dan batasan untuk subnet

- EMR Tanpa Server dengan subnet publik tidak mendukung Lake Formation. AWS
- Lalu lintas masuk tidak didukung untuk subnet publik.

Grup keamanan

Pilih satu atau beberapa grup keamanan yang dapat berkomunikasi dengan penyimpanan data Anda. Halaman Buat aplikasi mencantumkan semua grup keamanan di VPC Anda. EMR Tanpa Server mengaitkan grup keamanan ini dengan antarmuka jaringan elastis yang terpasang ke subnet VPC Anda.

Note

Kami menyarankan Anda membuat grup keamanan terpisah untuk aplikasi EMR Tanpa Server. EMR Tanpa Server tidak memungkinkan Anda untuk Membuat /Perbaruan/ Mulai aplikasi jika grup keamanan memiliki port yang terbuka ke internet publik pada kisaran 0.0.0.0/0 atau :/0. Ini memberikan peningkatan keamanan, isolasi, dan membuat pengelolaan aturan jaringan lebih efisien. Misalnya, ini memblokir lalu lintas tak terduga ke pekerja dengan alamat IP publik. Untuk berkomunikasi dengan cluster Amazon Redshift, misalnya, tentukan aturan lalu lintas antara grup keamanan Redshift dan EMR Tanpa Server, seperti yang ditunjukkan pada contoh di bagian berikut.

Example Contoh - Komunikasi dengan cluster Amazon Redshift

1. Tambahkan aturan untuk lalu lintas masuk ke grup keamanan Amazon Redshift dari salah satu grup keamanan EMR Tanpa Server.

Tipe	Protokol	Rentang port	Sumber
Semua TCP	TCP	5439	emr-serve rless-sec urity-group

2. Tambahkan aturan untuk lalu lintas keluar dari salah satu grup keamanan EMR Tanpa Server. Lakukan ini dengan salah satu dari dua cara. Pertama, buka lalu lintas keluar ke semua port.

Tipe	Protokol	Rentang Port	Tujuan
Semua Lalu lintas	TCP	SEMUA	0.0.0.0/0

Atau, Anda dapat membatasi lalu lintas keluar ke cluster Amazon Redshift. Ini berguna hanya ketika aplikasi harus berkomunikasi dengan cluster Amazon Redshift dan tidak ada yang lain.

Tipe	Protokol	Rentang port	Sumber
Semua TCP	TCP	5439	redshift-security-group

Konfigurasi aplikasi

Anda dapat mengubah konfigurasi jaringan untuk aplikasi EMR Tanpa Server yang ada dari halaman Konfigurasi aplikasi.

Akses detail pekerjaan

Pada halaman detail Job run, akses subnet yang digunakan oleh pekerjaan Anda untuk menjalankan tertentu. Perhatikan bahwa pekerjaan hanya berjalan di satu subnet yang dipilih dari subnet yang ditentukan.

Praktik terbaik untuk perencanaan subnet

AWS sumber daya dibuat dalam subnet yang merupakan bagian dari alamat IP yang tersedia di Amazon VPC. Misalnya, VPC dengan netmask /16 memiliki hingga 65.536 alamat IP yang tersedia yang dapat dipecah menjadi beberapa jaringan yang lebih kecil menggunakan subnet mask. Sebagai contoh, Anda dapat membagi rentang ini menjadi dua subnet dengan masing-masing menggunakan /17 mask dan 32.768 alamat IP yang tersedia. Subnet berada dalam Availability Zone dan tidak dapat menjangkau seluruh zona.

Subnet harus dirancang dengan mengingat batas penskalaan aplikasi EMR Tanpa Server Anda. Misalnya, jika Anda memiliki aplikasi yang meminta 4 pekerja vCPU dan dapat menskalakan hingga 4.000 vCPU, maka aplikasi Anda membutuhkan paling banyak 1.000 pekerja untuk total 1.000

antarmuka jaringan. Kami menyarankan Anda membuat subnet di beberapa Availability Zone. Hal ini memungkinkan EMR Tanpa Server untuk mencoba kembali pekerjaan Anda atau menyediakan kapasitas pra-inisialisasi di Availability Zone yang berbeda dalam kejadian yang tidak mungkin terjadi ketika Availability Zone gagal. Oleh karena itu, setiap subnet di setidaknya dua Availability Zone harus memiliki lebih dari 1.000 alamat IP yang tersedia.

Anda memerlukan subnet dengan ukuran topeng lebih rendah dari atau sama dengan 22 untuk menyediakan 1.000 antarmuka jaringan. Masker apa pun yang lebih besar dari 22 tidak memenuhi persyaratan. Misalnya, subnet mask dari /23 menyediakan 512 alamat IP, sedangkan mask /22 menyediakan 1024 dan mask /21 menyediakan 2048 alamat IP. Di bawah ini adalah contoh 4 subnet dengan /22 mask di VPC /16 netmask yang dapat dialokasikan ke Availability Zones yang berbeda. Ada perbedaan lima antara alamat IP yang tersedia dan yang dapat digunakan karena empat alamat IP pertama dan alamat IP terakhir di setiap subnet dicadangkan oleh AWS.

ID Subnet	Alamat Subnet	Topeng Subnet	Rentang Alamat IP	Alamat IP yang tersedia	Alamat IP yang Dapat Digunakan
1	10.0.0.0	255.255.252.0/22	10.0.0.0 - 10.0.3.255	1,024	1,019
2	10.0.4.0	255.255.252.0/22	10.0.4.0 - 10.0.7.255	1,024	1,019
3	10.0.8.0	255.255.252.0/22	10.0.8.0 - 10.0.11.255	1,024	1,019
4	10.0.12.0	255.255.252.0/22	10.0.12.0 - 10.0.15.255	1,024	1,019

Anda harus mengevaluasi apakah beban kerja Anda paling cocok untuk ukuran pekerja yang lebih besar. Menggunakan ukuran pekerja yang lebih besar membutuhkan antarmuka jaringan yang lebih sedikit. Misalnya, menggunakan pekerja 16vCPU dengan batas penskalaan aplikasi 4.000 vCPU membutuhkan paling banyak 250 pekerja dengan total 250 alamat IP yang tersedia untuk menyediakan antarmuka jaringan. Anda memerlukan subnet di beberapa Availability Zone dengan ukuran mask lebih rendah dari atau sama dengan 24 untuk menyediakan 250 antarmuka jaringan. Setiap ukuran topeng yang lebih besar dari 24 menawarkan kurang dari 250 alamat IP.

Jika Anda berbagi subnet di beberapa aplikasi, setiap subnet harus dirancang dengan mengingat batas penskalaan kolektif dari semua aplikasi Anda. Misalnya, jika Anda memiliki 3 aplikasi yang meminta 4 pekerja vCPU dan masing-masing dapat meningkatkan hingga 4000 vCPU dengan 12.000 kuota berbasis layanan tingkat akun vCPU, setiap subnet memerlukan 3000 alamat IP yang tersedia. Jika VPC yang ingin Anda gunakan tidak memiliki jumlah alamat IP yang cukup, cobalah untuk menambah jumlah alamat IP yang tersedia. Anda dapat melakukan ini dengan mengaitkan blok Classless Inter-Domain Routing (CIDR) tambahan dengan VPC Anda. Untuk informasi selengkapnya, lihat [Kaitkan blok IPv4 CIDR tambahan dengan VPC Anda](#) di Panduan Pengguna Amazon VPC.

Anda dapat menggunakan salah satu dari banyak alat yang tersedia secara online untuk menghasilkan definisi subnet dengan cepat dan meninjau berbagai alamat IP yang tersedia.

Amazon EMR Opsi arsitektur tanpa server

Arsitektur set instruksi aplikasi Amazon EMR Serverless menentukan jenis prosesor yang digunakan aplikasi untuk menjalankan pekerjaan. Amazon EMR menyediakan dua opsi arsitektur untuk aplikasi Anda: x86_64 dan arm64. EMR Tanpa Server secara otomatis memperbarui ke generasi terbaru instans saat tersedia, sehingga aplikasi Anda dapat menggunakan instance yang lebih baru tanpa memerlukan upaya tambahan dari Anda.

Topik

- [Menggunakan arsitektur x86_64](#)
- [Menggunakan arsitektur arm64 \(Graviton\)](#)
- [Meluncurkan aplikasi baru dengan dukungan Graviton](#)
- [Mengkonfigurasi aplikasi yang ada untuk menggunakan Graviton](#)
- [Pertimbangan saat menggunakan Graviton](#)

Menggunakan arsitektur x86_64

Arsitektur x86_64 juga dikenal sebagai x86 64-bit atau x64. x86_64 adalah opsi default untuk aplikasi EMR Tanpa Server. Arsitektur ini menggunakan prosesor berbasis x86 dan kompatibel dengan sebagian besar alat dan perpustakaan pihak ketiga.

Sebagian besar aplikasi kompatibel dengan platform perangkat keras x86 dan dapat berjalan dengan sukses pada arsitektur x86_64 default. Namun, jika aplikasi Anda kompatibel dengan ARM 64-bit, maka beralihlah ke arm64 untuk menggunakan prosesor Graviton untuk meningkatkan kinerja, daya

komputasi, dan memori. Biayanya lebih murah untuk menjalankan instance pada arsitektur arm64 daripada saat Anda menjalankan instance dengan ukuran yang sama pada arsitektur x86.

Menggunakan arsitektur arm64 (Graviton)

AWS Prosesor Graviton dirancang khusus AWS dengan inti ARM Neoverse 64-bit dan memanfaatkan arsitektur arm64 (juga dikenal sebagai Arch64 atau ARM 64-bit). Jajaran prosesor AWS Graviton yang tersedia di EMR Serverless termasuk prosesor Graviton3 dan Graviton2. Prosesor ini memberikan kinerja harga yang unggul untuk beban kerja Spark dan Hive dibandingkan dengan beban kerja setara yang berjalan pada arsitektur x86_64. EMR Tanpa Server secara otomatis menggunakan prosesor generasi terbaru bila tersedia tanpa usaha dari pihak Anda untuk meningkatkan ke prosesor generasi terbaru.

Meluncurkan aplikasi baru dengan dukungan Graviton

Gunakan salah satu metode berikut untuk meluncurkan aplikasi yang menggunakan arsitektur arm64.

AWS CLI

Untuk meluncurkan aplikasi menggunakan prosesor Graviton dari AWS CLI, tentukan ARM64 sebagai `architecture` parameter di `API. create-application`. Berikan nilai yang sesuai untuk aplikasi Anda di parameter lain.

```
aws emr-serverless create-application \  
  --name my-graviton-app \  
  --release-label emr-6.8.0 \  
  --type "SPARK" \  
  --architecture "ARM64" \  
  --region us-west-2
```

EMR Studio

Untuk meluncurkan aplikasi menggunakan prosesor Graviton dari EMR Studio, pilih arm64 sebagai opsi Arsitektur saat Anda membuat atau memperbarui aplikasi.

Mengkonfigurasi aplikasi yang ada untuk menggunakan Graviton

Anda dapat mengonfigurasi aplikasi Amazon EMR Tanpa Server yang ada untuk menggunakan arsitektur Graviton (arm64) dengan SDK,, atau EMR Studio. AWS CLI

Untuk mengonversi aplikasi yang ada dari x86 ke arm64

1. Konfirmasikan bahwa Anda menggunakan versi mayor terbaru dari [AWS CLI/SDK](#) yang mendukung parameter. `architecture`
2. Konfirmasikan bahwa tidak ada pekerjaan yang berjalan dan kemudian hentikan aplikasi.

```
aws emr-serverless stop-application \  
  --application-id application-id \  
  --region us-west-2
```

3. Untuk memperbarui aplikasi untuk menggunakan Graviton, ARM64 tentukan `architecture` parameter di API. `update-application`

```
aws emr-serverless update-application \  
  --application-id application-id \  
  --architecture 'ARM64' \  
  --region us-west-2
```

4. Untuk memverifikasi bahwa arsitektur CPU aplikasi sekarang ARM64, gunakan `get-application` API.

```
aws emr-serverless get-application \  
  --application-id application-id \  
  --region us-west-2
```

5. Saat Anda siap, mulai ulang aplikasi.

```
aws emr-serverless start-application \  
  --application-id application-id \  
  --region us-west-2
```

Pertimbangan saat menggunakan Graviton

Sebelum Anda meluncurkan aplikasi EMR Tanpa Server menggunakan arm64 untuk dukungan Graviton, konfirmasikan hal berikut.

Kompatibilitas perpustakaan

Saat Anda memilih Graviton (arm64) sebagai opsi arsitektur, pastikan paket dan pustaka pihak ketiga kompatibel dengan arsitektur ARM 64-bit. Untuk informasi tentang cara mengemas pustaka

Python ke dalam lingkungan virtual Python yang kompatibel dengan arsitektur pilihan Anda, lihat.

[Menggunakan pustaka Python dengan EMR Tanpa Server](#)

Untuk mempelajari lebih lanjut, lihat repositori [AWS Graviton Getting Started](#) di GitHub Repositori ini berisi sumber daya penting yang dapat membantu Anda memulai Graviton berbasis ARM.

Konkurensi pekerjaan dan antrian untuk aplikasi EMR Tanpa Server

Dimulai dengan Amazon EMR versi 7.0.0 dan yang lebih baru, tentukan batas waktu antrian tugas dan konfigurasi konkurensi untuk aplikasi Anda. Saat Anda menentukan konfigurasi ini, Amazon EMR Tanpa Server dimulai dengan mengantri pekerjaan Anda dan memulai eksekusi berdasarkan pemanfaatan konkurensi pada aplikasi Anda. Misalnya, jika konkurensi pekerjaan Anda adalah 10, hanya sepuluh pekerjaan yang dijalankan sekaligus pada aplikasi Anda. Pekerjaan yang tersisa diantrian sampai salah satu pekerjaan yang berjalan berakhir. Jika batas waktu antrian tercapai lebih awal, waktu kerja Anda habis. Untuk informasi selengkapnya, lihat [status Job run](#).

Manfaat utama konkurensi dan antrian

Konkurensi dan antrian pekerjaan memberikan manfaat berikut ketika banyak pengajuan pekerjaan diperlukan:

- Ini membantu mengontrol pekerjaan eksekusi bersamaan untuk secara efisien menggunakan batas kapasitas tingkat aplikasi Anda.
- Antrian dapat berisi ledakan pengiriman pekerjaan yang tiba-tiba, dengan pengaturan batas waktu yang dapat dikonfigurasi.

Memulai dengan konkurensi dan antrian

Prosedur berikut menunjukkan beberapa cara berbeda untuk menerapkan konkurensi dan antrian.

Menggunakan AWS CLI

1. Buat aplikasi Amazon EMR Tanpa Server dengan batas waktu antrian dan menjalankan pekerjaan bersamaan:

```
aws emr-serverless create-application \  
--release-label emr-7.0.0 \  

```

```
--type SPARK \  
--scheduler-configuration '{"maxConcurrentRuns": 1, "queueTimeoutMinutes": 30}'
```

2. Perbarui aplikasi untuk mengubah batas waktu antrian pekerjaan dan konkurensi:

```
aws emr-serverless update-application \  
--application-id application-id \  
--scheduler-configuration '{"maxConcurrentRuns": 5, "queueTimeoutMinutes": 30}'
```

Note

Anda dapat memperbarui aplikasi yang ada untuk mengaktifkan konkurensi pekerjaan dan antrian. Untuk melakukan ini, aplikasi harus memiliki label rilis emr-7.0.0 atau yang lebih baru.

Menggunakan Konsol Manajemen AWS

Langkah-langkah berikut menunjukkan cara memulai dengan konkurensi pekerjaan dan antrian, menggunakan: Konsol Manajemen AWS

1. Buka EMR Studio dan pilih untuk membuat aplikasi dengan label rilis EMR-7.0.0 atau lebih tinggi.
2. Di bawah Opsi pengaturan aplikasi, pilih opsi Gunakan pengaturan khusus.
3. Di bawah Konfigurasi tambahan ada bagian untuk Pengaturan Job Run. Pilih opsi Aktifkan konkurensi pekerjaan untuk mengaktifkan fitur.
4. Setelah pemilihan, pilih Pekerjaan bersamaan berjalan dan Batas waktu antrian untuk mengonfigurasi jumlah pekerjaan bersamaan dan batas waktu antrian, masing-masing. Jika Anda tidak memasukkan nilai untuk pengaturan ini, nilai default digunakan.
5. Pilih Buat Aplikasi dan aplikasi akan dibuat dengan fitur ini diaktifkan. Untuk memverifikasi, buka dasbor, pilih aplikasi Anda dan periksa di bawah tab properti untuk menentukan apakah fitur tersebut diaktifkan.

Mengikuti konfigurasi, kirimkan pekerjaan dengan fitur ini diaktifkan.

Pertimbangan untuk konkurensi dan antrian

Pertimbangkan hal-hal berikut saat Anda menerapkan konkurensi dan antrian:

- Konkurensi dan antrian Job didukung pada Amazon EMR rilis 7.0.0 dan yang lebih tinggi.
- Konkurensi dan antrian Job diaktifkan secara default di Amazon EMR rilis 7.3.0 dan yang lebih tinggi.
- Anda tidak dapat memperbarui konkurensi untuk aplikasi dalam status STARTED.
- Rentang yang valid untuk `maxConcurrentRuns` adalah 1 hingga 1000, dan untuk `queueTimeoutMinutes` itu adalah 15 hingga 720.
- Maksimal 2000 pekerjaan dapat berada dalam keadaan ANTRIAN untuk sebuah akun.
- Konkurensi dan antrian berlaku untuk pekerjaan batch dan streaming. Ini tidak dapat digunakan untuk pekerjaan interaktif. Untuk informasi selengkapnya, lihat [Jalankan beban kerja interaktif dengan EMR Tanpa Server melalui EMR Studio](#).

Dapatkan data ke S3 Express One Zone dengan EMR Tanpa Server

Dengan Amazon EMR merilis 7.2.0 dan yang lebih tinggi, gunakan EMR Tanpa Server dengan kelas penyimpanan [Amazon S3 Express](#) One Zone untuk meningkatkan kinerja saat Anda 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

- Izin S3 Express One Zone — Ketika S3 Express One Zone awalnya melakukan tindakan seperti GET, LIST, atau PUT pada objek S3, kelas penyimpanan memanggil `CreateSession` atas nama Anda. Kebijakan IAM Anda harus mengizinkan `s3express:CreateSession` izin agar S3A konektor dapat menjalankan API. `CreateSession` Untuk contoh kebijakan dengan izin ini, lihat [Memulai dengan S3 Express One Zone](#).
- S3A konektor — Untuk mengonfigurasi Spark untuk mengakses data dari bucket Amazon S3 yang menggunakan kelas penyimpanan S3 Express One Zone, gunakan konektor Apache Hadoop. S3A Untuk menggunakan konektor, pastikan semua S3 URIs menggunakan `s3a` skema. Jika tidak, ubah 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 `s3n` skema, 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"
    }
  }
]
```

Memulai dengan S3 Express One Zone

Ikuti langkah-langkah ini untuk memulai dengan S3 Express One Zone.

1. [Buat titik akhir VPC](#). Tambahkan titik akhir `com.amazonaws.us-west-2.s3express` ke titik akhir VPC.
2. Ikuti [Memulai Amazon EMR Tanpa Server untuk membuat aplikasi dengan](#) label rilis Amazon EMR 7.2.0 atau lebih tinggi.
3. [Konfigurasi aplikasi Anda](#) untuk menggunakan titik akhir VPC yang baru dibuat, grup subnet pribadi, dan grup keamanan.
4. Tambahkan `CreateSession` izin ke peran eksekusi pekerjaan Anda.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Resource": [
        "*"
      ],
      "Action": [
        "s3express:CreateSession"
      ],
      "Sid": "AllowS3EXPRESSCreatesession"
    }
  ]
}
```

5. Jalankan pekerjaan Anda. Perhatikan bahwa gunakan S3A skema untuk mengakses bucket S3 Express One Zone.

```
aws emr-serverless start-job-run \  
--application-id <application-id> \  
--execution-role-arn <job-role-arn> \  
--name <job-run-name> \  
--job-driver '{  
  "sparkSubmit": {  
  
    "entryPoint": "s3a://<DOC-EXAMPLE-BUCKET>/scripts/wordcount.py",  
    "entryPointArguments":["s3a://<DOC-EXAMPLE-BUCKET>/emr-serverless-spark/output"],  
    "sparkSubmitParameters": "--conf spark.executor.cores=4  
--conf spark.executor.memory=8g --conf spark.driver.cores=4  
--conf spark.driver.memory=8g --conf spark.executor.instances=2  
--conf spark.hadoop.fs.s3a.change.detection.mode=none  
--conf spark.hadoop.fs.s3a.endpoint.region={<AWS_REGION>}  
--conf spark.hadoop.fs.s3a.select.enabled=false  
--conf spark.sql.sources.fastS3PartitionDiscovery.enabled=false  
  }'  
'
```

Menjalankan pekerjaan

Setelah Anda menyediakan aplikasi Anda, kirimkan pekerjaan ke aplikasi. Bagian ini mencakup cara menggunakan AWS CLI untuk menjalankan pekerjaan ini. Bagian ini juga mengidentifikasi nilai default untuk setiap jenis aplikasi yang tersedia di EMR Tanpa Server.

Topik

- [Status tugas berjalan](#)
- [Pembatalan pekerjaan EMR Tanpa Server dengan masa tenggang](#)
- [Menjalankan pekerjaan dari konsol EMR Studio](#)
- [Menjalankan pekerjaan dari AWS CLI](#)
- [Eksekusi kebijakan IAM](#)
- [Menggunakan disk yang dioptimalkan dengan shuffle](#)
- [Menggunakan penyimpanan tanpa server untuk Amazon EMR Tanpa Server](#)
- [Pekerjaan streaming untuk memproses data yang dialirkan secara terus menerus](#)
- [Menggunakan konfigurasi Spark saat Anda menjalankan pekerjaan EMR Tanpa Server](#)
- [Menggunakan konfigurasi Hive saat Anda menjalankan pekerjaan EMR Tanpa Server](#)
- [Ketahanan Pekerjaan EMR Tanpa Server](#)
- [Konfigurasi metastore untuk EMR Tanpa Server](#)
- [Mengakses data S3 di AWS akun lain dari EMR Tanpa Server](#)
- [Memecahkan masalah kesalahan di EMR Tanpa Server](#)
- [Mengaktifkan Alokasi Biaya Level Pekerjaan](#)

Status tugas berjalan

Saat Anda mengirimkan pekerjaan ke antrean pekerjaan Amazon EMR Tanpa Server, pekerjaan berjalan memasuki status. SUBMITTED Suatu keadaan pekerjaan berlalu dari SUBMITTED melalui RUNNING sampai mencapai FAILED, SUCCESS, atau CANCELLING.

Tugas berjalan dapat memiliki status berikut:

Status	Deskripsi
Dikirim	Status pekerjaan awal saat Anda mengirimkan pekerjaan dijalankan ke EMR Tanpa Server. Pekerjaan menunggu untuk dijadwalkan untuk aplikasi. EMR Tanpa Server mulai memprioritaskan dan menjadwalkan pekerjaan berjalan.
Antre	Job run menunggu dalam keadaan ini ketika tingkat aplikasi job run concurrency terisi penuh. Untuk informasi lebih lanjut tentang antrian dan konkurensi, lihat. Konkurensi pekerjaan dan antrian untuk aplikasi EMR Tanpa Server
Tertunda	Penjadwal sedang mengevaluasi pekerjaan yang dijalankan untuk memprioritaskan dan menjadwalkan proses untuk aplikasi.
Dijadwalkan	EMR Serverless telah menjadwalkan pekerjaan yang dijalankan untuk aplikasi, dan mengalokasikan sumber daya untuk melaksanakan pekerjaan itu.
Berlari	EMR Tanpa Server telah mengalokasikan sumber daya yang awalnya dibutuhkan pekerjaan, dan pekerjaan berjalan dalam aplikasi. Dalam aplikasi Spark, ini berarti bahwa proses driver Spark ada di status <code>running</code> .
Gagal	EMR Tanpa Server gagal mengirimkan pekerjaan yang dijalankan ke aplikasi, atau tidak berhasil diselesaikan. Lihat <code>StateDetails</code> untuk informasi tambahan tentang kegagalan pekerjaan ini.
Sukses	Jalankan pekerjaan telah selesai dengan sukses.

Status	Deskripsi
Membatalkan	<code>CancelJobRun</code> API telah meminta pembatalan job run, atau job run telah habis waktunya. EMR Tanpa Server mencoba membatalkan pekerjaan dalam aplikasi dan melepaskan sumber daya.
Dibatalkan	Pekerjaan dibatalkan dengan sukses, dan sumber daya yang digunakannya telah dirilis.

Pembatalan pekerjaan EMR Tanpa Server dengan masa tenggang

Dalam sistem pemrosesan data, penghentian mendadak dapat menyebabkan pemborosan sumber daya, operasi yang tidak lengkap, dan potensi inkonsistensi data. Amazon EMR Tanpa Server memungkinkan Anda menentukan masa tenggang saat membatalkan pekerjaan berjalan. Fitur ini memungkinkan waktu untuk pembersihan yang tepat dan penyelesaian pekerjaan yang sedang berlangsung sebelum pemutusan hubungan kerja.

Note

Fitur ini didukung dengan Amazon EMR rilis 7.9.0 dan lebih tinggi.

Saat membatalkan pekerjaan, tentukan masa tenggang (dalam detik) menggunakan parameter di `shutdownGracePeriodInSeconds` mana pekerjaan dapat melakukan operasi pembersihan sebelum pemutusan akhir. Perilaku dan pengaturan default bervariasi antara pekerjaan batch dan streaming.

Masa Tenggang Untuk pekerjaan batch

Untuk pekerjaan batch, EMR Tanpa Server memungkinkan Anda menerapkan operasi pembersihan khusus yang dijalankan selama masa tenggang. Anda dapat mendaftarkan operasi pembersihan ini sebagai bagian dari hook shutdown JVM dalam kode aplikasi Anda.

Perilaku default

Perilaku default untuk shutdown adalah tidak memiliki masa tenggang. Ini terdiri dari dua tindakan berikut:

- Pengakhiran segera
- Sumber daya segera dirilis

Opsi konfigurasi

Anda dapat menentukan pengaturan yang menghasilkan shutdown yang anggun:

- Rentang yang Valid untuk Masa tenggang Shutdown: 15-1800 detik (opsional)
- Pengakhiran segera (tanpa masa tenggang): 0 detik

Aktifkan shutdown yang anggun

Untuk menerapkan shutdown yang anggun untuk pekerjaan batch, ikuti langkah-langkah berikut:

1. Tambahkan shutdown hook dalam kode aplikasi Anda yang berisi logika shutdown kustom.

Example in Scala

```
import org.apache.hadoop.util.ShutdownHookManager

// Register shutdown hook with priority (second argument)
// Higher priority hooks run first
ShutdownHookManager.get().addShutdownHook(() => {
    logger.info("Performing cleanup operations...")
}, 100)
```

Menggunakan [ShutdownHookManager](#)

Example in PySpark

```
import atexit

def cleanup():
    # Your cleanup logic here
    print("Performing cleanup operations...")

# Register the cleanup function
atexit.register(cleanup)
```

2. Tentukan masa tenggang saat membatalkan pekerjaan untuk memberikan waktu bagi kait yang ditambahkan sebelumnya untuk dieksekusi

Contoh

```
# Default (immediate termination)
aws emr-serverless cancel-job-run \
  --application-id APPLICATION_ID \
  --job-run-id JOB_RUN_ID

# With 5-minute grace period
aws emr-serverless cancel-job-run \
  --application-id APPLICATION_ID \
  --job-run-id JOB_RUN_ID \
  --shutdown-grace-period-in-seconds 300
```

Masa Tenggang Untuk Pekerjaan Streaming

Dalam Spark Structured Streaming, di mana perhitungan melibatkan membaca dari atau menulis ke sumber data eksternal, shutdown tiba-tiba dapat menyebabkan hasil yang tidak diinginkan. Streaming pekerjaan memproses data dalam batch mikro, dan mengganggu operasi ini di tengah jalan dapat menghasilkan pemrosesan duplikat dalam upaya berikutnya. Ini terjadi ketika pos pemeriksaan terbaru dari batch mikro sebelumnya tidak ditulis, menyebabkan data yang sama diproses lagi ketika pekerjaan streaming dimulai ulang. Pemrosesan duplikat semacam itu tidak hanya menyia-nyaiakan sumber daya komputasi tetapi juga dapat memengaruhi operasi bisnis, sehingga penting untuk menghindari penutupan mendadak.

EMR Tanpa Server menyediakan dukungan bawaan untuk shutdown yang anggun melalui pendengar kueri streaming. Ini memastikan penyelesaian yang tepat dari batch mikro yang sedang berlangsung sebelum pemutusan hubungan kerja. Layanan ini secara otomatis mengelola shutdown yang anggun antara batch mikro untuk aplikasi streaming, memastikan bahwa batch mikro saat ini menyelesaikan pemrosesan, pos pemeriksaan ditulis dengan benar, dan konteks streaming dihentikan dengan bersih tanpa menelan data baru selama proses shutdown.

Perilaku default

- Masa tenggang 120 detik diaktifkan secara default.
- Pendengar kueri streaming bawaan mengelola shutdown yang anggun.

Opsi konfigurasi

- Rentang yang Valid untuk Masa tenggang Shutdown: 15-1800 detik (opsional)
- Pengakhiran Segera: 0 detik

Aktifkan Shutdown Anggun

Untuk menerapkan shutdown yang anggun untuk pekerjaan streaming:

Tentukan masa tenggang saat membatalkan pekerjaan untuk memberikan waktu bagi batch mikro yang sedang berlangsung untuk diselesaikan.

Contoh

```
# Default graceful shutdown (120 seconds)
aws emr-serverless cancel-job-run \
  --application-id APPLICATION_ID \
  --job-run-id JOB_RUN_ID

# Custom grace period (e.g. 300 seconds)
aws emr-serverless cancel-job-run \
  --application-id APPLICATION_ID \
  --job-run-id JOB_RUN_ID \
  --shutdown-grace-period-in-seconds 300

# Immediate Termination
aws emr-serverless cancel-job-run \
  --application-id APPLICATION_ID \
  --job-run-id JOB_RUN_ID \
  --shutdown-grace-period-in-seconds 0
```

Tambahkan kait shutdown kustom (opsional)

Sementara EMR Serverless mengelola shutdown yang anggun secara default melalui pendengar kueri streaming bawaannya, Anda dapat secara opsional menerapkan logika shutdown khusus untuk kueri streaming individual. EMR Tanpa Server mendaftarkan pendengar shutdown yang anggun dengan prioritas 60 (mengggunakan). ShutdownHookManager Karena kait prioritas yang lebih tinggi dijalankan terlebih dahulu, Anda dapat mendaftarkan operasi pembersihan khusus Anda dengan prioritas lebih besar dari 60 untuk memastikannya dijalankan sebelum proses shutdown EMR Serverless dimulai.

Untuk menambahkan hook khusus, lihat contoh pertama dalam topik ini yang menunjukkan cara menambahkan hook shutdown dalam kode aplikasi Anda. Di sini, 100 adalah prioritas, yang lebih besar dari 60. Oleh karena itu kait shutdown seperti itu berjalan lebih dulu.

Note

Kait shutdown khusus bersifat opsional dan tidak diperlukan untuk fungsionalitas shutdown yang anggun, yang ditangani secara otomatis oleh EMR Tanpa Server.

Biaya Masa Tenggang dan Durasi Batch

Jika nilai default untuk masa tenggang (120 detik) digunakan:

- Jika durasi batch Anda kurang dari 120 detik, Anda hanya akan dikenakan biaya untuk waktu aktual yang diperlukan untuk menyelesaikan batch.
- Jika durasi batch Anda melebihi 120 detik, Anda akan dikenakan biaya untuk masa tenggang maksimum (120 detik), tetapi kueri mungkin tidak dimatikan dengan baik karena akan dihentikan secara paksa.

Untuk mengoptimalkan biaya dan memastikan shutdown yang anggun:

- Untuk durasi batch > 120 detik: Pertimbangkan untuk meningkatkan masa tenggang agar sesuai dengan durasi batch Anda
- Untuk durasi batch <120 detik: Tidak perlu menyesuaikan masa tenggang karena Anda hanya akan dikenakan biaya untuk waktu pemrosesan yang sebenarnya

Pertimbangan-pertimbangan

Perilaku Masa Tenggang

- Masa tenggang menyediakan waktu agar kait shutdown terdaftar Anda selesai.
- Job berakhir segera setelah shutdown hook selesai bahkan jika itu jauh sebelum masa tenggang.
- Jika operasi pembersihan melebihi masa tenggang, pekerjaan akan dihentikan secara paksa.

Perilaku Layanan

- Penutupan masa tenggang hanya tersedia untuk pekerjaan dalam status RUNNING.
- Permintaan pembatalan berikutnya selama status CANCELLING diabaikan.
- Jika EMR Tanpa Server gagal memulai penutupan masa tenggang karena kesalahan layanan internal:
 - Layanan akan mencoba lagi hingga 2 menit.
 - Jika percobaan ulang tidak berhasil, pekerjaan akan dihentikan secara paksa.

Penagihan

Pekerjaan ditagih untuk sumber daya komputasi yang digunakan sampai pekerjaan benar-benar ditutup, termasuk waktu yang diambil selama masa tenggang.

Menjalankan pekerjaan dari konsol EMR Studio

Anda dapat mengirimkan pekerjaan ke aplikasi EMR Tanpa Server dan mengakses pekerjaan dari konsol EMR Studio. [Untuk membuat atau menavigasi ke aplikasi EMR Tanpa Server di konsol EMR Studio, ikuti petunjuk di Memulai dari konsol.](#)


Mengirim tugas

Pada halaman Kirim pekerjaan, kirimkan pekerjaan ke aplikasi EMR Tanpa Server sebagai berikut.

Spark

1. Di bidang Nama, masukkan nama untuk menjalankan pekerjaan Anda.
2. Di bidang peran Runtime, masukkan nama peran IAM yang dapat diasumsikan oleh aplikasi EMR Tanpa Server Anda untuk menjalankan pekerjaan. Untuk mempelajari lebih lanjut tentang peran runtime, lihat. [Peran runtime Job untuk Amazon EMR Tanpa Server](#)
3. Di bidang Lokasi skrip, masukkan lokasi Amazon S3 untuk skrip atau JAR yang ingin Anda jalankan. Untuk pekerjaan Spark, skrip dapat berupa file Python `.py` () atau file JAR `.jar` ().
4. Jika lokasi skrip Anda adalah file JAR, masukkan nama kelas yang merupakan titik masuk untuk pekerjaan di bidang kelas Utama.
5. (Opsional) Masukkan nilai untuk bidang yang tersisa.

- Argumen skrip - Masukkan argumen apa pun yang ingin Anda teruskan ke skrip JAR atau Python utama Anda. Kode Anda membaca parameter ini. Pisahkan setiap argumen dalam array dengan koma.
- Properti percikan - Perluas bagian properti Spark dan masukkan parameter konfigurasi Spark apa pun di bidang ini.

 Note

Jika Anda menentukan driver Spark dan ukuran pelaksana, pertimbangkan overhead memori. Tentukan nilai overhead memori di properti `spark.driver.memoryOverhead` dan `spark.executor.memoryOverhead`. Memory overhead memiliki nilai default 10% dari memori kontainer, dengan minimal 384 MB. Memori eksekutor dan overhead memori bersama-sama tidak dapat melebihi memori pekerja. Misalnya, maksimum `spark.executor.memory` pada pekerja 30 GB harus 27 GB.

- Konfigurasi Job - Tentukan konfigurasi pekerjaan apa pun di bidang ini. Anda dapat menggunakan konfigurasi pekerjaan ini untuk mengganti konfigurasi default untuk aplikasi. Contoh berikut menunjukkan cara mengganti pengaturan default Spark seperti memori pelaksana dan driver.

```
{
  "applicationConfiguration": [
    {
      "classification": "spark-defaults",
      "configurations": [],
      "properties": {
        "spark.executor.memory": "8G",
        "spark.driver.memory": "6G",
        "spark.driver.cores": "2",
        "spark.executor.cores": "4"
      }
    }
  ]
}
```

- Pengaturan tambahan - Aktifkan atau nonaktifkan Katalog Data AWS Glue sebagai metastore dan ubah pengaturan log aplikasi. Untuk mempelajari lebih lanjut tentang

konfigurasi metastore, lihat. [Konfigurasi metastore untuk EMR Tanpa Server](#) Untuk mempelajari lebih lanjut tentang opsi pencatatan aplikasi, lihat [Menyimpan log](#).

- Tag - Tetapkan tag kustom ke aplikasi.

6. Pilih Submit job (Kirim tugas).

Hive

1. Di bidang Nama, masukkan nama untuk menjalankan pekerjaan Anda.
2. Di bidang peran Runtime, masukkan nama peran IAM yang dapat diasumsikan oleh aplikasi EMR Tanpa Server Anda untuk menjalankan pekerjaan.
3. Di bidang Lokasi skrip, masukkan lokasi Amazon S3 untuk skrip atau JAR yang ingin Anda jalankan. Untuk pekerjaan Hive, skrip harus berupa file Hive (.sql).
4. (Opsional) Masukkan nilai untuk bidang yang tersisa.
 - Lokasi skrip inialisasi - Masukkan lokasi skrip yang menginisialisasi tabel sebelum skrip Hive berjalan.
 - Properti sarang - Perluas bagian properti Hive dan masukkan parameter konfigurasi Hive apa pun di bidang ini.
 - Konfigurasi Job - Tentukan konfigurasi pekerjaan apa pun. Anda dapat menggunakan konfigurasi pekerjaan ini untuk mengganti konfigurasi default untuk aplikasi. Untuk pekerjaan Hive, `hive.exec.scratchdir` dan properti `hive.metastore.warehouse.dir` yang diperlukan dalam `hive-site` konfigurasi.

```
{
  "applicationConfiguration": [
    {
      "classification": "hive-site",
      "configurations": [],
      "properties": {
        "hive.exec.scratchdir": "s3://DOC-EXAMPLE_BUCKET/hive/scratch",
        "hive.metastore.warehouse.dir": "s3://DOC-EXAMPLE_BUCKET/hive/warehouse"
      }
    }
  ],
  "monitoringConfiguration": {}
}
```



```

--execution-timeout-minutes 15 \
--job-driver '{
  "hive": {
    "query": "s3://amzn-s3-demo-bucket/scripts/create_table.sql",
    "parameters": "--hiveconf hive.exec.scratchdir=s3://amzn-s3-demo-bucket/
hive/scratch --hiveconf hive.metastore.warehouse.dir=s3://amzn-s3-demo-bucket/hive/
warehouse"
  }
}' \
--configuration-overrides '{
  "applicationConfiguration": [{
    "classification": "hive-site",
    "properties": {
      "hive.client.cores": "2",
      "hive.client.memory": "4GIB"
    }
  ]
}'

```

Untuk menggambarkan pekerjaan, gunakan `get-job-run`. Perintah ini mengembalikan konfigurasi khusus pekerjaan dan kapasitas yang ditetapkan untuk pekerjaan baru Anda.

```

aws emr-serverless get-job-run \
--job-run-id job-id \
--application-id application-id

```

Untuk daftar pekerjaan Anda, gunakan `list-job-runs`. Perintah ini mengembalikan serangkaian properti yang disingkat yang mencakup tipe pekerjaan, status, dan atribut tingkat tinggi lainnya. Jika Anda tidak ingin mengakses semua pekerjaan Anda, tentukan jumlah maksimum pekerjaan yang ingin Anda akses, hingga 50. Contoh berikut menentukan bahwa Anda ingin mengakses dua pekerjaan terakhir Anda berjalan.

```

aws emr-serverless list-job-runs \
--max-results 2 \
--application-id application-id

```

Untuk membatalkan pekerjaan, gunakan `cancel-job-run`. Berikan `application-id` dan pekerjaan `job-id` yang ingin Anda batalkan.

```

aws emr-serverless cancel-job-run \
--job-run-id job-id \

```

```
--application-id application-id
```

Untuk informasi selengkapnya tentang cara menjalankan pekerjaan dari AWS CLI, lihat [Referensi API EMR Tanpa Server](#).

Eksekusi kebijakan IAM

Anda dapat menentukan Kebijakan IAM Eksekusi, selain Peran Eksekusi, saat mengirimkan pekerjaan berjalan di EMR Tanpa Server. Izin yang dihasilkan yang diasumsikan oleh job run adalah persimpangan izin dalam Peran Eksekusi dan Kebijakan IAM Eksekusi yang ditentukan.

Memulai

Langkah-langkah untuk menggunakan kebijakan Eksekusi IAM:

Buat `emr-serverless` aplikasi atau gunakan yang sudah ada dan kemudian jalankan `aws cli` berikut untuk memulai pekerjaan dengan kebijakan IAM sebaris:

```
aws emr-serverless start-job-run --region us-west-2 \  
  --application-id application-id \  
  --execution-role-arn execution-role-arn \  
  --job-driver job-driver-options \  
  --execution-iam-policy '{"policy": "inline-policy"}'
```

Contoh perintah CLI

Jika kami memiliki kebijakan berikut yang disimpan dalam `policy.json` file di mesin:

JSON

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "s3:GetObject",  
        "s3:ListBucket"  
      ],  
      "Resource": [  
        "arn:aws:s3:::my-test-bucket",
```

```

        "arn:aws:s3:::my-test-bucket/*"
    ],
    "Sid": "AllowS3GetObject"
  }
]
}

```

Kemudian kita dapat memulai pekerjaan dengan kebijakan ini menggunakan AWS CLI perintah berikut:

```

aws emr-serverless start-job-run --region us-west-2 \
  --application-id application-id \
  --execution-role-arn execution-role-arn \
  --job-driver job-driver-options
  --execution-iam-policy '{
    "policy": '$(jq -c '. | @json' policy.json)'
  }'

```

Anda juga dapat menggunakan keduanya AWS dan kebijakan yang dikelola Pelanggan, menentukannya ARNs melalui:

```

aws emr-serverless start-job-run --region us-west-2 \
  --application-id application-id \
  --execution-role-arn execution-role-arn \
  --job-driver job-driver-options
  --execution-iam-policy '{
    "policyArns": [
      "arn:aws:iam::aws:policy/AmazonS3FullAccess",
      "arn:aws:iam::aws:policy/CloudWatchLogsFullAccess"
    ]
  }'

```

Dimungkinkan untuk menentukan kebijakan IAM sebaris dan kebijakan terkelola ARNs dalam permintaan yang sama juga:

```

aws emr-serverless start-job-run --region us-west-2 \
  --application-id application-id \
  --execution-role-arn execution-role-arn \
  --job-driver job-driver-options
  --execution-iam-policy '{
    "policy": '$(jq -c '. | @json' policy.json)',

```

```
"policyArns": [
  "arn:aws:iam::aws:policy/AmazonS3FullAccess",
  "arn:aws:iam::aws:policy/CloudWatchLogsFullAccess"
]
```

Catatan Penting

- `execution-role-policy` bidang ini dapat memiliki panjang maksimum 2048 karakter.
- String kebijakan IAM sebaris yang ditentukan dalam `policy` bidang `execution-iam-policy`'s harus sesuai dengan standar string json, tanpa baris baru dan tanda kutip yang lolos seperti pada contoh sebelumnya.
- Daftar hingga 10 kebijakan terkelola ARNs dapat ditentukan sebagai nilai pada `execution-iam-policy policyArns` bidang.
- Kebijakan terkelola ARNs harus berupa daftar ARN yang valid AWS atau yang dikelola Nasabah kebijakan ARN, ketika kebijakan yang dikelola Nasabah ARN ditentukan, kebijakan tersebut harus dimiliki oleh AWS akun EMR-S yang sama. JobRun
- Jika kebijakan IAM sebaris dan kebijakan terkelola digunakan, teks biasa yang Anda gunakan untuk gabungan kebijakan sebaris dan terkelola tidak dapat melebihi 2.048 karakter.
- Izin yang dihasilkan diasumsikan oleh JobRun adalah persimpangan izin dalam Peran Eksekusi dan Kebijakan IAM Eksekusi yang ditentukan.

Persimpangan Kebijakan

Izin yang dihasilkan yang diasumsikan oleh job run adalah persimpangan izin dalam Peran Eksekusi dan Kebijakan IAM Eksekusi yang ditentukan. Yang berarti setiap izin yang diperlukan harus ditentukan di kedua tempat JobRun agar dapat berfungsi. Namun, Anda dapat menentukan pernyataan izin selimut tambahan dalam kebijakan sebaris untuk izin apa pun yang tidak ingin Anda perbarui atau timpa.

Contoh

Mengingat kebijakan peran IAM eksekusi berikut:

JSON

```
{
```

```

"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "s3:*"
    ],
    "Resource": [
      "*"
    ],
    "Sid": "AllowS3"
  },
  {
    "Effect": "Allow",
    "Action": [
      "logs:DescribeLogGroups"
    ],
    "Resource": [
      "arn:aws:logs:us-west-2:123456789012:log-group::log-stream"
    ],
    "Sid": "AllowLOGSDescribeLogGroups"
  },
  {
    "Effect": "Allow",
    "Action": [
      "dynamodb:DescribeTable"
    ],
    "Resource": [
      "arn:aws:dynamodb:*:*:table/MyCompany1table"
    ],
    "Sid": "AllowDYNAMODescribeTable"
  }
]
}

```

Dan kebijakan IAM inline berikut:

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [

```

```

    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::my-test-bucket/tenant1",
        "arn:aws:s3:::my-test-bucket/tenant1/*"
      ],
      "Sid": "AllowS3GetObject"
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:*",
        "dynamodb:*"
      ],
      "Resource": [
        "*"
      ],
      "Sid": "AllowLOGS"
    }
  ]
}

```

Izin yang dihasilkan diasumsikan oleh JobRun adalah::

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::my-test-bucket/tenant1",
        "arn:aws:s3:::my-test-bucket/tenant1/*"
      ]
    }
  ]
}

```

```
    ],
    "Sid": "AllowS3GetObject"
  },
  {
    "Effect": "Allow",
    "Action": [
      "logs:DescribeLogGroups"
    ],
    "Resource": [
      "arn:aws:logs:us-west-2:123456789012:log-group:log-stream"
    ],
    "Sid": "AllowLOGSDescribeLogGroups"
  },
  {
    "Effect": "Allow",
    "Action": [
      "dynamodb:DescribeTable"
    ],
    "Resource": [
      "arn:aws:dynamodb:*:*:table/MyCompany1table"
    ],
    "Sid": "AllowDYNAMODBDescribeTable"
  }
]
}
```

Menggunakan disk yang dioptimalkan dengan shuffle

Dengan Amazon EMR rilis 7.1.0 dan yang lebih tinggi, gunakan disk yang dioptimalkan secara acak saat Anda menjalankan pekerjaan Apache Spark atau Hive untuk meningkatkan kinerja I/O-intensive workloads. Compared to standard disks, shuffle-optimized disks provide higher IOPS (I/O operasi per detik) untuk pergerakan data yang lebih cepat dan mengurangi latensi selama operasi shuffle. Disk yang dioptimalkan dengan shuffle memungkinkan Anda melampirkan ukuran disk hingga 2 TB per pekerja, jadi konfigurasi kapasitas yang sesuai untuk kebutuhan beban kerja Anda.

Manfaat utama

Disk yang dioptimalkan dengan shuffle memberikan manfaat berikut.

- Kinerja IOPS tinggi — disk yang dioptimalkan dengan shuffle memberikan IOPS yang lebih tinggi daripada disk standar, yang mengarah ke pengocokan data yang lebih efisien dan cepat selama pekerjaan Spark and Hive dan beban kerja intensif acak lainnya.
- Ukuran disk yang lebih besar - Disk yang dioptimalkan dengan shuffle mendukung ukuran disk dari 20GB hingga 2TB per pekerja, jadi pilihlah kapasitas yang sesuai berdasarkan beban kerja Anda.

Memulai

Lihat langkah-langkah berikut untuk menggunakan disk yang dioptimalkan secara acak di alur kerja Anda.

Spark

1. Buat aplikasi EMR Serverless rilis 7.1.0 dengan perintah berikut.

```
aws emr-serverless create-application \  
  --type "SPARK" \  
  --name my-application-name \  
  --release-label emr-7.1.0 \  
  --region <AWS_REGION>
```

2. Konfigurasi pekerjaan Spark Anda untuk menyertakan parameter yang akan dijalankan dengan `spark.emr-serverless.driver.disk.type` and/or `spark.emr-serverless.executor.disk.type` disk yang dioptimalkan secara acak. Anda dapat menggunakan salah satu atau kedua parameter, tergantung pada kasus penggunaan Anda.

```
aws emr-serverless start-job-run \  
  --application-id application-id \  
  --execution-role-arn job-role-arn \  
  --job-driver '{  
    "sparkSubmit": {  
      "entryPoint": "/usr/lib/spark/examples/jars/spark-examples.jar",  
      "entryPointArguments": ["1"],  
      "sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi  
      --conf spark.executor.cores=4  
      --conf spark.executor.memory=20g  
      --conf spark.driver.cores=4  
      --conf spark.driver.memory=8g  
      --conf spark.executor.instances=1  
      --conf spark.emr-serverless.executor.disk.type=shuffle_optimized"
```

```
}
}'
```

Untuk informasi lebih lanjut, lihat [Properti pekerjaan Spark](#).

Hive

1. Buat aplikasi EMR Serverless rilis 7.1.0 dengan perintah berikut.

```
aws emr-serverless create-application \
  --type "HIVE" \
  --name my-application-name \
  --release-label emr-7.1.0 \
  --region <AWS_REGION>
```

2. Konfigurasi pekerjaan Hive Anda untuk menyertakan parameter yang akan dijalankan dengan `hive.driver.disk.type` and/or `hive.tez.disk.type` disk yang dioptimalkan secara acak. Anda dapat menggunakan salah satu atau kedua parameter, tergantung pada kasus penggunaan Anda.

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "hive": {
      "query": "s3://<DOC-EXAMPLE-BUCKET>/emr-serverless-hive/query/hive-
query.sql",
      "parameters": "--hiveconf hive.log.explain.output=false"
    }
  }' \
  --configuration-overrides '{
    "applicationConfiguration": [{
      "classification": "hive-site",
      "properties": {
        "hive.exec.scratchdir": "s3://<DOC-EXAMPLE-BUCKET>/emr-
serverless-hive/hive/scratch",
        "hive.metastore.warehouse.dir": "s3://<DOC-EXAMPLE-BUCKET>/emr-
serverless-hive/hive/warehouse",
        "hive.driver.cores": "2",
        "hive.driver.memory": "4g",
        "hive.tez.container.size": "4096",
```

```

        "hive.tez.cpu.vcores": "1",
        "hive.driver.disk.type": "shuffle_optimized",
        "hive.tez.disk.type": "shuffle_optimized"
    }
}]]
}'

```

Untuk informasi lebih lanjut, [properti pekerjaan Hive](#).

Mengkonfigurasi aplikasi dengan kapasitas pra-inisialisasi

Lihat contoh berikut untuk membuat aplikasi berdasarkan Amazon EMR rilis 7.1.0. Aplikasi ini memiliki properti berikut:

- 5 driver Spark pra-inisialisasi, masing-masing dengan 2 vCPU, memori 4 GB, dan 50 GB disk yang dioptimalkan dengan shuffle.
- 50 eksekutor pra-inisialisasi, masing-masing dengan 4 vCPU, memori 8 GB, dan 500 GB disk yang dioptimalkan secara acak.

Ketika aplikasi ini menjalankan pekerjaan Spark, pertama-tama ia mengkonsumsi pekerja pra-inisialisasi dan kemudian menskalakan pekerja sesuai permintaan hingga kapasitas maksimum 400 vCPU dan 1024 GB memori. Secara opsional, Anda dapat menghilangkan kapasitas untuk salah satu atau DRIVER. EXECUTOR

Spark

```

aws emr-serverless create-application \
  --type "SPARK" \
  --name <my-application-name> \
  --release-label emr-7.1.0 \
  --initial-capacity '{
    "DRIVER": {
      "workerCount": 5,
      "workerConfiguration": {
        "cpu": "2vCPU",
        "memory": "4GB",
        "disk": "50GB",
        "diskType": "SHUFFLE_OPTIMIZED"
      }
    },

```

```

"EXECUTOR": {
  "workerCount": 50,
  "workerConfiguration": {
    "cpu": "4vCPU",
    "memory": "8GB",
    "disk": "500GB",
    "diskType": "SHUFFLE_OPTIMIZED"
  }
}
}' \
--maximum-capacity '{
  "cpu": "400vCPU",
  "memory": "1024GB"
}'

```

Hive

```

aws emr-serverless create-application \
--type "HIVE" \
--name <my-application-name> \
--release-label emr-7.1.0 \
--initial-capacity '{
  "DRIVER": {
    "workerCount": 5,
    "workerConfiguration": {
      "cpu": "2vCPU",
      "memory": "4GB",
      "disk": "50GB",
      "diskType": "SHUFFLE_OPTIMIZED"
    }
  },
  "EXECUTOR": {
    "workerCount": 50,
    "workerConfiguration": {
      "cpu": "4vCPU",
      "memory": "8GB",
      "disk": "500GB",
      "diskType": "SHUFFLE_OPTIMIZED"
    }
  }
}' \
--maximum-capacity '{
  "cpu": "400vCPU",

```

```
"memory": "1024GB"  
}'
```

Menggunakan penyimpanan tanpa server untuk Amazon EMR Tanpa Server

Dengan Amazon EMR merilis 7.12 dan yang lebih tinggi, gunakan penyimpanan tanpa server saat Anda menjalankan pekerjaan Apache Spark untuk menghilangkan penyediaan disk lokal dan mengurangi biaya pemrosesan data, serta mencegah kegagalan pekerjaan dari kendala kapasitas disk. Penyimpanan tanpa server secara otomatis menangani operasi shuffle, tumpahan disk, dan disk caching untuk pekerjaan Anda tanpa memerlukan konfigurasi kapasitas dan menyimpan data perantara tanpa biaya. Amazon EMR Serverless menyimpan data perantara dalam penyimpanan tanpa server yang dikelola sepenuhnya yang menskalakan secara otomatis berdasarkan permintaan beban kerja dan memungkinkan Spark untuk segera melepaskan pekerja komputasi saat idle, sehingga mengurangi biaya komputasi.

Manfaat utama

Penyimpanan tanpa server untuk EMR Tanpa Server memberikan manfaat berikut.

- Zero-configuration penyimpanan — Penyimpanan tanpa server menghilangkan kebutuhan untuk mengkonfigurasi jenis dan ukuran disk lokal untuk setiap aplikasi atau pekerjaan. EMR Tanpa Server secara otomatis mengelola operasi data perantara tanpa perencanaan kapasitas.
- Mencegah kegagalan pekerjaan melalui penskalaan otomatis — Skala kapasitas penyimpanan secara otomatis berdasarkan permintaan beban kerja, mencegah kegagalan pekerjaan karena kapasitas disk yang tidak mencukupi.
- Mengurangi biaya pemrosesan data — Penyimpanan tanpa server mengurangi biaya pemrosesan melalui dua mekanisme. Pertama, penyimpanan data perantara disediakan tanpa biaya — Anda hanya membayar untuk sumber daya komputasi dan memori. Kedua, penyimpanan terpisah dengan alokasi sumber daya dinamis Spark memungkinkan Spark melepaskan pekerja segera saat idle daripada menyimpannya untuk menyimpan data perantara pada disk lokal. Hal ini memungkinkan scale-out dan scale-in yang lebih cepat per tahap Spark, mengurangi biaya komputasi untuk pekerjaan di mana tahap selanjutnya membutuhkan lebih sedikit pekerja daripada tahap awal.
- Penyimpanan terenkripsi dengan isolasi tingkat pekerjaan - Semua data perantara dienkripsi saat transit dan diam dengan isolasi tingkat pekerjaan yang ketat.

- Fine-grained dukungan kontrol akses - Penyimpanan tanpa server mendukung kontrol akses berbutir halus melalui integrasi Lake Formation. AWS

Memulai

Lihat langkah-langkah berikut untuk menggunakan penyimpanan tanpa server untuk EMR Tanpa Server di alur kerja Spark Anda.

1. Buat aplikasi EMR Tanpa Server

Buat aplikasi EMR Serverless release 7.12 (atau yang lebih baru) dengan penyimpanan tanpa server diaktifkan dengan menyetel properti spark ke true dalam klasifikasi spark-defaults.

spark.aws.serverlessStorage.enabled

```
aws emr-serverless create-application \  
  --type "SPARK" \  
  --name my-application \  
  --release-label emr-7.12.0 \  
  --runtime-configuration '[{  
    "classification": "spark-defaults",  
    "properties": {  
      "spark.aws.serverlessStorage.enabled": "true"  
    }  
  }]' \  
  --region <AWS_REGION>
```

2. Memulai pekerjaan Spark

Mulai pekerjaan yang dijalankan di aplikasi Anda. Penyimpanan tanpa server untuk EMR Serverless secara otomatis menangani operasi data perantara seperti shuffle untuk pekerjaan Anda.

```
aws emr-serverless start-job-run \  
  --application-id <application-id> \  
  --execution-role-arn <job-role-arn> \  
  --job-driver '{  
    "sparkSubmit": {  
      "entryPoint": "s3://<bucket>/script.py",  
      "sparkSubmitParameters": "--conf spark.executor.cores=4  
        --conf spark.executor.memory=20g  
        --conf spark.driver.cores=4
```

```

    --conf spark.driver.memory=8g
    --conf spark.executor.instances=10"
  }
}'

```

Anda juga dapat mengaktifkan penyimpanan tanpa server untuk EMR Tanpa Server di tingkat pekerjaan bahkan ketika tidak diaktifkan di tingkat aplikasi. Ini akan meluncurkan node pekerja yang diaktifkan dengan penyimpanan tanpa server untuk memproses pekerjaan Anda. Anda juga dapat menonaktifkan penyimpanan tanpa server untuk pekerjaan tertentu dengan menyetel properti **spark.aws.serverlessStorage.enabled** Spark yang sama ke false.

```

# Turn on serverless storage for EMR serverless for a specific job
aws emr-serverless start-job-run \
  --application-id <application-id> \
  --execution-role-arn <job-role-arn> \
  --job-driver '{
"sparkSubmit": {
"entryPoint": "/usr/lib/spark/examples/jars/spark-examples.jar",
  "entryPointArguments": ["1"],
  "sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi
  --conf spark.aws.serverlessStorage.enabled": "true"
  }
}'

```

Note

Untuk terus menggunakan penyediaan disk lokal tradisional, hilangkan **spark.aws.serverlessStorage.enabled** konfigurasi atau atur ke false.

Pertimbangan dan batasan

- Versi rilis - Penyimpanan tanpa server didukung pada Amazon EMR rilis 7.12 dan yang lebih baru.
- Batas volume data - Setiap pekerjaan dapat membaca dan menulis hingga total 200 GB data perantara per pekerjaan yang dijalankan. Pekerjaan yang melebihi batas ini akan gagal dengan pesan kesalahan yang menunjukkan bahwa batas penyimpanan tanpa server telah tercapai.

- Batas waktu eksekusi Job - Penyimpanan tanpa server mendukung pekerjaan dengan batas waktu eksekusi hingga 24 jam. Pekerjaan yang dikonfigurasi untuk batas waktu eksekusi yang lebih lama akan gagal dengan pesan kesalahan.
- Pre-initialized kapasitas — Pre-initialized kapasitas pekerja tidak mendukung penyimpanan tanpa server. Ketika Anda mengonfigurasi kapasitas pra-inisialisasi, itu hanya akan digunakan oleh pekerjaan yang secara eksplisit menonaktifkan penyimpanan tanpa server di tingkat pekerjaan. Pekerjaan dengan penyimpanan tanpa server diaktifkan akan selalu menyediakan pekerja baru sesuai permintaan dan tidak akan menggunakan kapasitas pra-inisialisasi, terlepas dari konfigurasi di tingkat aplikasi.
- Jenis beban kerja - Penyimpanan tanpa server tidak didukung untuk streaming dan pekerjaan interaktif.
- Konfigurasi pekerja - Penyimpanan tanpa server tidak didukung untuk pekerja dengan 1 atau 2 vCPU.

Didukung Wilayah AWS

EMR Tanpa Server mendukung penyimpanan tanpa server di wilayah berikut:

- Timur AS (N. Virginia)
- AS Timur (Ohio)
- AS Barat (California Utara)
- AS Barat (Oregon)
- Afrika (Cape Town)
- Asia Pasifik (Hong Kong)
- Asia Pasifik (Jakarta)
- Asia Pasifik (Melbourne)
- Asia Pasifik (Mumbai)
- Asia Pasifik (Osaka)
- Asia Pasifik (Seoul)
- Asia Pasifik (Singapura)
- Asia Pasifik (Sydney)
- Asia Pasifik (Tokyo)
- (Canada (Central))

- Kanada Barat (Calgary)
- Eropa (Frankfurt)
- Eropa (Irlandia)
- Eropa (London)
- Eropa (Milan)
- Eropa (Paris)
- Eropa (Spanyol)
- Eropa (Stockholm)
- Europe (Zurich)
- Amerika Selatan (Sao Paulo)

Pekerjaan streaming untuk memproses data yang dialirkan secara terus menerus

Pekerjaan streaming di EMR Serverless adalah mode pekerjaan yang memungkinkan Anda menganalisis dan memproses data streaming dalam waktu dekat. Pekerjaan yang sudah berjalan lama ini melakukan polling data streaming dan terus memproses hasil saat data tiba. Pekerjaan streaming paling cocok untuk tugas yang memerlukan pemrosesan data waktu nyata, seperti analitik dekat waktu nyata, deteksi penipuan, dan mesin rekomendasi. Pekerjaan streaming tanpa server EMR memberikan pengoptimalan, seperti ketahanan kerja bawaan, pemantauan waktu nyata, manajemen log yang ditingkatkan, dan integrasi dengan konektor streaming.

Berikut ini adalah beberapa kasus penggunaan dengan pekerjaan streaming:

- Near real-time analytics — pekerjaan streaming di Amazon EMR Serverless memungkinkan Anda memproses data streaming hampir real-time, sehingga Anda dapat melakukan analisis real-time pada aliran data berkelanjutan, seperti data log, data sensor, atau data clickstream untuk memperoleh wawasan dan membuat keputusan tepat waktu berdasarkan informasi terbaru.
- Deteksi penipuan — gunakan pekerjaan streaming untuk menjalankan deteksi penipuan real-time dalam transaksi keuangan, operasi kartu kredit, atau aktivitas online saat Anda menganalisis aliran data dan mengidentifikasi pola atau anomali yang mencurigakan saat terjadi.
- Mesin rekomendasi — pekerjaan streaming dapat memproses data aktivitas pengguna dan memperbarui model rekomendasi. Melakukan hal itu membuka kemungkinan untuk rekomendasi yang dipersonalisasi dan real-time berdasarkan perilaku dan preferensi.

- Analisis media sosial — pekerjaan streaming dapat memproses data media sosial, seperti tweet, komentar, dan posting, sehingga organisasi dapat memantau tren, analisis sentimen, dan mengelola reputasi merek dalam waktu dekat.
- Analisis Internet of Things (IoT) — pekerjaan streaming dapat menangani dan menganalisis aliran data berkecepatan tinggi dari perangkat IoT, sensor, dan mesin yang terhubung, jadi jalankan deteksi anomali, pemeliharaan prediktif, dan kasus penggunaan analitik IoT lainnya.
- Analisis Clickstream — pekerjaan streaming dapat memproses dan menganalisis data clickstream dari situs web atau aplikasi seluler. Bisnis yang menggunakan data tersebut dapat menjalankan analitik untuk mempelajari lebih lanjut tentang perilaku pengguna, mempersonalisasi pengalaman pengguna, dan mengoptimalkan kampanye pemasaran.
- Pemantauan dan analisis log — pekerjaan streaming juga dapat memproses data log dari server, aplikasi, dan perangkat jaringan. Ini memberi Anda deteksi anomali, pemecahan masalah, serta kesehatan dan kinerja sistem.

Manfaat utama

Pekerjaan streaming di EMR Tanpa Server secara otomatis memberikan ketahanan kerja, yang merupakan kombinasi dari faktor-faktor berikut:

- Auto-retry— EMR Tanpa Server secara otomatis mencoba ulang pekerjaan apa pun yang gagal tanpa masukan manual dari Anda.
- Ketahanan Availability Zone (AZ) — EMR Tanpa Server secara otomatis mengalihkan pekerjaan streaming ke AZ yang sehat jika AZ asli mengalami masalah.
- Manajemen log:
 - Rotasi log — untuk manajemen penyimpanan disk yang lebih efisien, EMR Serverless secara berkala memutar log untuk pekerjaan streaming yang lama. Melakukannya mencegah akumulasi log yang mungkin menghabiskan semua ruang disk.
 - Pemadatan log - membantu Anda mengelola dan mengoptimalkan file log secara efisien dalam kegigihan terkelola. Pemadatan juga meningkatkan pengalaman debug saat Anda menggunakan server riwayat percikan terkelola.

Sumber data dan sink data yang didukung

EMR Serverless bekerja dengan sejumlah sumber data input dan sink data output:

- Sumber data input yang didukung - Amazon Kinesis Data Streams, Amazon Managed Streaming untuk Apache Kafka, dan cluster Apache Kafka yang dikelola sendiri. Secara default, Amazon EMR merilis 7.1.0 dan yang lebih tinggi menyertakan konektor [Amazon Kinesis Data Streams](#), jadi Anda tidak perlu membuat atau mengunduh paket tambahan apa pun.
- Sinks data keluaran yang didukung - Tabel Katalog Data AWS Glue, Amazon S3, Amazon Redshift, MySQL, PostgreSQL Oracle, Oracle, Microsoft SQL, Apache Iceberg, Delta Lake, dan Apache Hudi.

Pertimbangan dan batasan

Saat Anda menggunakan pekerjaan streaming, ingatlah pertimbangan dan batasan berikut.

- Pekerjaan streaming didukung dengan [rilis Amazon EMR 7.1.0](#) dan yang lebih tinggi.
- EMR Tanpa Server mengharapkan pekerjaan streaming berjalan untuk waktu yang lama, sehingga Anda tidak dapat mengatur batas waktu eksekusi untuk membatasi runtime pekerjaan.
- Pekerjaan streaming hanya kompatibel dengan mesin Spark, yang dibangun di atas [kerangka kerja streaming terstruktur](#).
- EMR Tanpa Server mencoba ulang pekerjaan streaming tanpa batas waktu, dan Anda tidak dapat menyesuaikan jumlah upaya maksimum. Pencegahan thrash secara otomatis disertakan untuk menghentikan percobaan ulang pekerjaan jika jumlah upaya yang gagal telah melampaui ambang batas yang ditetapkan selama jendela per jam. Ambang batas default adalah lima upaya gagal selama satu jam. Anda dapat mengonfigurasi ambang batas ini menjadi antara 1 dan 10 upaya. Untuk informasi lebih lanjut, lihat [Ketahanan Job](#).
- Pekerjaan streaming memiliki pos pemeriksaan untuk menghemat status dan kemajuan runtime, sehingga EMR Tanpa Server dapat melanjutkan pekerjaan streaming dari pos pemeriksaan terbaru. Untuk informasi lebih lanjut, lihat [Memulihkan dari kegagalan dengan Checkpointing](#) dalam dokumentasi Apache Spark.

Memulai pekerjaan streaming

Lihat petunjuk berikut untuk mempelajari cara memulai pekerjaan streaming.

1. Ikuti [Memulai Amazon EMR Tanpa Server untuk membuat aplikasi](#). Perhatikan bahwa aplikasi Anda harus menjalankan [Amazon EMR rilis 7.1.0](#) atau yang lebih tinggi.
2. Setelah aplikasi Anda siap, atur mode parameter STREAMING untuk mengirimkan pekerjaan streaming, mirip dengan AWS CLI contoh berikut.

```
aws emr-serverless start-job-run \  
--application-id <APPLICATION_ID> \  
--execution-role-arn <JOB_EXECUTION_ROLE> \  
--mode 'STREAMING' \  
--job-driver '{  
  "sparkSubmit": {  
    "entryPoint": "s3://<streaming script>",  
    "entryPointArguments": ["s3://<DOC-EXAMPLE-BUCKET-OUTPUT>/output"],  
    "sparkSubmitParameters": "--conf spark.executor.cores=4  
      --conf spark.executor.memory=16g  
      --conf spark.driver.cores=4  
      --conf spark.driver.memory=16g  
      --conf spark.executor.instances=3"  
  }  
}'
```

Konektor streaming yang didukung

Konektor streaming memfasilitasi membaca data dari sumber streaming dan juga dapat menulis data ke wastafel streaming.

Berikut ini adalah konektor streaming yang didukung:

Konektor Amazon Kinesis Data Streams

Konektor [Amazon Kinesis Data Streams](#) untuk Apache Spark memungkinkan pembuatan aplikasi streaming dan pipeline yang menggunakan data dari dan menulis data ke Amazon Kinesis Data Streams. Konektor mendukung peningkatan konsumsi kipas dengan tingkat throughput baca khusus hingga 2MB/second per pecahan. Secara default, Amazon EMR Serverless 7.1.0 dan yang lebih tinggi menyertakan konektor, jadi Anda tidak perlu membuat atau mengunduh paket tambahan apa pun. Untuk informasi lebih lanjut tentang konektor, lihat halaman [spark-sql-kinesis-connector](#) di GitHub

Berikut ini adalah contoh bagaimana memulai pekerjaan dengan ketergantungan konektor Kinesis Data Streams.

```
aws emr-serverless start-job-run \  
--application-id <APPLICATION_ID> \  
--execution-role-arn <JOB_EXECUTION_ROLE> \  
--mode 'STREAMING' \  

```

```
--job-driver '{
  "sparkSubmit": {
    "entryPoint": "s3://<Kinesis-streaming-script>",
    "entryPointArguments": ["s3://<DOC-EXAMPLE-BUCKET-OUTPUT>/output"],
    "sparkSubmitParameters": "--conf spark.executor.cores=4
      --conf spark.executor.memory=16g
      --conf spark.driver.cores=4
      --conf spark.driver.memory=16g
      --conf spark.executor.instances=3
      --jars /usr/share/aws/kinesis/spark-sql-kinesis/lib/spark-streaming-
sql-kinesis-connector.jar"
  }
}'
```

Untuk terhubung ke Kinesis Data Streams, konfigurasi aplikasi EMR Tanpa Server dengan akses VPC dan gunakan titik akhir VPC untuk memungkinkan akses pribadi. atau gunakan NAT Gateway untuk mendapatkan akses publik. Untuk informasi selengkapnya, lihat [Mengonfigurasi akses VPC](#). Anda juga harus memastikan bahwa peran runtime pekerjaan Anda memiliki izin baca dan tulis yang diperlukan untuk mengakses aliran data yang diperlukan. Untuk mempelajari lebih lanjut tentang cara mengonfigurasi peran runtime pekerjaan, lihat peran [runtime Job untuk Amazon EMR Tanpa Server](#). Untuk daftar lengkap semua izin yang diperlukan, lihat halaman [spark-sql-kinesis-connector](#) di GitHub

Konektor Apache Kafka

Konektor Apache Kafka untuk streaming terstruktur Spark adalah konektor open-source dari komunitas Spark dan tersedia di repositori Maven. Konektor ini memfasilitasi aplikasi streaming terstruktur Spark untuk membaca data dari dan menulis data ke Apache Kafka yang dikelola sendiri dan Amazon Managed Streaming for Apache Kafka. Untuk informasi lebih lanjut tentang konektor, lihat [Panduan Integrasi Streaming Terstruktur+Kafka](#) dalam dokumentasi Apache Spark.

Contoh berikut menunjukkan cara memasukkan konektor Kafka dalam permintaan menjalankan pekerjaan Anda.

```
aws emr-serverless start-job-run \
--application-id <APPLICATION_ID> \
--execution-role-arn <JOB_EXECUTION_ROLE> \
--mode 'STREAMING' \
--job-driver '{
  "sparkSubmit": {
    "entryPoint": "s3://<Kafka-streaming-script>",
```

```

    "entryPointArguments": ["s3://<DOC-EXAMPLE-BUCKET-OUTPUT>/output"],
    "sparkSubmitParameters": "--conf spark.executor.cores=4
        --conf spark.executor.memory=16g
        --conf spark.driver.cores=4
        --conf spark.driver.memory=16g
        --conf spark.executor.instances=3
        --packages org.apache.spark:spark-sql-
kafka-0-10_2.12:<KAFKA_CONNECTOR_VERSION>"
    }
}'

```

Versi konektor Apache Kafka tergantung pada versi rilis EMR Serverless Anda dan versi Spark yang sesuai. Untuk menemukan versi Kafka yang benar, lihat lihat lihat lihat Panduan [Streaming Terstruktur+Integrasi Kafka](#).

Untuk menggunakan Amazon Managed Streaming for Apache Kafka Kafka dengan autentikasi IAM, sertakan dependensi lain untuk mengaktifkan konektor Kafka terhubung ke Amazon MSK dengan IAM. Untuk informasi lebih lanjut, lihat repositori [aws-msk-iam-auth](#) di GitHub Anda juga harus memastikan bahwa peran runtime pekerjaan memiliki izin IAM yang diperlukan. Contoh berikut menunjukkan bagaimana menggunakan konektor dengan otentikasi IAM.

```

aws emr-serverless start-job-run \
--application-id <APPLICATION_ID> \
--execution-role-arn <JOB_EXECUTION_ROLE> \
--mode 'STREAMING' \
--job-driver '{
    "sparkSubmit": {
        "entryPoint": "s3://<Kafka-streaming-script>",
        "entryPointArguments": ["s3://<DOC-EXAMPLE-BUCKET-OUTPUT>/output"],
        "sparkSubmitParameters": "--conf spark.executor.cores=4
            --conf spark.executor.memory=16g
            --conf spark.driver.cores=4
            --conf spark.driver.memory=16g
            --conf spark.executor.instances=3
            --packages org.apache.spark:spark-sql-
kafka-0-10_2.12:<KAFKA_CONNECTOR_VERSION>,software.amazon.msk:aws-msk-iam-
auth:<MSK_IAM_LIB_VERSION>"
    }
}'

```

Untuk menggunakan konektor Kafka dan pustaka otentikasi IAM dari Amazon MSK, konfigurasi aplikasi EMR Tanpa Server dengan akses VPC. Subnet Anda harus memiliki akses Internet dan

menggunakan NAT Gateway untuk mengakses dependensi Maven. Untuk informasi selengkapnya, lihat [Mengonfigurasi akses VPC](#). Subnet harus memiliki konektivitas jaringan untuk mengakses cluster Kafka. Ini benar terlepas dari apakah cluster Kafka Anda dikelola sendiri atau jika Anda menggunakan Amazon Managed Streaming for Apache Kafka.

Manajemen log pekerjaan streaming

Pekerjaan streaming mendukung rotasi log untuk log aplikasi Spark dan log peristiwa, dan pemadatan log untuk log peristiwa Spark. Ini membantu Anda mengelola sumber daya Anda secara efektif.

Rotasi log

Pekerjaan streaming mendukung rotasi log untuk log aplikasi Spark dan log peristiwa. Rotasi log mencegah pekerjaan streaming panjang menghasilkan file log besar yang mungkin menghabiskan semua ruang disk Anda yang tersedia. Rotasi log membantu Anda menghemat penyimpanan disk dan mencegah kegagalan pekerjaan karena ruang disk yang rendah. Untuk informasi lebih lanjut, lihat [Rotating logs](#).

Pemadatan log

Pekerjaan streaming juga mendukung pemadatan log untuk log peristiwa Spark setiap kali logging terkelola tersedia. Untuk detail selengkapnya tentang logging terkelola, lihat [Logging dengan penyimpanan terkelola](#). Pekerjaan streaming dapat berjalan untuk waktu yang lama, dan jumlah data acara dapat bertambah dari waktu ke waktu dan secara signifikan meningkatkan ukuran file log. Spark History Server membaca dan memuat peristiwa ini ke dalam memori untuk UI aplikasi Spark. Proses ini dapat menyebabkan latensi dan biaya tinggi, terutama jika log peristiwa yang disimpan di Amazon S3 sangat besar.

Pemadatan log mengurangi ukuran log peristiwa, sehingga Server Riwayat Spark tidak perlu memuat lebih dari 1 GB log peristiwa kapan saja. Untuk informasi lebih lanjut, lihat [Pemantauan dan Instrumentasi](#) dalam dokumentasi Apache Spark.

Menggunakan konfigurasi Spark saat Anda menjalankan pekerjaan EMR Tanpa Server

Anda dapat menjalankan pekerjaan Spark pada aplikasi dengan type parameter yang disetel keSPARK. Pekerjaan harus kompatibel dengan versi Spark yang kompatibel dengan versi rilis Amazon EMR. Misalnya, ketika Anda menjalankan pekerjaan dengan Amazon EMR rilis 6.6.0,

pekerjaan Anda harus kompatibel dengan Apache Spark 3.2.0. Untuk informasi tentang versi aplikasi untuk setiap rilis, lihat [Amazon EMR Versi rilis tanpa server](#).

Parameter pekerjaan percikan

Saat Anda menggunakan [StartJobRunAPI](#) untuk menjalankan pekerjaan Spark, tentukan parameter berikut.

Parameter yang diperlukan

- [Peran runtime pekerjaan percikan](#)
- [Parameter pengemudi pekerjaan percikan](#)
- [Parameter penggantian konfigurasi percikan](#)
- [Spark optimasi alokasi sumber daya dinamis](#)

Peran runtime pekerjaan percikan

Gunakan **executionRoleArn** untuk menentukan ARN untuk peran IAM yang digunakan aplikasi Anda untuk menjalankan pekerjaan Spark. Peran ini harus berisi izin berikut:

- Baca dari bucket S3 atau sumber data lain di mana data Anda berada
- Baca dari bucket atau awalan S3 tempat PySpark skrip atau file JAR Anda berada
- Menulis ke ember S3 di mana Anda ingin menulis hasil akhir Anda
- Menulis log ke bucket S3 atau awalan yang menentukan `S3MonitoringConfiguration`
- Akses ke kunci KMS jika Anda menggunakan kunci KMS untuk mengenkripsi data di bucket S3 Anda
- Akses ke Katalog Data AWS Glue jika Anda menggunakan SparkSQL

Jika pekerjaan Spark Anda membaca atau menulis data ke atau dari sumber data lain, tentukan izin yang sesuai dalam peran IAM ini. Jika Anda tidak memberikan izin ini ke peran IAM, pekerjaan mungkin gagal. Untuk informasi lebih lanjut, lihat [Peran runtime Job untuk Amazon EMR Tanpa Server](#) dan [Menyimpan log](#).

Parameter pengemudi pekerjaan percikan

Gunakan **jobDriver** untuk memberikan masukan pada pekerjaan. Parameter driver pekerjaan hanya menerima satu nilai untuk jenis pekerjaan yang ingin Anda jalankan. Untuk pekerjaan Spark,

nilai parameternya adalah `sparkSubmit`. Anda dapat menggunakan jenis pekerjaan ini untuk menjalankan Scala, Java PySpark, dan pekerjaan lain yang didukung melalui pengiriman Spark. Pekerjaan percikan memiliki parameter berikut:

- **sparkSubmitParameters**— Ini adalah parameter Spark tambahan yang ingin Anda kirim ke pekerjaan. Gunakan parameter ini untuk mengganti properti Spark default seperti memori driver atau jumlah pelaksana, seperti yang didefinisikan dalam argumen atau. `--conf --class`
- **entryPointArguments**— Ini adalah array argumen yang ingin Anda sampaikan ke JAR utama Anda atau file Python. Anda harus menangani membaca parameter ini menggunakan kode `entrypoint` Anda. Pisahkan setiap argumen dalam array dengan koma.
- **entryPoint**— Ini adalah referensi di Amazon S3 ke JAR utama atau file Python yang ingin Anda jalankan. Jika Anda menjalankan Scala atau Java JAR, tentukan kelas entri utama dalam `sparkSubmitParameters` menggunakan `--class` argumen.

Untuk informasi tambahan, lihat [Peluncuran Aplikasi dengan spark-submit](#).

Parameter penggantian konfigurasi percikan

Gunakan **configurationOverrides** untuk mengganti properti konfigurasi tingkat pemantauan dan tingkat aplikasi. Parameter ini menerima objek JSON dengan dua bidang berikut:

- **monitoringConfiguration**- Gunakan bidang ini untuk menentukan URL Amazon S3 (`s3MonitoringConfiguration`) di mana Anda ingin pekerjaan EMR Tanpa Server untuk menyimpan log pekerjaan Spark Anda. Pastikan Anda telah membuat bucket ini dengan yang sama Akun AWS yang meng-host aplikasi Anda, dan di tempat yang sama Wilayah AWS di mana pekerjaan Anda berjalan.
- **applicationConfiguration**— Untuk mengganti konfigurasi default untuk aplikasi, Anda dapat menyediakan objek konfigurasi di bidang ini. 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.

Note

Klasifikasi konfigurasi yang tersedia bervariasi menurut rilis EMR Tanpa Server tertentu. Misalnya, klasifikasi untuk `Log4j` kustom `spark-driver-log4j2` dan hanya `spark-executor-log4j2` tersedia dengan rilis 6.8.0 dan yang lebih tinggi.

Jika Anda menggunakan konfigurasi yang sama dalam penggantian aplikasi dan dalam parameter pengiriman Spark, parameter pengiriman Spark akan diprioritaskan. Konfigurasi peringkat dalam prioritas sebagai berikut, dari tertinggi ke terendah:

- Konfigurasi yang disediakan EMR Tanpa Server saat dibuat. `SparkSession`
- Konfigurasi yang Anda berikan sebagai bagian dari `sparkSubmitParameters --conf` argumen.
- Konfigurasi yang Anda berikan sebagai bagian dari penggantian aplikasi Anda ketika Anda memulai pekerjaan.
- Konfigurasi yang Anda berikan sebagai bagian dari `runtimeConfiguration` saat Anda membuat aplikasi.
- Konfigurasi yang dioptimalkan yang digunakan Amazon EMR untuk rilis.
- Konfigurasi sumber terbuka default untuk aplikasi.

Untuk informasi selengkapnya tentang mendeklarasikan konfigurasi di tingkat aplikasi, dan mengganti konfigurasi selama menjalankan pekerjaan, lihat [Konfigurasi aplikasi default untuk EMR Serverless](#)

Spark optimasi alokasi sumber daya dinamis

Gunakan `dynamicAllocationOptimization` untuk mengoptimalkan penggunaan sumber daya di EMR Tanpa Server. Menyetel properti ini ke `true` dalam klasifikasi konfigurasi Spark Anda menunjukkan kepada EMR Tanpa Server untuk mengoptimalkan alokasi sumber daya pelaksana untuk menyelaraskan tingkat permintaan Spark dan membatalkan pelaksana dengan tingkat di mana EMR Serverless membuat dan melepaskan pekerja. Dengan demikian, EMR Serverless lebih optimal menggunakan kembali pekerja di seluruh tahap, menghasilkan biaya yang lebih rendah saat menjalankan pekerjaan dengan beberapa tahap sambil mempertahankan kinerja yang sama.

Properti ini tersedia di semua versi rilis Amazon EMR.

Berikut ini adalah klasifikasi konfigurasi sampel dengandynamicAllocationOptimization.

```
[
  {
    "Classification": "spark",
    "Properties": {
      "dynamicAllocationOptimization": "true"
    }
  }
]
```

Pertimbangkan hal berikut jika Anda menggunakan optimasi alokasi dinamis:

- Pengoptimalan ini tersedia untuk pekerjaan Spark yang Anda aktifkan alokasi sumber daya dinamis.
- Untuk mencapai efisiensi biaya terbaik, kami menyarankan konfigurasi batas penskalaan atas pada pekerja yang menggunakan pengaturan tingkat pekerjaan `spark.dynamicAllocation.maxExecutors` atau pengaturan [kapasitas maksimum tingkat aplikasi](#) berdasarkan beban kerja Anda.
- Anda mungkin tidak melihat peningkatan biaya dalam pekerjaan yang lebih sederhana. Misalnya, jika pekerjaan Anda berjalan pada kumpulan data kecil atau selesai berjalan dalam satu tahap, Spark mungkin tidak memerlukan jumlah pelaksana yang lebih besar atau beberapa peristiwa penskalaan.
- Pekerjaan dengan urutan tahap besar, tahapan yang lebih kecil, dan kemudian tahap besar lagi mungkin mengalami regresi dalam runtime pekerjaan. Karena EMR Serverless menggunakan sumber daya secara lebih efisien, ini dapat menyebabkan lebih sedikit pekerja yang tersedia untuk tahap yang lebih besar, yang mengarah ke runtime yang lebih lama.

Properti pekerjaan percikan

Tabel berikut mencantumkan properti Spark opsional dan nilai defaultnya yang dapat Anda ganti saat mengirimkan pekerjaan Spark.

Properti Spark opsional dan nilai default

Key	Deskripsi	Nilai default
<code>spark.archives</code>	Daftar arsip yang dipisahkan koma yang diekstrak Spark ke	NULL

Key	Deskripsi	Nilai default
	dalam direktori kerja masing-masing pelaksana. Jenis file yang didukung termasuk .jar, .tar.gz, .tgz dan .zip. Untuk menentukan nama direktori yang akan diekstrak, tambahkan # setelah nama file yang ingin Anda ekstrak. Misalnya, file.zip#directory .	
spark.authenticate	Opsi yang mengaktifkan otentikasi koneksi internal Spark.	TRUE
spark.driver.cores	Jumlah core yang digunakan driver.	4
spark.driver.extraJavaOptions	Opsi Java ekstra untuk driver Spark.	NULL
spark.driver.memory	Jumlah memori yang digunakan pengemudi.	14G
spark.dynamicAllocation.enabled	Opsi yang mengaktifkan alokasi sumber daya dinamis. Opsi ini menaikkan atau menurunkan jumlah pelaksana yang terdaftar dengan aplikasi, berdasarkan beban kerja.	TRUE

Key	Deskripsi	Nilai default
<code>spark.dynamicAllocation.executorIdleTimeout</code>	Lamanya waktu seorang eksekutor dapat tetap menganggur sebelum Spark menghapusnya. Ini hanya berlaku jika Anda mengaktifkan alokasi dinamis.	60-an
<code>spark.dynamicAllocation.initialExecutors</code>	Jumlah awal pelaksana untuk dijalankan jika Anda mengaktifkan alokasi dinamis.	3
<code>spark.dynamicAllocation.maxExecutors</code>	Batas atas untuk jumlah pelaksana jika Anda mengaktifkan alokasi dinamis.	Untuk 6.10.0 dan lebih tinggi, <code>infinity</code> Untuk 6.9.0 dan lebih rendah, <code>100</code>
<code>spark.dynamicAllocation.minExecutors</code>	Batas bawah untuk jumlah pelaksana jika Anda mengaktifkan alokasi dinamis.	0
<code>spark.emr-serverless.allocation.batch.size</code>	Jumlah kontainer untuk meminta dalam setiap siklus alokasi pelaksana. Ada kesenjangan satu detik antara setiap siklus alokasi.	20
<code>spark.emr-serverless.driver.disk</code>	Disk driver Spark.	20G
<code>spark.emr-serverless.driverEnv.</code> [KEY]	Opsi yang menambahkan variabel lingkungan ke driver Spark.	NULL
<code>spark.emr-serverless.executor.disk</code>	Disk eksekutor Spark.	20G

Key	Deskripsi	Nilai default
<code>spark.emr-serverless.memoryOverheadFactor</code>	Mengatur overhead memori untuk ditambahkan ke driver dan memori kontainer pelaksana.	0,1
<code>spark.emr-serverless.driver.disk.type</code>	Jenis disk yang terpasang pada driver Spark.	Standar
<code>spark.emr-serverless.executor.disk.type</code>	Jenis disk yang melekat pada pelaksana Spark.	Standar
<code>spark.executor.cores</code>	Jumlah core yang digunakan setiap eksekutor.	4
<code>spark.executor.extraJavaOptions</code>	Opsi Java ekstra untuk eksekutor Spark.	NULL
<code>spark.executor.instances</code>	Jumlah kontainer pelaksana Spark untuk dialokasikan.	3
<code>spark.executor.memory</code>	Jumlah memori yang digunakan setiap eksekutor.	14G
<code>spark.executorEnv. [KEY]</code>	Opsi yang menambahkan variabel lingkungan ke pelaksana Spark.	NULL
<code>spark.files</code>	Daftar file yang dipisahkan koma untuk masuk ke direktori kerja masing-masing pelaksana. Anda dapat mengakses jalur file dari file-file ini di pelaksana dengan <code>sparkFiles.get(fileName)</code> .	NULL

Key	Deskripsi	Nilai default
<code>spark.hadoop.hive.metastore.client.factory.class</code>	Kelas implementasi metastore Hive.	NULL
<code>spark.jars</code>	Guci tambahan untuk ditambahkan ke classpath runtime driver dan executor.	NULL
<code>spark.network.crypto.enabled</code>	Opsi yang mengaktifkan enkripsi AES-based RPC. Ini termasuk protokol otentikasi yang ditambahkan di Spark 2.2.0.	FALSE
<code>spark.sql.warehouse.dir</code>	Lokasi default untuk database dan tabel terkelola.	Nilai <code>\$PWD/spark-warehouse</code>
<code>spark.submit.pyFiles</code>	Daftar .zip, .egg, atau .py file yang dipisahkan koma untuk ditempatkan di aplikasi untuk PYTHONPATH Python.	NULL

Tabel berikut mencantumkan parameter pengiriman Spark default.

Parameter kirimkan Spark default

Key	Deskripsi	Nilai default
<code>archives</code>	Daftar arsip yang dipisahkan koma yang diekstrak Spark ke dalam direktori kerja masing-masing pelaksana.	NULL
<code>class</code>	Kelas utama aplikasi (untuk aplikasi Java dan Scala).	NULL

Key	Deskripsi	Nilai default
<code>conf</code>	Properti konfigurasi Spark arbitrer.	NULL
<code>driver-cores</code>	Jumlah core yang digunakan driver.	4
<code>driver-memory</code>	Jumlah memori yang digunakan pengemudi.	14G
<code>executor-cores</code>	Jumlah core yang digunakan setiap eksekutor.	4
<code>executor-memory</code>	Jumlah memori yang digunakan eksekutor.	14G
<code>files</code>	Daftar file yang dipisahkan koma untuk ditempatkan di direktori kerja masing-masing pelaksana. Anda dapat mengakses jalur file dari file-file ini di pelaksana dengan <code>SparkFiles.get(<i>fileName</i>)</code> .	NULL
<code>jars</code>	Daftar stoples yang dipisahkan koma untuk disertakan pada classpath driver dan eksekutor.	NULL
<code>num-executors</code>	Jumlah eksekutor yang akan diluncurkan.	3
<code>py-files</code>	Daftar <code>.zip</code> , <code>.egg</code> , atau <code>.py</code> file yang dipisahkan koma untuk ditempatkan di aplikasi untuk <code>PYTHONPATH</code> Python.	NULL

Key	Deskripsi	Nilai default
verbose	Opsi yang mengaktifkan output debug tambahan.	NULL

Praktik terbaik konfigurasi sumber daya

Mengkonfigurasi sumber daya driver dan pelaksana melalui API StartJobRun

Note

Spark driver dan inti pelaksana dan properti memori, jika ditentukan, harus langsung ditentukan dalam permintaan StartJobRun API.

Mengkonfigurasi sumber daya Anda dengan cara ini memastikan bahwa EMR Serverless dapat mengalokasikan sumber daya yang benar sebelum menjalankan pekerjaan. Ini berbeda dengan pengaturan yang disediakan dalam skrip pengguna, seperti di file.py atau .jar, yang dievaluasi terlambat, karena pekerja driver dan eksekutor terkadang disediakan sebelumnya sebelum eksekusi skrip dimulai. Ada dua cara yang didukung untuk mengonfigurasi sumber daya ini selama pengiriman pekerjaan:

Opsi 1: Gunakan percikan SubmitParameters

```
"jobDriver": {
  "sparkSubmit": {
    "entryPoint": "s3://your-script-path.py",
    "sparkSubmitParameters": "-conf spark.driver.memory=4g \
-conf spark.driver.cores=2 \
-conf spark.executor.memory=8g \
-conf spark.executor.cores=4"
  }
}
```

Opsi 2: Gunakan ConfigurationOverrides untuk klasifikasi spark-defaults

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

```
{
  "classification": "spark-defaults",
  "properties": {
    "spark.driver.memory": "4g",
    "spark.driver.cores": "2",
    "spark.executor.memory": "8g",
    "spark.executor.cores": "4"
  }
}
]
```

Contoh percikan

Contoh berikut menunjukkan cara menggunakan StartJobRun API untuk menjalankan skrip Python. Untuk tutorial ujung ke ujung yang menggunakan contoh ini, lihat [Memulai dengan Amazon EMR Tanpa Server](#) Anda dapat menemukan contoh tambahan tentang cara menjalankan PySpark pekerjaan dan menambahkan dependensi Python di repositori [EMR Serverless Sampel](#). [GitHub](#)

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "s3://us-east-1.elasticmapreduce/emr-containers/samples/
wordcount/scripts/wordcount.py",
      "entryPointArguments": ["s3://amzn-s3-demo-destination-bucket/
wordcount_output"],
      "sparkSubmitParameters": "--conf spark.executor.cores=1 --conf
spark.executor.memory=4g --conf spark.driver.cores=1 --conf spark.driver.memory=4g --
conf spark.executor.instances=1"
    }
  }'
```

Contoh berikut menunjukkan cara menggunakan StartJobRun API untuk menjalankan Spark JAR.

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "/usr/lib/spark/examples/jars/spark-examples.jar",
```

```
        "entryPointArguments": ["1"],
        "sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi --conf
spark.executor.cores=4 --conf spark.executor.memory=20g --conf spark.driver.cores=4 --
conf spark.driver.memory=8g --conf spark.executor.instances=1"
    }
}'
```

Menggunakan konfigurasi Hive saat Anda menjalankan pekerjaan EMR Tanpa Server

Anda dapat menjalankan pekerjaan Hive pada aplikasi dengan type parameter yang disetel keHIVE. Jobs harus kompatibel dengan versi Hive yang kompatibel dengan versi rilis Amazon EMR. Misalnya, ketika Anda menjalankan pekerjaan pada aplikasi dengan Amazon EMR rilis 6.6.0, pekerjaan Anda harus kompatibel dengan Apache Hive 3.1.2. Untuk informasi tentang versi aplikasi untuk setiap rilis, lihat[Amazon EMR Versi rilis tanpa server](#).

Parameter pekerjaan sarang

Saat Anda menggunakan [StartJobRunAPI](#) untuk menjalankan pekerjaan Hive, tentukan parameter berikut.

Parameter yang diperlukan

- [Peran runtime pekerjaan sarang](#)
- [Parameter pengemudi pekerjaan sarang](#)
- [Parameter penggantian konfigurasi sarang](#)

Peran runtime pekerjaan sarang

Gunakan **executionRoleArn** untuk menentukan ARN untuk peran IAM yang digunakan aplikasi Anda untuk menjalankan pekerjaan Hive. Peran ini harus berisi izin berikut:

- Baca dari bucket S3 atau sumber data lain di mana data Anda berada
- Baca dari bucket atau awalan S3 tempat file kueri Hive dan file kueri init Anda berada
- Baca dan tulis ke bucket S3 tempat direktori Hive Scratch dan direktori gudang Hive Metastore Anda berada
- Menulis ke ember S3 di mana Anda ingin menulis hasil akhir Anda

- Menulis log ke bucket S3 atau awalan yang menentukan `S3MonitoringConfiguration`
- Akses ke kunci KMS jika Anda menggunakan kunci KMS untuk mengenkripsi data di bucket S3 Anda
- Akses ke Katalog Data AWS Glue

Jika pekerjaan Hive Anda membaca atau menulis data ke atau dari sumber data lain, tentukan izin yang sesuai dalam peran IAM ini. Jika Anda tidak memberikan izin ini ke peran IAM, pekerjaan Anda mungkin gagal. Untuk informasi lebih lanjut, lihat [Peran runtime Job untuk Amazon EMR Tanpa Server](#).

Parameter pengemudi pekerjaan sarang

Gunakan **jobDriver** untuk memberikan masukan pada pekerjaan. Parameter driver pekerjaan hanya menerima satu nilai untuk jenis pekerjaan yang ingin Anda jalankan. Saat Anda menentukan hive sebagai jenis pekerjaan, EMR Serverless meneruskan kueri Hive ke parameter. `jobDriver` Pekerjaan sarang memiliki parameter berikut:

- **query**— Ini adalah referensi di Amazon S3 ke file kueri Hive yang ingin Anda jalankan.
- **parameters**— Ini adalah properti konfigurasi Hive tambahan yang ingin Anda timpa. Untuk mengganti properti, teruskan ke parameter ini sebagai `--hiveconf property=value`. Untuk mengganti variabel, berikan mereka ke parameter ini sebagai `--hivevar key=value`.
- **initQueryFile**— Ini adalah file query init Hive. Hive menjalankan file ini sebelum kueri Anda dan dapat menggunakannya untuk menginisialisasi tabel.

Parameter penggantian konfigurasi sarang

Gunakan **configurationOverrides** untuk mengganti properti konfigurasi tingkat pemantauan dan tingkat aplikasi. Parameter ini menerima objek JSON dengan dua bidang berikut:

- **monitoringConfiguration**— Gunakan bidang ini untuk menentukan URL Amazon S3 (`s3MonitoringConfiguration`) di mana Anda ingin pekerjaan EMR Tanpa Server untuk menyimpan log pekerjaan Hive Anda. Pastikan Anda membuat bucket ini dengan yang sama Akun AWS yang meng-host aplikasi Anda, dan di tempat yang sama Wilayah AWS di mana pekerjaan Anda berjalan.
- **applicationConfiguration**— Anda dapat memberikan objek konfigurasi di bidang ini untuk mengganti konfigurasi default untuk aplikasi. 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.

Note

Klasifikasi konfigurasi yang tersedia bervariasi menurut rilis EMR Tanpa Server tertentu. Misalnya, klasifikasi untuk Log4j kustom `spark-driver-log4j2` dan hanya `spark-executor-log4j2` tersedia dengan rilis 6.8.0 dan yang lebih tinggi.

Jika Anda melewati konfigurasi yang sama dalam penggantian aplikasi dan dalam parameter Hive, parameter Hive akan diprioritaskan. Daftar berikut memberi peringkat konfigurasi dari prioritas tertinggi hingga prioritas terendah.

- Konfigurasi yang Anda berikan sebagai bagian dari parameter Hive. `--hiveconf property=value`
- Konfigurasi yang Anda berikan sebagai bagian dari penggantian aplikasi Anda ketika Anda memulai pekerjaan.
- Konfigurasi yang Anda berikan sebagai bagian dari `runtimeConfiguration` saat Anda membuat aplikasi.
- Konfigurasi yang dioptimalkan yang ditetapkan Amazon EMR untuk rilis.
- Konfigurasi sumber terbuka default untuk aplikasi.

Untuk informasi selengkapnya tentang mendeklarasikan konfigurasi di tingkat aplikasi, dan mengganti konfigurasi selama menjalankan pekerjaan, lihat [Konfigurasi aplikasi default untuk EMR Serverless](#)

Properti pekerjaan sarang

Tabel berikut mencantumkan properti wajib yang dikonfigurasi saat Anda mengirimkan pekerjaan Hive.

Properti pekerjaan Wajib Hive

Pengaturan	Deskripsi
<code>hive.exec.scratchdir</code>	Lokasi Amazon S3 tempat EMR Tanpa Server membuat file sementara selama eksekusi pekerjaan Hive.
<code>hive.metastore.warehouse.dir</code>	Lokasi Amazon S3 database untuk tabel terkelola di Hive.

Tabel berikut mencantumkan properti Hive opsional dan nilai defaultnya yang dapat Anda ganti saat mengirimkan pekerjaan Hive.

Properti Hive opsional dan nilai default

Pengaturan	Deskripsi	Nilai default
<code>fs.s3.customAWSCredentialsProvider</code>	Penyedia AWS Credentials yang ingin Anda gunakan.	<code>com.amazonaws.auth.defaultAWSCredentialsProviderChain</code>
<code>fs.s3a.aws.credentials.provider</code>	Penyedia AWS Credentials yang ingin Anda gunakan dengan sistem file S3A.	<code>com.amazonaws.auth.defaultAWSCredentialsProviderChain</code>
<code>hive.auto.convert.join</code>	Opsi yang mengaktifkan konversi otomatis gabungan umum menjadi mapjoins, berdasarkan ukuran file input.	TRUE
<code>hive.auto.convert.join.noconditionaltask</code>	Opsi yang mengaktifkan pengoptimalan saat Hive mengonversi gabungan umum menjadi mapjoin berdasarkan ukuran file input.	TRUE

Pengaturan	Deskripsi	Nilai default
<code>hive.auto.convert.join.noconditionaltask.size</code>	Gabungan mengonversi langsung ke mapjoin di bawah ukuran ini.	Nilai optimal dihitung berdasarkan memori tugas Tez
<code>hive.cbo.enable</code>	Opsi yang mengaktifkan pengoptimalan berbasis biaya dengan kerangka Calcite.	TRUE
<code>hive.cli.tez.session.async</code>	Opsi untuk memulai sesi Tez latar belakang saat kueri Hive Anda dikompilasi. Saat disetel ke <code>false</code> , Tez AM diluncurkan setelah kueri Hive Anda dikompilasi.	TRUE
<code>hive.compute.query.using.stats</code>	Opsi yang mengaktifkan Hive untuk menjawab pertanyaan tertentu dengan statistik yang disimpan di metastore. Untuk statistik dasar, atur <code>hive.stats.autogather</code> ke <code>TRUE</code> . Untuk koleksi kueri yang lebih canggih, jalankan <code>analyze table queries</code> .	TRUE
<code>hive.default.fileformat</code>	Format file default untuk <code>CREATE TABLE</code> pernyataan. Anda dapat secara eksplisit mengganti ini jika Anda menentukan <code>STORED AS [FORMAT]</code> dalam perintah Anda. <code>CREATE TABLE</code>	TEXTFILE
<code>hive.driver.cores</code>	Jumlah core yang digunakan untuk proses driver Hive.	2

Pengaturan	Deskripsi	Nilai default
<code>hive.driver.disk</code>	Ukuran disk untuk driver Hive.	20G
<code>hive.driver.disk.type</code>	Jenis disk untuk driver Hive.	Standar
<code>hive.tez.disk.type</code>	Ukuran disk untuk pekerja tez.	Standar
<code>hive.driver.memory</code>	Jumlah memori yang digunakan per proses driver Hive. The Hive CLI dan Tez Application Master berbagi memori ini secara setara dengan 20% ruang kepala.	6G
<code>hive.emr-serverless.launch.env.[<i>KEY</i>]</code>	Opsi untuk mengatur variabel <i>KEY</i> lingkungan di semua proses khusus HIVE, seperti driver Hive Anda, Tez AM, dan tugas Tez.	
<code>hive.exec.dynamic.partition</code>	Opsi yang mengaktifkan partisi dinamis di DML/DDDL.	TRUE
<code>hive.exec.dynamic.partition.mode</code>	Opsi yang menentukan apakah Anda ingin menggunakan modus ketat atau modus non-ketat. Dalam mode ketat, tentukan setidaknya satu partisi statis jika Anda secara tidak sengaja menimpa semua partisi. Dalam mode non-ketat, semua partisi dibiarkan dinamis.	strict

Pengaturan	Deskripsi	Nilai default
<code>hive.exec.max.dynamic.partitions</code>	Jumlah maksimum partisi dinamis yang dibuat Hive secara total.	1000
<code>hive.exec.max.dynamic.partitions.per.node</code>	Jumlah maksimum partisi dinamis yang dibuat Hive di setiap node mapper dan reducer.	100
<code>hive.exec.orc.split.strategy</code>	Mengharapkan salah satu nilai berikut:BI,ETL, atauHYBRID. Ini bukan konfigurasi tingkat pengguna. BI menentukan bahwa Anda ingin menghabiskan lebih sedikit waktu dalam pembuatan terpisah dibandingkan dengan eksekusi kueri. ETL menentukan bahwa Anda ingin menghabiskan lebih banyak waktu dalam generasi terpisah. HYBRID menentukan pilihan strategi sebelumnya berdasarkan heuristik.	HYBRID
<code>hive.exec.reducers.bytes.per.reducer</code>	Ukuran per peredam. Defaultnya adalah 256 MB. Jika ukuran input 1G, pekerjaan menggunakan 4 reduksi.	256000000
<code>hive.exec.reducers.max</code>	Jumlah maksimum reduksi.	256

Pengaturan	Deskripsi	Nilai default
<code>hive.exec.stagingdir</code>	Nama direktori yang menyimpan file sementara yang Hive buat di dalam lokasi tabel dan di lokasi direktori awal yang ditentukan dalam <code>hive.exec.scratchdir</code> properti.	<code>.hive-staging</code>
<code>hive.fetch.task.conversion</code>	Mengharapkan salah satu nilai berikut: NONE, MINIMAL, atau MORE. Hive dapat mengonversi kueri pilih menjadi satu FETCH tugas. Ini meminimalkan latensi.	MORE
<code>hive.groupby.position.alias</code>	Opsi yang menyebabkan Hive menggunakan alias posisi kolom dalam GROUP BY pernyataan.	FALSE
<code>hive.input.format</code>	Format input default. Atur ke <code>HiveInputFormat</code> jika Anda mengalami masalah dengan <code>CombineHiveInputFormat</code> .	<code>org.apache.hadoop.hive ql.io.CombineHiveInputFormat</code>
<code>hive.log.explain.output</code>	Opsi yang mengaktifkan penjelasan output diperpanjang untuk kueri apa pun di log Hive Anda.	FALSE
<code>hive.log.level</code>	Tingkat penebangan sarang.	INFO

Pengaturan	Deskripsi	Nilai default
<code>hive.mapred.reduce.tasks.speculative.execution</code>	Opsi yang mengaktifkan peluncuran spekulatif untuk reduksi. Hanya didukung dengan Amazon EMR 6.10.x dan yang lebih rendah.	TRUE
<code>hive.max-task-containers</code>	Jumlah maksimum kontainer bersamaan. Memori mapper yang dikonfigurasi dikalikan dengan nilai ini untuk menentukan memori yang tersedia yang membagi komputasi dan penggunaan preemption tugas.	1000
<code>hive.merge.mapfiles</code>	Opsi yang menyebabkan file kecil bergabung di akhir pekerjaan khusus peta.	TRUE
<code>hive.merge.size.per.task</code>	Ukuran file gabungan di akhir pekerjaan.	256000000
<code>hive.merge.tezfiles</code>	Opsi yang mengaktifkan penggabungan file kecil di akhir Tez DAG.	FALSE
<code>hive.metastore.client.factory.class</code>	Nama kelas pabrik yang menghasilkan objek yang mengimplementasikan <code>IMetaStoreClient</code> antarmuka.	<code>com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveClientFactory</code>

Pengaturan	Deskripsi	Nilai default
<code>hive.metastore.glue.catalogid</code>	Jika Katalog Data AWS Glue bertindak sebagai metastore tetapi berjalan di tempat yang berbeda Akun AWS dari pekerjaan, ID Akun AWS tempat pekerjaan berjalan.	NULL
<code>hive.metastore.uris</code>	URI hemat yang digunakan klien metastore untuk terhubung ke metastore jarak jauh.	NULL
<code>hive.optimize.ppd</code>	Opsi yang menyalakan predikat pushdown.	TRUE
<code>hive.optimize.ppd.storage</code>	Opsi yang mengaktifkan predikat pushdown ke penanganan penyimpanan.	TRUE
<code>hive.orderby.position.alias</code>	Opsi yang menyebabkan Hive menggunakan alias posisi kolom dalam ORDER BY pernyataan.	TRUE
<code>hive.prewarm.enabled</code>	Opsi yang menyalakan wadah prewarm untuk Tez.	FALSE
<code>hive.prewarm.numcontainers</code>	Jumlah wadah untuk pra-hangat untuk Tez.	10
<code>hive.stats.autogather</code>	Opsi yang menyebabkan Hive mengumpulkan statistik dasar secara otomatis selama INSERT OVERWRITE perintah.	TRUE

Pengaturan	Deskripsi	Nilai default
<code>hive.stats.fetch.column.stats</code>	Opsi yang mematikan pengambilan statistik kolom dari metastore. Pengambilan statistik kolom bisa mahal ketika jumlah kolom tinggi.	FALSE
<code>hive.stats.gather.num.threads</code>	Jumlah utas yang digunakan <code>partialscan</code> dan <code>noscan</code> menganalisis perintah untuk tabel yang dipartisi. Ini hanya berlaku untuk format file yang menerapkan <code>StatsProvidingRecordReader</code> (seperti ORC).	10
<code>hive.strict.checks.cartesian.product</code>	Opsi yang mengaktifkan pemeriksaan gabungan Cartesian yang ketat. Pemeriksaan ini melarang produk Cartesian (gabungan silang).	FALSE
<code>hive.strict.checks.type.safety</code>	Opsi yang mengaktifkan pemeriksaan keamanan tipe ketat dan mematikan perbandingan <code>bigint</code> dengan keduanya <code>string</code> dan <code>double</code> .	TRUE

Pengaturan	Deskripsi	Nilai default
<code>hive.support.quote d.identifiers</code>	Mengharapkan nilai NONE atau COLUMN. NONE menyiratkan hanya karakter alfanumerik dan garis bawah yang valid dalam pengidentifikasi. COLUMN menyiratkan nama kolom dapat berisi karakter apa pun.	COLUMN
<code>hive.tez.auto.redu cer.parallelism</code>	Opsi yang mengaktifkan fitur paralelisme peredam otomatis Tez. Hive masih memperkirakan ukuran data dan menetapkan perkiraan paralelisme. Tez mengambil sampel ukuran output dari simpul sumber dan menyesuaikan perkiraan saat runtime seperlunya.	TRUE
<code>hive.tez.container .size</code>	Jumlah memori yang digunakan per proses tugas Tez.	6144
<code>hive.tez.cpu.vcores</code>	Jumlah core yang digunakan untuk setiap tugas Tez.	2
<code>hive.tez.disk.size</code>	Ukuran disk untuk setiap wadah tugas.	20G
<code>hive.tez.input.for mat</code>	Format input untuk generasi split di Tez AM.	<code>org.apache.hadoop. hive ql.io.HiveInp utFormat</code>

Pengaturan	Deskripsi	Nilai default
<code>hive.tez.min.partition.factor</code>	Batas bawah reduksi yang ditentukan Tez saat Anda mengaktifkan paralelisme peredam otomatis.	0,25
<code>hive.vectorized.execution.enabled</code>	Opsi yang mengaktifkan mode vektor eksekusi kueri.	TRUE
<code>hive.vectorized.execution.reduce.enabled</code>	Opsi yang mengaktifkan mode vektor dari sisi pengurangan eksekusi kueri.	TRUE
<code>javax.jdo.option.ConnectionDriverName</code>	Nama kelas driver untuk metastore JDBC.	<code>org.apache.derby.jdbc.EmbeddedDriver</code>
<code>javax.jdo.option.ConnectionPassword</code>	Kata sandi yang terkait dengan database metastore.	NULL
<code>javax.jdo.option.ConnectionURL</code>	JDBC menghubungkan string untuk metastore JDBC.	<code>jdbc:derby;;databaseName=metastore_db;create=true</code>
<code>javax.jdo.option.ConnectionUserName</code>	Nama pengguna yang terkait dengan database metastore.	NULL
<code>mapreduce.input.fileinputformat.split.maxsize</code>	Ukuran maksimum split selama komputasi split saat format input Anda. <code>org.apache.hadoop.hive ql.io.Combine HiveInputFormat</code> Nilai 0 menunjukkan tidak ada batas.	0
<code>tez.am.dag.cleanup.on.completion</code>	Opsi yang mengaktifkan pembersihan data acak saat DAG selesai.	TRUE

Pengaturan	Deskripsi	Nilai default
<code>tez.am.emr-serverless.launch.env.[KEY]</code>	Pilihan untuk mengatur variabel KEY lingkungan dalam proses Tez AM. Untuk Tez AM, nilai ini mengesampingkan nilainya. <code>hive.emr-serverless.launch.env.[KEY]</code>	
<code>tez.am.log.level</code>	Level logging root yang diberikan EMR Serverless ke aplikasi utama Tez.	INFO
<code>tez.am.sleep.time.before.exit.millis</code>	EMR Tanpa Server harus mendorong peristiwa ATS setelah periode waktu ini setelah permintaan shutdown AM.	0
<code>tez.am.speculation.enabled</code>	Opsi yang menyebabkan peluncuran spekulatif tugas yang lebih lambat. Ini dapat membantu mengurangi latensi pekerjaan ketika beberapa tugas berjalan lebih lambat karena mesin yang buruk atau lambat. Hanya didukung dengan Amazon EMR 6.10.x dan yang lebih rendah.	FALSE
<code>tez.am.task.max.failed.attempts</code>	Jumlah maksimum upaya yang dapat gagal untuk tugas tertentu sebelum tugas gagal. Nomor ini tidak menghitung upaya yang dihentikan secara manual.	3

Pengaturan	Deskripsi	Nilai default
<code>tez.am.vertex.cleanup.height</code>	Jarak di mana, jika semua simpul dependen selesai, Tez AM akan menghapus data vertex shuffle. Fitur ini dimatikan ketika nilainya 0. Amazon EMR versi 6.8.0 dan yang lebih baru mendukung fitur ini.	0
<code>tez.client.asynchronous-stop</code>	Opsi yang menyebabkan EMR Tanpa Server mendorong peristiwa ATS sebelum mengakhiri driver Hive.	FALSE
<code>tez.grouping.max-size</code>	Batas ukuran atas (dalam byte) dari pemisahan yang dikelompokkan. Batas ini mencegah perpecahan yang terlalu besar.	1073741824
<code>tez.grouping.min-size</code>	Batas ukuran yang lebih rendah (dalam byte) dari pemisahan yang dikelompokkan. Batas ini mencegah terlalu banyak perpecahan kecil.	16777216
<code>tez.runtime.io.sort.mb</code>	Ukuran buffer lunak saat Tez mengurutkan output diurutkan.	Nilai optimal dihitung berdasarkan memori tugas Tez
<code>tez.runtime.unordered.output.buffer.size-mb</code>	Ukuran buffer yang akan digunakan jika Tez tidak menulis langsung ke disk.	Nilai optimal dihitung berdasarkan memori tugas Tez

Pengaturan	Deskripsi	Nilai default
<code>tez.shuffle-vertex-manager.max-src-fraction</code>	Fraksi tugas sumber yang harus diselesaikan sebelum EMR Tanpa Server menjadwalkan semua tugas untuk simpul saat ini (dalam kasus koneksi). ScatterGather Jumlah tugas yang siap untuk penjadwalan pada skala simpul saat ini secara linier antara <code>min-fraction</code> dan <code>max-fraction</code> . Ini default nilai default atau <code>tez.shuffle-vertex-manager.min-src-fraction</code> , mana yang lebih besar.	0,75
<code>tez.shuffle-vertex-manager.min-src-fraction</code>	Fraksi tugas sumber yang harus diselesaikan sebelum EMR Serverless menjadwalkan tugas untuk simpul saat ini (dalam kasus koneksi). ScatterGather	0,25
<code>tez.task.emr-serverless.launch.env.[KEY]</code>	Opsi untuk mengatur variabel KEY lingkungan dalam proses tugas Tez. Untuk tugas Tez, nilai ini mengesampingkan nilainya. <code>hive.emr-serverless.launch.env.[KEY]</code>	
<code>tez.task.log.level</code>	Level logging root yang diberikan EMR Serverless ke tugas Tez.	INFO

Pengaturan	Deskripsi	Nilai default
tez.yarn.ats.event .flush.timeout.millis	Jumlah maksimum waktu AM harus menunggu acara dibilas sebelum dimatikan.	300000

Contoh pekerjaan sarang

Contoh kode berikut menunjukkan cara menjalankan query Hive dengan StartJobRun API.

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "hive": {
      "query": "s3://amzn-s3-demo-bucket/emr-serverless-hive/query/hive-
query.q1",
      "parameters": "--hiveconf hive.log.explain.output=false"
    }
  }' \
  --configuration-overrides '{
    "applicationConfiguration": [{
      "classification": "hive-site",
      "properties": {
        "hive.exec.scratchdir": "s3://amzn-s3-demo-bucket/emr-serverless-hive/
hive/scratch",
        "hive.metastore.warehouse.dir": "s3://amzn-s3-demo-bucket/emr-
serverless-hive/hive/warehouse",
        "hive.driver.cores": "2",
        "hive.driver.memory": "4g",
        "hive.tez.container.size": "4096",
        "hive.tez.cpu.vcores": "1"
      }
    }
  ]
}'
```

Anda dapat menemukan contoh tambahan tentang cara menjalankan pekerjaan Hive di repositori [EMR Sampel Tanpa Server](#) GitHub .

Ketahanan Pekerjaan EMR Tanpa Server

EMR Tanpa Server rilis 7.1.0 dan yang lebih tinggi menyertakan dukungan untuk ketahanan pekerjaan, sehingga secara otomatis mencoba ulang pekerjaan yang gagal tanpa masukan manual dari Anda. Manfaat lain dari ketahanan kerja adalah bahwa EMR Serverless memindahkan pekerjaan ke Availability Zone (AZ) yang berbeda jika AZ mengalami masalah apa pun.

Untuk mengaktifkan ketahanan pekerjaan untuk suatu pekerjaan, tetapkan kebijakan coba lagi untuk pekerjaan Anda. Kebijakan coba lagi memastikan bahwa EMR Tanpa Server secara otomatis memulai ulang pekerjaan jika gagal pada titik mana pun. Kebijakan coba lagi didukung untuk pekerjaan batch dan streaming, jadi sesuaikan ketahanan pekerjaan sesuai dengan kasus penggunaan Anda. Tabel berikut membandingkan perilaku dan perbedaan ketahanan kerja di seluruh pekerjaan batch dan streaming.

	Tugas Batch	Lowongan kerja streaming
Perilaku default	Tidak menjalankan kembali pekerjaan.	Selalu mencoba menjalankan pekerjaan karena aplikasi membuat pos pemeriksaan saat menjalankan pekerjaan.
Poin coba lagi	Pekerjaan batch tidak memiliki pos pemeriksaan, jadi EMR Serverless selalu menjalankan kembali pekerjaan dari awal.	Pekerjaan streaming mendukung pos pemeriksaan, jadi konfigurasi kueri streaming untuk menyimpan status runtime dan melanjutkan ke lokasi pos pemeriksaan di Amazon S3. EMR Tanpa Server melanjutkan pekerjaan yang dijalankan dari pos pemeriksaan. Untuk informasi selengkapnya, lihat Memulihkan dari kegagalan dengan Checkpointing di dokumentasi Apache Spark .
Maksimal upaya coba lagi	Memungkinkan maksimal 10 percobaan ulang.	Pekerjaan streaming memiliki kontrol pencegahan thrash

	Tugas Batch	Lowongan kerja streaming
		<p>bawaan, sehingga aplikasi berhenti mencoba lagi pekerjaan jika terus gagal setelah satu jam. Jumlah default percobaan ulang dalam satu jam adalah lima upaya. Anda dapat mengonfigurasi jumlah percobaan ulang ini menjadi antara 1 atau 10. Anda tidak dapat menyesuaikan jumlah upaya maksimum. Nilai 1 menunjukkan tidak ada percobaan ulang.</p>

Ketika EMR Serverless mencoba menjalankan kembali pekerjaan, ia juga mengindeks pekerjaan dengan nomor percobaan, jadi lacak siklus hidup pekerjaan di seluruh upayanya.

Gunakan operasi API EMR Tanpa Server atau AWS CLI untuk mengubah ketahanan pekerjaan atau mengakses informasi yang terkait dengan ketahanan pekerjaan. Untuk informasi selengkapnya, lihat panduan [EMR Serverless API](#).

Secara default, EMR Tanpa Server tidak menjalankan kembali pekerjaan batch. Untuk mengaktifkan percobaan ulang untuk pekerjaan batch, konfigurasi `maxAttempts` parameter saat memulai pekerjaan batch. `maxAttempts` parameter ini hanya berlaku untuk pekerjaan batch. Defaultnya adalah 1, yang berarti tidak menjalankan kembali pekerjaan. Nilai yang diterima adalah 1 hingga 10, inklusif.

Contoh berikut menunjukkan bagaimana menentukan jumlah maksimal 10 upaya ketika memulai pekerjaan berjalan.

```
aws emr-serverless start-job-run
  --application-id <APPLICATION_ID> \
  --execution-role-arn <JOB_EXECUTION_ROLE> \
  --mode 'BATCH' \
  --retry-policy '{
    "maxAttempts": 10
  }' \
```

```
--job-driver '{
  "sparkSubmit": {
    "entryPoint": "/usr/lib/spark/examples/jars/spark-examples-does-not-
exist.jar",
    "entryPointArguments": ["1"],
    "sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi"
  }
}'
```

EMR Tanpa Server tanpa batas waktu mencoba lagi pekerjaan streaming jika gagal. Untuk mencegah meronta-ronta karena kegagalan berulang yang tidak dapat dipulihkan, gunakan `maxFailedAttemptsPerHour` untuk mengonfigurasi kontrol pencegahan thrash untuk streaming percobaan ulang pekerjaan. Parameter ini memungkinkan Anda menentukan jumlah maksimum upaya gagal yang diizinkan dengan satu jam sebelum EMR Tanpa Server berhenti mencoba lagi. Defaultnya adalah lima. Nilai yang diterima adalah 1 hingga 10, inklusif.

```
aws emr-serverless start-job-run
--application-id <APPLICATION_ID> \
--execution-role-arn <JOB_EXECUTION_ROLE> \
--mode 'STREAMING' \
--retry-policy '{
  "maxFailedAttemptsPerHour": 7
}' \
--job-driver '{
  "sparkSubmit": {
    "entryPoint": "/usr/lib/spark/examples/jars/spark-examples-does-not-
exist.jar",
    "entryPointArguments": ["1"],
    "sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi"
  }
}'
```

Anda juga dapat menggunakan operasi API yang menjalankan pekerjaan lainnya untuk mendapatkan informasi tentang pekerjaan. Misalnya, gunakan `attempt` parameter dengan `GetJobRun` operasi untuk mendapatkan detail tentang upaya pekerjaan tertentu. Jika Anda tidak menyertakan `attempt` parameter, operasi mengembalikan informasi tentang upaya terbaru.

```
aws emr-serverless get-job-run \
--job-run-id <job-run-id> \
--application-id <application-id> \
--attempt 1
```

ListJobRunAttempts Operasi mengembalikan informasi tentang semua upaya yang terkait dengan menjalankan pekerjaan.

```
aws emr-serverless list-job-run-attempts \  
  --application-id application-id \  
  --job-run-id job-run-id
```

GetDashboardForJobRun Operasi membuat dan mengembalikan URL yang digunakan untuk mengakses aplikasi UIs untuk menjalankan pekerjaan. attemptParameter memungkinkan Anda mendapatkan URL untuk upaya tertentu. Jika Anda tidak menyertakan attempt parameter, operasi mengembalikan informasi tentang upaya terbaru.

```
aws emr-serverless get-dashboard-for-job-run \  
  --application-id application-id \  
  --job-run-id job-run-id \  
  --attempt 1
```

Memantau pekerjaan dengan kebijakan coba lagi

Dukungan ketahanan Job juga menambahkan acara baru EMR Serverless job run retry. EMR Tanpa Server menerbitkan acara ini pada setiap percobaan ulang pekerjaan. Anda dapat menggunakan notifikasi ini untuk melacak percobaan ulang pekerjaan. Untuk informasi selengkapnya tentang acara, lihat [EventBridge acara Amazon](#).

Logging dengan kebijakan coba lagi

Setiap kali EMR Serverless mencoba kembali pekerjaan, upaya tersebut menghasilkan kumpulan lognya sendiri. Untuk memastikan bahwa EMR Tanpa Server dapat berhasil mengirimkan log ini ke Amazon S3 dan Amazon tanpa CloudWatch menimpa apa pun, EMR Tanpa Server menambahkan awalan ke format jalur log S3 dan nama aliran log untuk menyertakan nomor percobaan pekerjaan. CloudWatch

Berikut ini adalah contoh seperti apa formatnya.

```
'/applications/<applicationId>/jobs/<jobId>/attempts/<attemptNumber>/'.
```

Format ini memastikan EMR Tanpa Server menerbitkan semua log untuk setiap upaya pekerjaan ke lokasi yang ditunjuk sendiri di Amazon S3 dan. CloudWatch Untuk detail selengkapnya, lihat [Menyimpan log](#).

Note

EMR Tanpa Server hanya menggunakan format awalan ini dengan semua pekerjaan streaming dan pekerjaan batch apa pun yang telah mencoba lagi diaktifkan.

Konfigurasi metastore untuk EMR Tanpa Server

Metastore Hive adalah lokasi terpusat yang menyimpan informasi struktural tentang tabel Anda, termasuk skema, nama partisi, dan tipe data. Dengan EMR Tanpa Server, pertahankan metadata tabel ini dalam metastore yang memiliki akses ke pekerjaan Anda.

Anda memiliki dua opsi untuk metastore Hive:

- Katalog Data AWS Glue
- Metastore Apache Hive eksternal

Menggunakan Katalog Data AWS Glue sebagai metastore

Anda dapat mengonfigurasi pekerjaan Spark dan Hive Anda untuk menggunakan Katalog Data AWS Glue sebagai metastore. Kami merekomendasikan konfigurasi ini ketika Anda memerlukan metastore persisten atau metastore yang dibagikan oleh berbagai aplikasi, layanan, atau Akun AWS. Untuk informasi lebih lanjut tentang Katalog Data, lihat [Mengisi Katalog Data AWS Glue](#). Untuk informasi tentang harga AWS Glue, lihat [harga AWS Glue](#).

Anda dapat mengonfigurasi pekerjaan EMR Tanpa Server Anda untuk menggunakan Katalog Data AWS Glue baik yang Akun AWS sama dengan aplikasi Anda, atau yang berbeda. Akun AWS

Konfigurasi Katalog Data AWS Glue

Untuk mengkonfigurasi Katalog Data, pilih jenis aplikasi EMR Tanpa Server yang ingin Anda gunakan.

Spark

Saat Anda menggunakan EMR Studio untuk menjalankan pekerjaan Anda dengan aplikasi EMR Serverless Spark, Katalog Data AWS Glue adalah metastore default.

Saat Anda menggunakan SDK atau AWS CLI, setelah `spark.hadoop.hive.metastore.client.factory.class` konfigurasi ke `com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveClientFactory` dalam `sparkSubmit` parameter menjalankan pekerjaan Anda. Contoh berikut menunjukkan cara mengkonfigurasi Katalog Data dengan AWS CLI.

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "s3://amzn-s3-demo-bucket/code/pyspark/
extreme_weather.py",
      "sparkSubmitParameters": "--conf
spark.hadoop.hive.metastore.client.factory.class=com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveClientFactory
--conf spark.driver.cores=1 --conf spark.driver.memory=3g --conf
spark.executor.cores=4 --conf spark.executor.memory=3g"
    }
  }'
```

Atau, Anda dapat mengatur konfigurasi ini ketika Anda membuat yang baru `SparkSession` dalam kode Spark Anda.

```
from pyspark.sql import SparkSession

spark = (
    SparkSession.builder.appName("SparkSQL")
    .config(
        "spark.hadoop.hive.metastore.client.factory.class",
        "com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveClientFactory",
    )
    .enableHiveSupport()
    .getOrCreate()
)

# we can query tables with SparkSQL
spark.sql("SHOW TABLES").show()

# we can also them with native Spark
print(spark.catalog.listTables())
```

Hive

Untuk aplikasi EMR Serverless Hive, Katalog Data adalah metastore default. Artinya, ketika Anda menjalankan pekerjaan pada aplikasi EMR Serverless Hive, Hive mencatat informasi metastore dalam Katalog Data sama dengan aplikasi Anda. Akun AWS Anda tidak memerlukan virtual private cloud (VPC) untuk menggunakan Katalog Data sebagai metastore Anda.

Untuk mengakses tabel metastore Hive, tambahkan kebijakan Glue yang diperlukan yang diuraikan dalam [Menyiapkan Izin IAM](#) untuk AWS Glue. AWS

Konfigurasi akses lintas akun untuk EMR Serverless AWS dan Glue Data Catalog

Untuk mengatur akses lintas akun untuk EMR Tanpa Server, pertama-tama masuk ke yang berikut: Akun AWS

- AccountA— Akun AWS Tempat Anda telah membuat aplikasi EMR Tanpa Server.
 - AccountB— Sebuah Akun AWS yang berisi Katalog Data AWS Glue yang Anda ingin pekerjaan EMR Tanpa Server Anda berjalan untuk mengakses.
1. Pastikan administrator atau identitas resmi lainnya AccountB melampirkan kebijakan sumber daya ke Katalog Data diAccountB. Kebijakan ini memberikan izin lintas akun AccountA tertentu untuk melakukan operasi pada sumber daya dalam katalog. AccountB

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:GetDatabase",
        "glue:CreateDatabase",
        "glue:GetDataBases",
        "glue:CreateTable",
        "glue:GetTable",
        "glue:UpdateTable",
        "glue>DeleteTable",
        "glue:GetTables",
```

```

    "glue:GetPartition",
    "glue:GetPartitions",
    "glue:CreatePartition",
    "glue:BatchCreatePartition",
    "glue:GetUserDefinedFunctions"
  ],
  "Resource": [
    "arn:aws:glue:*:123456789012:catalog"
  ],
  "Sid": "AllowGLUEGetdatabase"
}
]
}

```

2. Tambahkan kebijakan IAM ke peran runtime pekerjaan EMR Tanpa Server sehingga peran dapat mengakses sumber daya Katalog Data AccountA di AccountB

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:GetDatabase",
        "glue:CreateDatabase",
        "glue:GetDataBases",
        "glue:CreateTable",
        "glue:GetTable",
        "glue:UpdateTable",
        "glue>DeleteTable",
        "glue:GetTables",
        "glue:GetPartition",
        "glue:GetPartitions",
        "glue:CreatePartition",
        "glue:BatchCreatePartition",
        "glue:GetUserDefinedFunctions"
      ],
      "Resource": [
        "arn:aws:glue:*:123456789012:catalog"
      ],
      "Sid": "AllowGLUEGetdatabase"
    }
  ]
}

```

```

    }
  ]
}

```

3. Mulai menjalankan pekerjaan Anda. Langkah ini sedikit berbeda tergantung pada jenis aplikasi AccountA EMR Serverless.

Spark

Lulus `spark.hadoop.hive.metastore.glue.catalogid` properti di `sparkSubmitParameters` seperti yang ditunjukkan pada contoh berikut. Ganti *AccountB-catalog-id* dengan ID Katalog Data di AccountB.

```

aws emr-serverless start-job-run \
--application-id "application-id" \
--execution-role-arn "job-role-arn" \
--job-driver '{
  "sparkSubmit": {
    "entryPoint": "s3://amzn-s3-demo-bucket/scripts/test.py",
    "sparkSubmitParameters": "--conf
spark.hadoop.hive.metastore.client.factory.class=com.amazonaws.glue.catalog.metastore.A
--conf spark.hadoop.hive.metastore.glue.catalogid=AccountB-catalog-
id --conf spark.executor.cores=1 --conf spark.executor.memory=1g
--conf spark.driver.cores=1 --conf spark.driver.memory=1g --conf
spark.executor.instances=1"
  }
}' \
--configuration-overrides '{
  "monitoringConfiguration": {
    "s3MonitoringConfiguration": {
      "logUri": "s3://amzn-s3-demo-bucket/logs/"
    }
  }
}'

```

Hive

Tetapkan `hive.metastore.glue.catalogid` properti dalam `hive-site` klasifikasi seperti yang ditunjukkan pada contoh berikut. Ganti *AccountB-catalog-id* dengan ID Katalog Data di AccountB.

```

aws emr-serverless start-job-run \

```

```

--application-id "application-id" \
--execution-role-arn "job-role-arn" \
--job-driver '{
  "hive": {
    "query": "s3://amzn-s3-demo-bucket/hive/scripts/create_table.sql",
    "parameters": "--hiveconf hive.exec.scratchdir=s3://amzn-s3-demo-bucket/
hive/scratch --hiveconf hive.metastore.warehouse.dir=s3://amzn-s3-demo-bucket/
hive/warehouse"
  }
}' \
--configuration-overrides '{
  "applicationConfiguration": [{
    "classification": "hive-site",
    "properties": {
      "hive.metastore.glue.catalogid": "AccountB-catalog-id"
    }
  ]
}'

```

Pertimbangan saat menggunakan Katalog Data AWS Glue

Anda dapat menambahkan JAR tambahan dengan ADD JAR skrip Hive Anda. Untuk pertimbangan tambahan, lihat [Pertimbangan saat menggunakan AWS Glue Data Catalog](#).

Menggunakan metastore Hive eksternal

Anda dapat mengonfigurasi pekerjaan EMR Serverless Spark dan Hive untuk terhubung ke metastore Hive eksternal, seperti Amazon Aurora atau Amazon RDS for MySQL. Bagian ini menjelaskan cara menyiapkan metastore Amazon RDS Hive, mengonfigurasi VPC, dan mengonfigurasi pekerjaan EMR Tanpa Server untuk menggunakan metastore eksternal.

Buat metastore Hive eksternal

1. Buat Amazon Virtual Private Cloud (Amazon VPC) dengan subnet pribadi dengan mengikuti petunjuk di [Buat VPC](#).
2. Buat aplikasi EMR Tanpa Server Anda dengan VPC Amazon baru dan subnet pribadi Anda. Ketika Anda mengkonfigurasi aplikasi EMR Serverless Anda dengan VPC, pertama-tama menyediakan sebuah elastic network interface untuk setiap subnet yang Anda tentukan. Kemudian melampirkan grup keamanan yang Anda tentukan ke antarmuka jaringan itu. Ini memberikan kontrol akses aplikasi Anda. Untuk detail selengkapnya tentang cara mengatur VPC

- Anda, lihat. [Mengkonfigurasi akses VPC untuk aplikasi EMR Tanpa Server untuk terhubung ke data](#)
3. Buat database MySQL atau Aurora PostgreSQL di subnet pribadi di Amazon VPC Anda. Untuk informasi tentang cara membuat database Amazon RDS, lihat [Membuat instans Amazon RDS DB](#).
 4. [Ubah grup keamanan database MySQL atau Aurora Anda untuk mengizinkan koneksi JDBC dari grup keamanan EMR Tanpa Server Anda dengan mengikuti langkah-langkah dalam Memodifikasi instans Amazon RDS DB](#). Tambahkan aturan untuk lalu lintas masuk ke grup keamanan RDS dari salah satu grup keamanan EMR Tanpa Server Anda.

Tipe	Protokol	Rentang port	Sumber
Semua TCP	TCP	3306	emr-serverless-security-group

Konfigurasi opsi Spark

Menggunakan JDBC

Untuk mengonfigurasi aplikasi EMR Serverless Spark agar terhubung ke metastore Hive berdasarkan Amazon RDS for MySQL atau Amazon Aurora MySQL, gunakan koneksi JDBC. Lewati `mariadb-connector-java.jar` dengan `--jars` dalam `spark-submit` parameter lari pekerjaan Anda.

```
aws emr-serverless start-job-run \
  --application-id "application-id" \
  --execution-role-arn "job-role-arn" \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "s3://amzn-s3-demo-bucket/scripts/spark-jdbc.py",
      "sparkSubmitParameters": "--jars s3://amzn-s3-demo-bucket/mariadb-connector-java.jar
      --conf
      spark.hadoop.javax.jdo.option.ConnectionDriverName=org.mariadb.jdbc.Driver
      --conf spark.hadoop.javax.jdo.option.ConnectionUserName=<connection-user-name>
      --conf spark.hadoop.javax.jdo.option.ConnectionPassword=<connection-password>
```

```

--conf spark.hadoop.java.jdbc.option.ConnectionURL=<JDBC-Connection-
string>
--conf spark.driver.cores=2
--conf spark.executor.memory=10G
--conf spark.driver.memory=6G
--conf spark.executor.cores=4"
    }
}' \
--configuration-overrides '{
    "monitoringConfiguration": {
        "s3MonitoringConfiguration": {
            "logUri": "s3://amzn-s3-demo-bucket/spark/logs/"
        }
    }
}'
}'

```

Contoh kode berikut adalah skrip entrypoint Spark yang berinteraksi dengan metastore Hive di Amazon RDS.

```

from os.path import expanduser, join, abspath
from pyspark.sql import SparkSession
from pyspark.sql import Row
# warehouse_location points to the default location for managed databases and tables
warehouse_location = abspath('spark-warehouse')
spark = SparkSession \
    .builder \
    .config("spark.sql.warehouse.dir", warehouse_location) \
    .enableHiveSupport() \
    .getOrCreate()
spark.sql("SHOW DATABASES").show()
spark.sql("CREATE EXTERNAL TABLE `sampledb`.`sparknyctaxi`(`dispatching_base_num`
string, `pickup_datetime` string, `dropoff_datetime` string, `polocationid` bigint,
`dolocationid` bigint, `sr_flag` bigint) STORED AS PARQUET LOCATION 's3://<s3 prefix>/
nyctaxi_parquet/'")
spark.sql("SELECT count(*) FROM sampledb.sparknyctaxi").show()
spark.stop()

```

Menggunakan layanan penghematan

Anda dapat mengonfigurasi aplikasi EMR Serverless Hive Anda untuk terhubung ke metastore Hive berdasarkan Amazon RDS for MySQL atau Amazon Aurora MySQL instance. Untuk melakukan ini, jalankan server penghematan pada simpul utama cluster EMR Amazon yang ada. Opsi ini sangat

ideal jika Anda sudah memiliki cluster EMR Amazon dengan server hemat yang ingin Anda gunakan untuk menyederhanakan konfigurasi pekerjaan EMR Tanpa Server Anda.

```
aws emr-serverless start-job-run \
  --application-id "application-id" \
  --execution-role-arn "job-role-arn" \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "s3://amzn-s3-demo-bucket/thriftscript.py",
      "sparkSubmitParameters": "--jars s3://amzn-s3-demo-bucket/mariadb-
connector-java.jar
      --conf spark.driver.cores=2
      --conf spark.executor.memory=10G
      --conf spark.driver.memory=6G
      --conf spark.executor.cores=4"
    }
  }' \
  --configuration-overrides '{
    "monitoringConfiguration": {
      "s3MonitoringConfiguration": {
        "logUri": "s3://amzn-s3-demo-bucket/spark/logs/"
      }
    }
  }'
```

Contoh kode berikut adalah entrypoint script (`thriftscript.py`) yang menggunakan protokol hemat untuk terhubung ke metastore Hive. Perhatikan bahwa `hive.metastore.uris` properti perlu disetel untuk membaca dari metastore Hive eksternal.

```
from os.path import expanduser, join, abspath
from pyspark.sql import SparkSession
from pyspark.sql import Row
# warehouse_location points to the default location for managed databases and tables
warehouse_location = abspath('spark-warehouse')
spark = SparkSession \
    .builder \
    .config("spark.sql.warehouse.dir", warehouse_location) \
    .config("hive.metastore.uris", "thrift://thrift-server-host:thrift-server-port") \
    .enableHiveSupport() \
    .getOrCreate()
spark.sql("SHOW DATABASES").show()
```

```
spark.sql("CREATE EXTERNAL TABLE sampledb.`sparknyctaxi`(`dispatching_base_num`  
  string, `pickup_datetime` string, `dropoff_datetime` string, `pulocationid` bigint,  
  `dolocationid` bigint, `sr_flag` bigint) STORED AS PARQUET LOCATION 's3://<s3 prefix>/  
nyctaxi_parquet/'")  
spark.sql("SELECT * FROM sampledb.sparknyctaxi").show()  
spark.stop()
```

Konfigurasi opsi Hive

Menggunakan JDBC

Jika Anda ingin menentukan lokasi database Hive eksternal pada instans Amazon RDS MySQL atau Amazon Aurora, Anda dapat mengganti konfigurasi metastore default.

Note

Di Hive, Anda dapat melakukan beberapa penulisan ke tabel metastore secara bersamaan. Jika Anda berbagi informasi metastore antara dua pekerjaan, pastikan Anda tidak menulis ke tabel metastore yang sama secara bersamaan kecuali Anda menulis ke partisi yang berbeda dari tabel metastore yang sama.

Atur konfigurasi berikut dalam `hive-site` klasifikasi untuk mengaktifkan metastore Hive eksternal.

```
{  
  "classification": "hive-site",  
  "properties": {  
    "hive.metastore.client.factory.class":  
"org.apache.hadoop.hive.q1.metadata.SessionHiveMetaStoreClientFactory",  
    "javax.jdo.option.ConnectionDriverName": "org.mariadb.jdbc.Driver",  
    "javax.jdo.option.ConnectionURL": "jdbc:mysql://db-host:db-port/db-name",  
    "javax.jdo.option.ConnectionUserName": "username",  
    "javax.jdo.option.ConnectionPassword": "password"  
  }  
}
```

Menggunakan server penghematan

Anda dapat mengonfigurasi aplikasi EMR Serverless Hive untuk terhubung ke metastore Hive berdasarkan Amazon RDS for MySQL atau Amazon Aurora MySQL Instance. Untuk melakukan ini, jalankan server barang bekas di simpul utama cluster EMR Amazon yang ada. Opsi ini sangat ideal

jika Anda sudah memiliki cluster EMR Amazon yang menjalankan server barang bekas dan Anda ingin menggunakan konfigurasi pekerjaan EMR Tanpa Server Anda.

Atur konfigurasi berikut dalam hive-site klasifikasi sehingga EMR Serverless dapat mengakses metastore penghematan jarak jauh. Perhatikan bahwa Anda harus mengatur `hive.metastore.uris` properti untuk dibaca dari metastore Hive eksternal.

```
{
  "classification": "hive-site",
  "properties": {
    "hive.metastore.client.factory.class":
    "org.apache.hadoop.hive.ql.metadata.SessionHiveMetaStoreClientFactory",
    "hive.metastore.uris": "thrift://thrift-server-host:thrift-server-port"
  }
}
```

Bekerja dengan hirarki multi-katalog AWS Glue di EMR Serverless

Anda dapat mengonfigurasi aplikasi EMR Tanpa Server Anda untuk bekerja dengan hierarki multi-katalog Glue AWS . Contoh berikut menunjukkan cara menggunakan EMR-S Spark dengan hirarki multi-katalog AWS Glue.

Untuk mempelajari lebih lanjut tentang hierarki multi-katalog, lihat [Bekerja dengan hierarki multi-katalog di Katalog Data AWS Glue dengan Spark di Amazon EMR](#).

Menggunakan Redshift Managed Storage (RMS) dengan Iceberg dan Glue Data Catalog AWS

Berikut ini menunjukkan cara mengkonfigurasi Spark untuk integrasi dengan AWS Glue Data Catalog dengan Iceberg:

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "s3://amzn-s3-demo-bucket/myscript.py",
      "sparkSubmitParameters": "--conf spark.sql.catalog.nfgac_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.defaultCatalog=rms
        --conf
spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions"
    }
}'

```

Contoh kueri dari tabel dalam katalog, berikut integrasi:

```
SELECT * FROM my_rms_schema.my_table
```

Menggunakan Redshift Managed Storage (RMS) dengan Iceberg REST API dan Glue Data Catalog AWS

Berikut ini menunjukkan cara mengkonfigurasi Spark untuk bekerja dengan katalog Iceberg REST:

```

aws emr-serverless start-job-run \
--application-id application-id \
--execution-role-arn job-role-arn \
--job-driver '{
"sparkSubmit": {
"entryPoint": "s3://amzn-s3-demo-bucket/myscript.py",
"sparkSubmitParameters": "
--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"
}
}'

```

Contoh kueri dari tabel di katalog:

```
SELECT * FROM my_rms_schema.my_table
```

Pertimbangan saat menggunakan metastore eksternal

- Anda dapat mengonfigurasi database yang kompatibel dengan MariaDB JDBC sebagai metastore Anda. Contoh database ini adalah RDS untuk MariaDB, MySQL, dan Amazon Aurora.

- Metastores tidak diinisialisasi secara otomatis. [Jika metastore Anda tidak diinisialisasi dengan skema untuk versi Hive Anda, gunakan Hive Schema Tool.](#)
- EMR Tanpa Server tidak mendukung otentikasi Kerberos. Anda tidak dapat menggunakan server metastore barang bekas dengan otentikasi Kerberos dengan EMR Serverless Spark atau Hive jobs.
- Anda harus mengkonfigurasi akses VPC untuk menggunakan hierarki multi-katalog.

Mengakses data S3 di AWS akun lain dari EMR Tanpa Server

Anda dapat menjalankan pekerjaan Amazon EMR Tanpa Server dari satu AWS akun dan mengonfigurasinya untuk mengakses data di bucket Amazon S3 milik akun lain. AWS Halaman ini menjelaskan cara mengonfigurasi akses lintas akun ke S3 dari EMR Tanpa Server.

Pekerjaan yang berjalan di EMR Tanpa Server dapat menggunakan kebijakan bucket S3 atau peran yang diasumsikan untuk mengakses data di Amazon S3 dari akun lain. AWS

Prasyarat

Untuk mengatur akses lintas akun untuk Amazon EMR Tanpa Server, selesaikan tugas saat masuk ke dua akun: AWS

- **AccountA**— Ini adalah AWS akun tempat Anda membuat aplikasi Amazon EMR Tanpa Server. Sebelum Anda mengatur akses lintas akun, siapkan hal-hal berikut di akun ini:
 - Aplikasi Amazon EMR Tanpa Server tempat Anda ingin menjalankan pekerjaan.
 - Peran eksekusi pekerjaan yang memiliki izin yang diperlukan untuk menjalankan pekerjaan dalam aplikasi. Untuk informasi lebih lanjut, lihat [Peran runtime Job untuk Amazon EMR Tanpa Server](#).
- **AccountB**— Ini adalah AWS akun yang berisi ember S3 yang Anda inginkan untuk diakses oleh pekerjaan Amazon EMR Tanpa Server Anda.

Menggunakan kebijakan bucket S3 untuk mengakses data S3 lintas akun

Untuk mengakses bucket S3 in account B from account A, lampirkan kebijakan berikut ke bucket S3 di account B

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ExamplePermissions1",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:root"
      },
      "Action": [
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::my-bucket-name"
      ]
    },
    {
      "Sid": "ExamplePermissions2",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:root"
      },
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:DeleteObject"
      ],
      "Resource": [
        "arn:aws:s3:::my-bucket-name/*"
      ]
    }
  ]
}
```

Untuk informasi selengkapnya tentang akses lintas akun S3 dengan kebijakan bucket S3, lihat [Contoh 2: Pemilik bucket yang memberikan izin bucket lintas akun di](#) Panduan Pengguna Layanan Penyimpanan Sederhana Amazon.

Gunakan peran yang diasumsikan untuk mengakses data S3 lintas akun

Cara lain untuk mengatur akses lintas akun untuk Amazon EMR Tanpa Server adalah dengan tindakan `AssumeRole` dari (). AWS Security Token Service AWS STS AWS STS adalah layanan web global yang memungkinkan Anda meminta kredensial hak istimewa terbatas sementara untuk pengguna. Anda dapat melakukan panggilan API ke EMR Tanpa Server dan Amazon S3 dengan kredensial keamanan sementara yang Anda buat. `AssumeRole`

Langkah-langkah berikut menggambarkan cara menggunakan peran yang diasumsikan untuk mengakses data S3 lintas akun dari EMR Tanpa Server:

1. Buat bucket Amazon S3, *cross-account-bucket*, di AccountB. Untuk informasi selengkapnya, lihat [Membuat bucket](#) di Panduan Pengguna Layanan Penyimpanan Sederhana Amazon. Jika Anda ingin memiliki akses lintas akun ke DynamoDB, buat juga tabel DynamoDB di AccountB Untuk informasi selengkapnya, lihat [Membuat tabel DynamoDB di Panduan Pengembang Amazon DynamoDB](#).
2. Buat IAM role `Cross-Account-Role-B` dalam AccountB yang dapat mengakses *cross-account-bucket*.
 - a. Masuk ke Konsol Manajemen AWS dan buka konsol IAM di <https://console.aws.amazon.com/iam/>.
 - b. 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](#).
 - c. Buat kebijakan IAM yang menentukan izin untuk `Cross-Account-Role-B` untuk mengakses S3 bucket *cross-account-bucket*, seperti yang ditunjukkan pernyataan kebijakan berikut. Kemudian lampirkan kebijakan IAM ke `Cross-Account-Role-B`. Untuk informasi selengkapnya, lihat [Membuat kebijakan IAM](#) 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 Anda memerlukan akses DynamoDB, buat kebijakan IAM yang menentukan izin untuk mengakses tabel DynamoDB lintas akun. Kemudian lampirkan kebijakan IAM ke `Cross-Account-Role-B`. Untuk informasi selengkapnya, lihat [Amazon DynamoDB: Mengizinkan akses ke tabel tertentu](#) di Panduan Pengguna IAM.

Berikut ini adalah kebijakan untuk mengizinkan akses ke tabel DynamoDB. `CrossAccountTable` JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:*"
      ],
      "Resource": [
        "arn:aws:dynamodb:*:123456789012:table/CrossAccountTable"
      ],
      "Sid": "AllowDYNAMODB"
    }
  ]
}

```

3. Cara mengedit hubungan kepercayaan untuk peran `Cross-Account-Role-B`.
 - a. Untuk mengonfigurasi hubungan kepercayaan untuk peran tersebut, pilih tab Trust Relationships di konsol IAM untuk peran `Cross-Account-Role-B` yang Anda buat di Langkah 2.
 - b. Pilih Edit Hubungan Kepercayaan.
 - c. Tambahkan dokumen kebijakan berikut. Hal ini memungkinkan `Job-Execution-Role-A` `AccountA` untuk mengambil `Cross-Account-Role-B` peran.

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. Berikan Job-Execution-Role-A AWS STS AssumeRole izin AccountA untuk berasumsiCross-Account-Role-B.
 - a. Di konsol IAM untuk AWS akunAccountA, pilihJob-Execution-Role-A.
 - b. 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::123456789012:role/Cross-Account-Role-B"
      ],
      "Sid": "AllowSTSAssumerole"
    }
  ]
}
```

Contoh peran yang diasumsikan

Gunakan satu peran yang diasumsikan untuk mengakses semua sumber daya S3 di akun, atau dengan Amazon EMR 6.11 dan yang lebih tinggi, konfigurasi beberapa peran IAM untuk diasumsikan saat Anda mengakses bucket S3 lintas akun yang berbeda.

Topik

- [Akses sumber daya S3 dengan satu peran yang diasumsikan](#)
- [Akses sumber daya S3 dengan beberapa peran yang diasumsikan](#)

Akses sumber daya S3 dengan satu peran yang diasumsikan

Note

Saat Anda mengonfigurasi pekerjaan untuk menggunakan satu peran yang diasumsikan, semua sumber daya S3 di seluruh pekerjaan menggunakan peran tersebut, termasuk `entryPoint` skrip.

Jika Anda ingin menggunakan satu peran yang diasumsikan untuk mengakses semua sumber daya S3 di akun B, tentukan konfigurasi berikut:

1. Tentukan konfigurasi `fs.s3.customAWSCredentialsProvider` EMRFS ke `com.amazonaws.emr.AssumeRoleAWSCredentialsProvider`
2. Untuk Spark, gunakan `spark.emr-serverless.driverEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN` dan `spark.executorEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN` tentukan variabel lingkungan pada driver dan pelaksana.
3. Untuk Hive, gunakan `hive.emr-serverless.launch.env.ASSUME_ROLE_CREDENTIALS_ROLE_ARN`, `tez.am.emr-serverless.launch.env.ASSUME_ROLE_CREDENTIALS_ROLE_ARN`, dan `tez.task.emr-serverless.launch.env.ASSUME_ROLE_CREDENTIALS_ROLE_ARN` untuk menentukan variabel lingkungan pada driver Hive, aplikasi utama Tez, dan wadah tugas Tez.

Contoh berikut menunjukkan cara menggunakan peran yang diasumsikan untuk memulai pekerjaan EMR Tanpa Server dengan akses lintas akun.

Spark

Contoh berikut menunjukkan cara menggunakan peran yang diasumsikan untuk memulai pekerjaan EMR Serverless Spark yang dijalankan dengan akses lintas akun ke S3.

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "entrypoint_location",
      "entryPointArguments": [":argument_1:", ":argument_2:"],
      "sparkSubmitParameters": "--conf spark.executor.cores=4 --conf
spark.executor.memory=20g --conf spark.driver.cores=4 --conf spark.driver.memory=8g
--conf spark.executor.instances=1"
    }
  }' \
  --configuration-overrides '{
    "applicationConfiguration": [{
      "classification": "spark-defaults",
      "properties": {
        "spark.hadoop.fs.s3.customAWSCredentialsProvider":
"com.amazonaws.emr.AssumeRoleAWSCredentialsProvider",
        "spark.emr-serverless.driverEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN":
"arn:aws:iam::AccountB:role/Cross-Account-Role-B",
        "spark.executorEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN":
"arn:aws:iam::AccountB:role/Cross-Account-Role-B"
      }
    }]
  }'
```

Hive

Contoh berikut menunjukkan cara menggunakan peran yang diasumsikan untuk memulai pekerjaan EMR Serverless Hive yang dijalankan dengan akses lintas akun ke S3.

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "hive": {
      "query": "query_location",
      "parameters": "hive_parameters"
    }
  }'
```

```

    }
  } \
  --configuration-overrides '{
    "applicationConfiguration": [{
      "classification": "hive-site",
      "properties": {
        "fs.s3.customAWSCredentialsProvider":
"com.amazonaws.emr.serverless.credentialsprovider.AssumeRoleAWSCredentialsProvider",
        "hive.emr-serverless.launch.env.ASSUME_ROLE_CREDENTIALS_ROLE_ARN":
"arn:aws:iam::AccountB:role/Cross-Account-Role-B",
        "tez.am.emr-serverless.launch.env.ASSUME_ROLE_CREDENTIALS_ROLE_ARN":
"arn:aws:iam::AccountB:role/Cross-Account-Role-B",
        "tez.task.emr-
serverless.launch.env.ASSUME_ROLE_CREDENTIALS_ROLE_ARN":
"arn:aws:iam::AccountB:role/Cross-Account-Role-B"
      }
    }
  ]
}'

```

Akses sumber daya S3 dengan beberapa peran yang diasumsikan

Dengan rilis EMR Tanpa Server 6.11.0 dan yang lebih tinggi, konfigurasi beberapa peran IAM untuk diasumsikan saat Anda mengakses bucket lintas akun yang berbeda. Jika Anda ingin mengakses sumber daya S3 yang berbeda dengan peran yang diasumsikan berbeda di akun B, gunakan konfigurasi berikut saat Anda memulai pekerjaan:

1. Tentukan konfigurasi `fs.s3.customAWSCredentialsProvider` EMRFS ke `com.amazonaws.emr.serverless.credentialsprovider.BucketLevelAssumeRoleCredenti`
2. Tentukan konfigurasi EMRFS `fs.s3.bucketLevelAssumeRoleMapping` untuk menentukan pemetaan dari nama bucket S3 ke peran IAM di akun B untuk diasumsikan. Nilai harus dalam format `bucket1->role1;bucket2->role2`.

Misalnya, gunakan `arn:aws:iam::AccountB:role/Cross-Account-Role-B-1` untuk mengakses `bucketbucket1`, dan gunakan `arn:aws:iam::AccountB:role/Cross-Account-Role-B-2` untuk mengakses `bucketbucket2`. Contoh berikut menunjukkan cara memulai pekerjaan EMR Tanpa Server dengan akses lintas akun melalui beberapa peran yang diasumsikan.

Spark

Contoh berikut menunjukkan cara menggunakan beberapa peran yang diasumsikan untuk membuat EMR Serverless Spark job run.

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "entrypoint_location",
      "entryPointArguments": [":argument_1:", ":argument_2:"],
      "sparkSubmitParameters": "--conf spark.executor.cores=4 --conf
spark.executor.memory=20g --conf spark.driver.cores=4 --conf spark.driver.memory=8g
--conf spark.executor.instances=1"
    }
  }' \
  --configuration-overrides '{
    "applicationConfiguration": [{
      "classification": "spark-defaults",
      "properties": {
        "spark.hadoop.fs.s3.customAWSCredentialsProvider":
"com.amazonaws.emr.serverless.credentialsprovider.BucketLevelAssumeRoleCredentialsProvider"
        "spark.hadoop.fs.s3.bucketLevelAssumeRoleMapping":
"bucket1->arn:aws:iam::AccountB:role/Cross-Account-Role-B-1;bucket2->
arn:aws:iam::AccountB:role/Cross-Account-Role-B-2"
      }
    }]
  }'
```

Hive

Contoh berikut menunjukkan cara menggunakan beberapa peran yang diasumsikan untuk membuat pekerjaan EMR Serverless Hive berjalan.

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "hive": {
      "query": "query_location",
      "parameters": "hive_parameters"
    }
  }'
```

```
}' \  
--configuration-overrides '{  
  "applicationConfiguration": [{  
    "classification": "hive-site",  
    "properties": {  
      "fs.s3.customAWSCredentialsProvider":  
"com.amazonaws.emr.serverless.credentialsprovider.AssumeRoleAWSCredentialsProvider",  
      "fs.s3.bucketLevelAssumeRoleMapping": "bucket1-  
>arn:aws:iam:::role/Cross-Account-Role-B-1;bucket2-  
>arn:aws:iam:::role/Cross-Account-Role-B-2"  
    }  
  }]  
'
```

Memecahkan masalah kesalahan di EMR Tanpa Server

Gunakan informasi berikut untuk membantu mendiagnosis dan memperbaiki masalah umum yang terjadi saat bekerja dengan Amazon EMR Tanpa Server.

Topik

- [Kesalahan: Job gagal karena akun telah mencapai batas layanan pada vCPU maksimum yang dapat digunakan secara bersamaan.](#)
- [Kesalahan: Job gagal karena aplikasi telah melampaui pengaturan MaximumCapacity.](#)
- [Kesalahan: Job gagal karena Worker tidak dapat dialokasikan karena aplikasi telah melebihi MaximumCapacity.](#)
- [Kesalahan: Akses S3 ditolak. Silakan periksa izin akses S3 dari peran runtime pekerjaan pada sumber daya S3 yang diperlukan.](#)
- [Kesalahan: ModuleNotFoundError: Tidak ada modul bernama<module>. Silakan merujuk ke panduan pengguna tentang cara menggunakan pustaka python dengan EMR Tanpa Server.](#)
- [Kesalahan: Tidak dapat mengambil peran eksekusi <role name>karena tidak ada atau tidak diatur dengan hubungan kepercayaan yang diperlukan.](#)

Kesalahan: Job gagal karena akun telah mencapai batas layanan pada vCPU maksimum yang dapat digunakan secara bersamaan.

Kesalahan ini menunjukkan bahwa EMR Tanpa Server tidak dapat mengirimkan pekerjaan karena akun telah melebihi kapasitas maksimum. Tingkatkan kapasitas maksimum untuk akun. Periksa batas layanan Anda di kuota layanan [EMR Tanpa Server](#).

Kesalahan: Job gagal karena aplikasi telah melampaui pengaturan MaximumCapacity.

Kesalahan ini menunjukkan bahwa EMR Tanpa Server tidak dapat mengirimkan pekerjaan karena aplikasi telah melebihi kapasitas maksimum yang dikonfigurasi. Tingkatkan kapasitas maksimum untuk aplikasi.

Kesalahan: Job gagal karena Worker tidak dapat dialokasikan karena aplikasi telah melebihi MaximumCapacity.

Kesalahan ini menunjukkan bahwa pekerjaan tidak dapat diselesaikan. Pekerja tidak dapat dialokasikan karena aplikasi telah melampaui pengaturan MaximumCapacity.

Kesalahan: Akses S3 ditolak. Silakan periksa izin akses S3 dari peran runtime pekerjaan pada sumber daya S3 yang diperlukan.

Kesalahan ini menunjukkan bahwa pekerjaan Anda tidak memiliki akses ke sumber daya S3 Anda. Verifikasi bahwa peran runtime pekerjaan memiliki izin untuk mengakses sumber daya S3 yang perlu digunakan pekerjaan. Untuk mempelajari lebih lanjut tentang peran runtime, lihat. [Peran runtime Job untuk Amazon EMR Tanpa Server](#)

Kesalahan: ModuleNotFoundError: Tidak ada modul bernama<module>.
Silakan merujuk ke panduan pengguna tentang cara menggunakan pustaka python dengan EMR Tanpa Server.

Kesalahan ini menunjukkan bahwa modul Python tidak tersedia untuk pekerjaan Spark. Periksa apakah pustaka Python dependen tersedia untuk pekerjaan itu. Untuk informasi lebih lanjut tentang cara mengemas pustaka Python, lihat. [Menggunakan pustaka Python dengan EMR Tanpa Server](#)

Kesalahan: Tidak dapat mengambil peran eksekusi <role name>karena tidak ada atau tidak diatur dengan hubungan kepercayaan yang diperlukan.

Kesalahan ini menunjukkan bahwa peran runtime pekerjaan yang Anda tentukan untuk pekerjaan itu tidak ada, atau bahwa peran tersebut tidak memiliki hubungan kepercayaan untuk izin EMR Tanpa Server. Untuk memverifikasi bahwa peran IAM ada dan memvalidasi bahwa Anda telah menyiapkan kebijakan kepercayaan peran dengan benar, lihat instruksi di [Peran runtime Job untuk Amazon EMR Tanpa Server](#)

Mengaktifkan Alokasi Biaya Level Pekerjaan

Alokasi biaya tingkat pekerjaan memungkinkan atribusi penagihan granular untuk EMR Tanpa Server di tingkat pekerjaan individu, daripada menggabungkan semua biaya di tingkat aplikasi. Saat diaktifkan, Anda dapat memfilter dan melacak AWS biaya di Cost Explorer dan Laporan Biaya dan Penggunaan berdasarkan menjalankan pekerjaan tertentu IDs dan tag yang terkait dengan pekerjaan berjalan, memberikan visibilitas yang lebih baik ke biaya untuk pekerjaan yang dikirimkan.

Perilaku default

Alokasi biaya tingkat pekerjaan tidak diaktifkan secara default.

Cara mengaktifkan atau menonaktifkan fitur

Anda dapat mengonfigurasi alokasi biaya tingkat pekerjaan selama pembuatan aplikasi atau memperbaruinya untuk aplikasi yang ada.

Tentukan `jobLevelCostAllocation` parameter saat membuat aplikasi baru:

```
# Enable job-level cost allocation:
aws emr-serverless create-application \
  --name "my-application" \
  --release-label "emr-7.12.0" \
  --type "SPARK" \
  --job-level-cost-allocation-configuration '{
    "enabled": true
  }'

# Disable job-level cost allocation:
aws emr-serverless create-application \
```

```
--name "my-application" \  
--release-label "emr-7.12.0" \  
--type "SPARK" \  
--job-level-cost-allocation-configuration '{  
    "enabled": false  
}'
```

Perbarui `jobLevelCostAllocationConfiguration` parameter untuk aplikasi yang ada:

```
# Enable job-level cost allocation:  
aws emr-serverless update-application \  
    --application-id <application-id> \  
    --job-level-cost-allocation-configuration '{  
        "enabled": true  
    }'  
  
# Disable job-level cost allocation:  
aws emr-serverless update-application \  
    --application-id <application-id> \  
    --job-level-cost-allocation-configuration '{  
        "enabled": false  
    }'
```

Pertimbangan dan batasan

- Mengaktifkan alokasi biaya tingkat pekerjaan tidak secara surut mengaitkan biaya untuk menjalankan pekerjaan yang diselesaikan sebelum fitur diaktifkan. Job run dimulai setelah mengaktifkan fitur akan memiliki atribusi biaya granular.
- Parameter alokasi biaya tingkat pekerjaan hanya dapat diperbarui ketika Aplikasi dalam status `CREATED` atau `STOPPED`.
- Ketika alokasi biaya tingkat pekerjaan diaktifkan, biaya dikaitkan dengan pekerjaan individu berjalan daripada aplikasi. Untuk melihat biaya gabungan di tingkat aplikasi, Anda harus menerapkan tag yang konsisten (seperti nama aplikasi atau id aplikasi) ke semua pekerjaan yang berjalan dalam aplikasi tersebut dan memfilter menurut tag tersebut di Cost Explorer atau Laporan Biaya dan Penggunaan.

Jalankan sesi interaktif dengan Amazon EMR Tanpa Server melalui Spark Connect

Dengan rilis Amazon EMR `emr-7.13.0` dan yang lebih baru, Anda dapat terhubung ke aplikasi Amazon EMR Tanpa Server dari PySpark klien yang dikelola sendiri seperti VS Code,, PyCharm dan notebook Jupyter menggunakan sesi EMR Tanpa Server dengan Apache Spark Connect. APIs Spark Connect menggunakan arsitektur client-server yang memisahkan kode aplikasi Anda dari proses driver Spark. Anda mengembangkan dan men-debug PySpark kode di IDE lokal Anda sementara operasi Spark berjalan pada komputasi EMR Tanpa Server. Spark Connect menawarkan manfaat berikut:

- Connect ke EMR Serverless dari PySpark klien mana pun, termasuk VS Code,, dan notebook Jupyter PyCharm.
- Tetapkan breakpoint dan langkahkan PySpark kode di IDE Anda saat DataFrames dijalankan pada data skala produksi dari jarak jauh.

Sesi Spark Connect adalah koneksi terkelola antara PySpark klien lokal Anda dan driver Spark yang berjalan di Amazon EMR Tanpa Server. Saat Anda memulai sesi, EMR Serverless menyediakan driver dan pelaksana Spark atas nama Anda. Klien lokal Anda mengirim DataFrame dan operasi SQL ke driver, dan driver menjalankannya dari jarak jauh. Sesi berlanjut hingga Anda menghentikannya atau mencapai batas waktu idle, sehingga Anda dapat menjalankan beberapa kueri secara interaktif tanpa memulai ulang Spark. Setiap sesi memiliki URL titik akhir dan token otentikasi sendiri yang Anda gunakan untuk terhubung.

Izin yang diperlukan

Selain izin yang diperlukan untuk mengakses Amazon EMR Tanpa Server, tambahkan juga izin berikut ke peran IAM Anda untuk mengakses titik akhir Spark Connect dan mengelola sesi Spark Connect:

```
emr-serverless:StartSession
```

Memberikan izin untuk membuat sesi Spark Connect pada aplikasi yang Anda tentukan sebagai Resource

`emr-serverless:GetSessionEndpoint`

Memberikan izin untuk mengambil URL titik akhir Spark Connect dan token otentikasi untuk suatu sesi.

`emr-serverless:GetSession`

Memberikan izin untuk mendapatkan status sesi.

`emr-serverless:ListSessions`

Memberikan izin untuk membuat daftar sesi pada aplikasi.

`emr-serverless:TerminateSession`

Memberikan izin untuk mengakhiri sesi.

`iam:PassRole`

Memberikan izin untuk mengakses peran eksekusi IAM saat membuat sesi Spark Connect. Amazon EMR Tanpa Server menggunakan peran ini untuk menjalankan beban kerja Anda.

`emr-serverless:GetResourceDashboard`

Memberikan izin untuk membuat URL UI Spark dan menyediakan akses ke log untuk sesi tersebut.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EMRServerlessApplicationLevelAccess",
      "Effect": "Allow",
      "Action": [
        "emr-serverless:StartSession",
        "emr-serverless:ListSessions"
      ],
      "Resource": [
        "arn:aws:emr-serverless:region:account-id:/applications/application-id"
      ]
    },
    {
      "Sid": "EMRServerlessSessionLevelAccess",
      "Effect": "Allow",
```

```

    "Action": [
      "emr-serverless:GetSession",
      "emr-serverless:GetSessionEndpoint",
      "emr-serverless:TerminateSession",
      "emr-serverless:GetResourceDashboard"
    ],
    "Resource": [
      "arn:aws:emr-serverless:region:account-id:/applications/application-id/
sessions/*"
    ]
  },
  {
    "Sid": "EMRServerlessRuntimeRoleAccess",
    "Effect": "Allow",
    "Action": [
      "iam:PassRole"
    ],
    "Resource": [
      "arn:aws:iam::account-id:role/EMRServerlessExecutionRole"
    ],
    "Condition": {
      "StringLike": {
        "iam:PassedToService": "emr-serverless.amazonaws.com"
      }
    }
  }
]
}

```

Bekerja dengan sesi interaktif

Untuk membuat aplikasi yang mendukung Spark Connect dan menghubungkannya, ikuti langkah-langkah ini.

Untuk memulai sesi Spark Connect

1. Buat aplikasi dengan sesi Spark Connect.

```

aws emr-serverless create-application \
  --type "SPARK" \
  --name "spark-connect-app" \
  --release-label emr-7.13.0 \

```

```
--interactive-configuration '{"sessionEnabled": true}'
```

2. Setelah Amazon EMR Serverless membuat aplikasi Anda, jalankan aplikasi jika Anda belum mengaktifkan auto-start untuk menerima sesi Spark Connect.

```
aws emr-serverless start-application \  
--application-id APPLICATION_ID
```

3. Gunakan perintah berikut untuk memeriksa status aplikasi Anda. Setelah status menjadi STARTED, mulailah sesi.

```
aws emr-serverless get-application \  
--application-id APPLICATION_ID
```

4. Mulai sesi dengan peran eksekusi IAM yang memberikan akses ke data Anda.

```
aws emr-serverless start-session \  
--application-id APPLICATION_ID \  
--execution-role-arn arn:aws:iam::account-id:role/EMRServerlessExecutionRole
```

5. Pantau status sesi menggunakan get-session API dan tunggu sesi masuk STARTED atau IDLE status.

```
aws emr-serverless get-session \  
--application-id APPLICATION_ID \  
--session-id SESSION_ID
```

6. Ambil titik akhir Spark Connect dan token otentikasi. URL endpoint yang dikembalikan oleh GetSessionEndpoint tidak menyertakan nomor port. Saat membuat URL sc:// koneksi, Anda harus menambahkan :443 — misalnya, sc:// hostname :443/;use_ssl=true;x-aws-proxy-auth= token Tanpa itu, PySpark klien default ke port 15002, yang tidak dapat dijangkau di EMR Tanpa Server.

```
aws emr-serverless get-session-endpoint \  
--application-id APPLICATION_ID \  
--session-id SESSION_ID
```

Responsnya mencakup URL endpoint dan token otentikasi:

```
{  
  "endpoint": "ENDPOINT_URL",
```

```
"authToken": "AUTH_TOKEN",
"authTokenExpiresAt": "AUTH_TOKEN_EXPIRY_TIME"
}
```

7. Setelah titik akhir siap, sambungkan dari PySpark klien. Instal PySpark klien yang cocok dengan versi Spark pada aplikasi EMR Serverless Anda, dan SDK untuk Python. AWS

```
# Match the PySpark version to your EMR Serverless release version (3.5.6 for
emr-7.13.0)
pip install pyspark[connect]==3.5.6
pip install boto3
```

Berikut ini adalah contoh skrip Python untuk memulai sesi dan mengirim permintaan langsung ke titik akhir sesi:

```
import boto3
import time
from pyspark.sql import SparkSession
from pyspark.sql.functions import col

client = boto3.client('emr-serverless', region_name='REGION')

APPLICATION_ID = 'APPLICATION_ID'
EXECUTION_ROLE = 'arn:aws:iam::account-id:role/EMRServerlessExecutionRole'

# Start the session
response = client.start_session(
    applicationId=APPLICATION_ID,
    executionRoleArn=EXECUTION_ROLE
)
session_id = response['sessionId']
print(f"Session {session_id} starting...")

# Wait for the session to be ready
while True:
    response = client.get_session(
        applicationId=APPLICATION_ID,
        sessionId=session_id
    )
    state = response['session']['state']
    print(f"Session state: {state}")
    if state in ('STARTED', 'IDLE'):
```

```
        break
    if state in ('FAILED', 'TERMINATED'):
        raise Exception(f"Session failed: {response['session'].get('stateDetails',
'Unknown error')}")
    time.sleep(5)

# Retrieve the Spark Connect endpoint and authentication token
response = client.get_session_endpoint(
    applicationId=APPLICATION_ID,
    sessionId=session_id
)

# Construct the authenticated remote URL
auth_token = response['authToken']
endpoint_url = response['endpoint']
connect_url = endpoint_url.replace("https://", "sc://", 1) + ":443/;use_ssl=true;"
connect_url += f"x-aws-proxy-auth={auth_token}"

# Start the Spark session
spark = SparkSession.builder.remote(connect_url).getOrCreate()
print(f"Connected. Spark version: {spark.version}")

# Run SQL
spark.sql("SELECT 1+1 AS result").show()

# Run DataFrame operations
df = spark.range(100).withColumn("squared", col("id") * col("id"))
df.show(10)
print(f"Count: {df.count()}")

# Stop the Spark session (disconnects the client only)
spark.stop()

# Terminate the EMR Serverless session to stop billing.
# spark.stop() only closes the local client connection. The remote session
# continues running and incurring charges until you explicitly terminate it
# or it reaches the idle timeout.
client.terminate_session(
    applicationId=APPLICATION_ID,
    sessionId=session_id
)
print(f"Session {session_id} terminated.")
```

Untuk mengakses UI Spark langsung atau Server Riwayat Spark untuk suatu sesi, gunakan API `GetResourceDashboard`

```
response = client.get_resource_dashboard(  
    applicationId=APPLICATION_ID,  
    resourceId=session_id,  
    resourceType='SESSION'  
)  
response['url']
```

Saat sesi aktif, URL membuka UI Apache Spark langsung untuk pemantauan kueri, tahapan, dan pelaksana secara real-time. Setelah sesi berakhir, Server Sejarah Spark tetap tersedia untuk analisis pasca-sesi melalui konsol Amazon EMR Tanpa Server.

Pertimbangan dan batasan

Pertimbangkan hal berikut saat menjalankan beban kerja interaktif melalui Spark Connect.

- Spark Connect didukung dengan rilis Amazon EMR Tanpa Server `emr-7.13.0` dan yang lebih baru.
- Spark Connect hanya didukung untuk mesin Apache Spark.
- Spark Connect mendukung DataFrame dan SQL APIs in. PySpark Berbasis RDD APIs tidak didukung.
- Token otentikasi dibatasi waktu hingga 1 jam. Ketika token kedaluwarsa, panggilan gRPC gagal dengan kesalahan otentikasi. Hubungi `GetSessionEndpoint` untuk mendapatkan token baru dan buat yang baru `SparkSession` dengan token yang diperbarui.
- Sesi berakhir setelah batas waktu idle yang dapat dikonfigurasi. Batas waktu default diatur ke 1 jam.
- Setiap sesi memiliki batas keras 24 jam secara default, setelah itu otomatis berakhir bahkan jika itu secara aktif menjalankan tugas.
- Setiap aplikasi EMR Tanpa Server mendukung hingga 25 sesi bersamaan secara default. Untuk meminta kenaikan batas, hubungi AWS Support.
- Secara default, `autoStopConfig` aktif untuk aplikasi. Aplikasi berhenti secara otomatis setelah 15 menit tanpa sesi aktif atau pekerjaan berjalan. Anda dapat mengubah konfigurasi ini sebagai bagian dari `update-application` permintaan `create-application` atau permintaan Anda.

- Untuk pengalaman startup terbaik, konfigurasi kapasitas pra-inisialisasi untuk driver dan pelaksana.
- Anda harus mengaktifkan `AutoStart` atau memulai aplikasi secara manual sebelum memulai sesi EMR Tanpa Server.
- PySpark Versi yang diinstal secara lokal harus cocok dengan versi Apache Spark pada aplikasi Amazon EMR Serverless Anda (3.5.6 untuk). `emr-7.13.0` Ketidakcocokan versi menyebabkan `ImportError` atau perilaku yang tidak terduga.
- Kontrol akses berbutir halus melalui Lake Formation tidak didukung untuk sesi Spark Connect.
- Propagasi Identitas Tepercaya tidak didukung untuk sesi interaktif dengan Spark Connect.
- Penyimpanan tanpa server di EMR Tanpa Server tidak didukung untuk sesi interaktif dengan Spark Connect.
- Tidak ada biaya tambahan untuk menggunakan Spark Connect. Anda hanya membayar untuk sumber daya komputasi EMR Tanpa Server (vCPU, memori, dan penyimpanan) yang dikonsumsi selama sesi Anda.
- Konfigurasi Spark dicadangkan `spark.connect.grpc.binding.address` oleh EMR Tanpa Server dan tidak dapat diganti oleh pengguna.
- PySpark Paket yang Anda instal secara lokal harus cocok dengan versi Spark pada aplikasi EMR Serverless Anda. Ketidakcocokan versi menyebabkan kesalahan koneksi. Python UDFs (`@udf,spark.udf.register`) juga memerlukan versi minor Python lokal untuk mencocokkan pekerja, atau gagal. `PYTHON_VERSION_MISMATCH` Fungsi dan DataFrame operasi SQL bawaan tidak memerlukan kecocokan versi Python.
- Untuk meneruskan konfigurasi Spark dengan `start-session`, atur `runtimeConfiguration` di bawah parameter. `--configuration-overrides start-job-runAPI` menggunakan `applicationConfiguration` sebagai gantinya.

Jalankan beban kerja interaktif dengan EMR Serverless melalui EMR Studio

Dengan aplikasi interaktif EMR Tanpa Server, jalankan beban kerja interaktif untuk Spark dengan EMR Tanpa Server menggunakan notebook yang di-host di EMR Studio.

Ikhtisar

Aplikasi interaktif adalah aplikasi EMR Tanpa Server yang memiliki kemampuan interaktif diaktifkan. Dengan aplikasi interaktif Amazon EMR Tanpa Server, Anda dapat menjalankan beban kerja interaktif dengan notebook Jupyter yang dikelola di Amazon EMR Studio. Ini membantu insinyur data, ilmuwan data, dan analis data menggunakan EMR Studio untuk menjalankan analitik interaktif dengan kumpulan data di penyimpanan data seperti Amazon S3 dan Amazon DynamoDB.

Kasus penggunaan untuk aplikasi interaktif di EMR Tanpa Server meliputi yang berikut:

- Insinyur data menggunakan pengalaman IDE di EMR Studio untuk membuat skrip ETL. Skrip menyerap data dari lokal, mengubah data untuk analisis, dan menyimpan data di Amazon S3.
- Ilmuwan data menggunakan notebook untuk mengeksplorasi kumpulan data dan melatih model pembelajaran mesin (ML) untuk mendeteksi anomali dalam kumpulan data.
- Analis data mengeksplorasi kumpulan data dan membuat skrip yang menghasilkan laporan harian untuk memperbarui aplikasi seperti dasbor bisnis.

Prasyarat

Untuk menggunakan beban kerja interaktif dengan EMR Serverless, memenuhi persyaratan berikut:

- EMR Aplikasi interaktif tanpa server didukung dengan Amazon EMR 6.14.0 dan yang lebih tinggi.
- Untuk mengakses aplikasi interaktif Anda, jalankan beban kerja yang Anda kirimkan, dan jalankan notebook interaktif dari EMR Studio, Anda memerlukan izin dan peran tertentu. Untuk informasi lebih lanjut, lihat [izin yang diperlukan untuk beban kerja interaktif](#).

Izin yang diperlukan untuk beban kerja interaktif

Selain [izin dasar yang diperlukan untuk mengakses EMR](#) Tanpa Server, konfigurasi izin tambahan untuk identitas atau peran IAM Anda:

Untuk mengakses aplikasi interaktif Anda

Siapkan izin pengguna dan Ruang Kerja untuk EMR Studio. Untuk informasi selengkapnya, lihat [Mengonfigurasi izin pengguna EMR Studio](#) di Panduan Manajemen EMR Amazon.

Untuk menjalankan beban kerja yang Anda kirimkan dengan EMR Tanpa Server

Siapkan peran runtime pekerjaan. Untuk informasi lebih lanjut, lihat [Buat peran runtime pekerjaan](#).

Untuk menjalankan notebook interaktif dari EMR Studio

Tambahkan izin tambahan berikut ke kebijakan IAM untuk pengguna Studio:

- **emr-serverless:AccessInteractiveEndpoints**- Memberikan izin untuk mengakses dan terhubung ke aplikasi interaktif yang Anda tentukan sebagai Resource. Izin ini diperlukan untuk melampirkan ke aplikasi EMR Tanpa Server dari EMR Studio Workspace.
- **iam:PassRole**- Memberikan izin untuk mengakses peran eksekusi IAM yang Anda rencanakan untuk digunakan saat Anda melampirkan ke aplikasi. PassRole izin yang sesuai diperlukan untuk melampirkan ke aplikasi EMR Tanpa Server dari EMR Studio Workspace.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EMRServerlessInteractiveAccess",
      "Effect": "Allow",
      "Action": [
        "emr-serverless:AccessInteractiveEndpoints"
      ],
      "Resource": [
        "arn:aws:emr-serverless:*:123456789012:/applications/*"
      ]
    },
    {
      "Sid": "EMRServerlessRuntimeRoleAccess",
      "Effect": "Allow",
      "Action": [
```

```
    "iam:PassRole"
  ],
  "Resource": [
    "arn:aws:iam::123456789012:role/EMRServerlessInteractiveRole"
  ],
  "Condition": {
    "StringLike": {
      "iam:PassedToService": "emr-serverless.amazonaws.com"
    }
  }
}
]
```

Mengonfigurasi aplikasi interaktif

Gunakan langkah-langkah tingkat tinggi berikut untuk membuat aplikasi EMR Tanpa Server dengan kemampuan interaktif dari Amazon EMR Studio di Konsol Manajemen AWS

1. Ikuti langkah-langkah [Memulai dengan Amazon EMR Tanpa Server](#) untuk membuat aplikasi.
2. Kemudian, luncurkan ruang kerja dari EMR Studio dan lampirkan ke aplikasi EMR Tanpa Server sebagai opsi komputasi. Untuk informasi selengkapnya, lihat tab Beban kerja interaktif di Langkah 2 dari dokumentasi [Memulai Tanpa Server EMR](#).

Saat Anda melampirkan aplikasi ke Studio Workspace, aplikasi mulai terpicu secara otomatis jika aplikasi tersebut belum berjalan. Anda juga dapat memulai aplikasi terlebih dahulu dan menyiapkannya sebelum Anda melampirkannya ke Workspace.

Pertimbangan dengan aplikasi interaktif

- EMR Aplikasi interaktif tanpa server didukung dengan Amazon EMR 6.14.0 dan yang lebih tinggi.
- EMR Studio adalah satu-satunya klien yang terintegrasi dengan aplikasi interaktif EMR Serverless. Kemampuan EMR Studio berikut tidak didukung dengan aplikasi interaktif EMR Serverless: Kolaborasi ruang kerja, SQL Explorer, dan eksekusi terprogram notebook.
- Aplikasi interaktif hanya didukung untuk mesin Spark.
- Aplikasi interaktif mendukung kernel Python 3, PySpark dan Spark Scala.
- Anda dapat menjalankan hingga 25 notebook bersamaan pada satu aplikasi interaktif.

- Tidak ada endpoint atau antarmuka API yang mendukung notebook Jupyter yang dihosting sendiri dengan aplikasi interaktif.
- Untuk pengalaman startup yang dioptimalkan, kami sarankan Anda mengonfigurasi kapasitas pra-inisialisasi untuk driver dan pelaksana, dan Anda memulai aplikasi terlebih dahulu. Ketika Anda memulai aplikasi terlebih dahulu, Anda memastikan bahwa itu siap ketika Anda ingin melampirkannya ke Workspace Anda.

```
aws emr-serverless start-application \  
--application-id your-application-id
```

- Secara default, `autoStopConfig` diaktifkan untuk aplikasi. Ini mematikan aplikasi setelah 30 menit waktu idle. Anda dapat mengubah konfigurasi ini sebagai bagian dari `update-application` permintaan `create-application` atau permintaan Anda.
- Saat menggunakan aplikasi interaktif, kami sarankan Anda mengonfigurasi kapasitas kernel, driver, dan pelaksana yang telah diinisialisasi sebelumnya untuk menjalankan notebook Anda. Setiap sesi interaktif Spark memerlukan satu kernel dan satu driver, sehingga EMR Serverless mempertahankan pekerja kernel pra-inisialisasi untuk setiap driver yang telah diinisialisasi sebelumnya. Secara default, EMR Serverless mempertahankan kapasitas pra-inisialisasi dari satu pekerja kernel di seluruh aplikasi bahkan jika Anda tidak menentukan kapasitas pra-inisialisasi untuk driver. Setiap pekerja kernel menggunakan 4 vCPU dan 16 GB memori. Untuk informasi harga saat ini, lihat halaman [Harga EMR Amazon](#).
- Anda harus memiliki kuota layanan vCPU yang cukup Akun AWS untuk menjalankan beban kerja interaktif. Jika Anda tidak menjalankan Formation-enabled beban kerja Lake, kami sarankan setidaknya 24 vCPU. Jika Anda melakukannya, kami sarankan setidaknya 28 vCPU.
- EMR Tanpa Server secara otomatis menghentikan kernel dari notebook jika mereka telah menganggur selama lebih dari 60 menit. EMR Tanpa Server menghitung waktu idle kernel dari aktivitas terakhir yang diselesaikan selama sesi notebook. Saat ini Anda tidak dapat mengubah pengaturan batas waktu idle kernel.
- Untuk mengaktifkan Lake Formation dengan beban kerja interaktif, atur konfigurasi `spark.emr-serverless.lakeformation.enabled` ke `true` bawah `spark-defaults` klasifikasi dalam `runtime-configuration` objek saat Anda [membuat aplikasi EMR Tanpa Server](#). Untuk mempelajari lebih lanjut, lihat [Mengaktifkan Formasi Danau di Amazon EMR](#).

Jalankan beban kerja interaktif dengan EMR Tanpa Server melalui titik akhir Apache Livy

Dengan Amazon EMR merilis 6.14.0 dan yang lebih tinggi, buat dan aktifkan titik akhir Apache Livy saat membuat aplikasi EMR Tanpa Server dan jalankan beban kerja interaktif melalui notebook yang dihosting sendiri atau dengan klien khusus. Endpoint Apache Livy menawarkan manfaat berikut:

- Anda dapat terhubung dengan aman ke titik akhir Apache Livy melalui notebook Jupyter dan mengelola beban kerja Apache Spark dengan antarmuka REST Apache Livy.
- Gunakan operasi Apache Livy REST API untuk aplikasi web interaktif yang menggunakan data dari beban kerja Apache Spark.

Prasyarat

Untuk menggunakan endpoint Apache Livy dengan EMR Serverless, memenuhi persyaratan berikut:

- Selesaikan langkah-langkah dalam [Memulai dengan Amazon EMR Tanpa Server](#).
- Untuk menjalankan beban kerja interaktif melalui titik akhir Apache Livy, Anda memerlukan izin dan peran tertentu. Untuk informasi selengkapnya, lihat [Izin yang diperlukan untuk beban kerja interaktif](#).

Izin yang diperlukan

Selain izin yang diperlukan untuk mengakses EMR Tanpa Server, tambahkan juga izin berikut ke peran IAM Anda untuk mengakses titik akhir Apache Livy dan menjalankan aplikasi:

- `emr-serverless:AccessLivyEndpoints`— memberikan izin untuk mengakses dan terhubung ke Livy-enabled aplikasi yang Anda tentukan sebagai `Resource`. Anda memerlukan izin ini untuk menjalankan operasi REST API yang tersedia dari titik akhir Apache Livy.
- `iam:PassRole`— memberikan izin untuk mengakses peran eksekusi IAM saat membuat sesi Apache Livy. EMR Tanpa Server akan menggunakan peran ini untuk menjalankan beban kerja Anda.
- `emr-serverless:GetDashboardForJobRun`— memberikan izin untuk menghasilkan UI Spark Live dan tautan log driver dan menyediakan akses ke log sebagai bagian dari hasil sesi Apache Livy.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EMRServerlessInteractiveAccess",
      "Effect": "Allow",
      "Action": [
        "emr-serverless:AccessLivyEndpoints"
      ],
      "Resource": [
        "arn:aws:emr-serverless:*:123456789012:/applications/*"
      ]
    },
    {
      "Sid": "EMRServerlessRuntimeRoleAccess",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
        "arn:aws:iam::123456789012:role/EMRServerlessExecutionRole"
      ],
      "Condition": {
        "StringLike": {
          "iam:PassedToService": "emr-serverless.amazonaws.com"
        }
      }
    },
    {
      "Sid": "EMRServerlessDashboardAccess",
      "Effect": "Allow",
      "Action": [
        "emr-serverless:GetDashboardForJobRun"
      ],
      "Resource": [
        "arn:aws:emr-serverless:*:123456789012:/applications/*"
      ]
    }
  ]
}
```

Memulai

Untuk membuat Livy-enabled aplikasi Apache dan menjalankannya, ikuti langkah-langkah ini.

1. Untuk membuat Livy-enabled aplikasi Apache, jalankan perintah berikut.

```
aws emr-serverless create-application \  
--name my-application-name \  
--type 'application-type' \  
--release-label <Amazon EMR-release-version>  
--interactive-configuration '{"livyEndpointEnabled": true}'
```

2. Setelah EMR Serverless membuat aplikasi Anda, mulai aplikasi untuk membuat endpoint Apache Livy tersedia.

```
aws emr-serverless start-application \  
--application-id application-id
```

Gunakan perintah berikut untuk memeriksa apakah status aplikasi Anda. Setelah status menjadi `STARTED`, akses titik akhir Apache Livy.

```
aws emr-serverless get-application \  
--region <AWS_REGION> --application-id >application_id<
```

3. Gunakan URL berikut untuk mengakses titik akhir:

```
https://_<application-id>_.livy.emr-serverless-  
services._<AWS_REGION>_.amazonaws.com
```

Setelah titik akhir siap, kirimkan beban kerja berdasarkan kasus penggunaan Anda. Anda harus menandatangani setiap permintaan ke titik akhir dengan [protokol SigV4](#) dan meneruskan header otorisasi. Anda dapat menggunakan metode berikut untuk menjalankan beban kerja:

- Klien HTTP - kirimkan operasi API endpoint Apache Livy Anda dengan klien HTTP kustom.
- Kernel Sparkmagic — jalankan kernel sparkmagic secara lokal dan kirimkan kueri interaktif dengan notebook Jupyter.

Klien HTTP

Untuk membuat sesi Apache Livy, `emr-serverless.session.executionRoleArn` kirimkan `conf` parameter badan permintaan Anda. Contoh berikut adalah `POST /sessions` permintaan sampel.

```
{
  "kind": "pyspark",
  "heartbeatTimeoutInSeconds": 60,
  "conf": {
    "emr-serverless.session.executionRoleArn": "<executionRoleArn>"
  }
}
```

Tabel berikut menjelaskan semua operasi Apache Livy API yang tersedia.

Operasi API	Deskripsi
DAPATKAN /sesi	Mengembalikan daftar semua sesi interaktif aktif.
POSTING/sesi	Membuat sesi interaktif baru melalui spark atau pyspark.
DAPATKAN /sesi/ < > <i>sessionId</i>	Mengembalikan informasi sesi.
DAPATKAN /sesi/ < >/status <i>sessionId</i>	Mengembalikan keadaan sesi.
HAPUS/sesi/ < > <i>sessionId</i>	Menghentikan dan menghapus sesi.
DAPATKAN /sesi/ < >/pernyataan <i>sessionId</i>	Mengembalikan semua pernyataan dalam sesi.
POST /sesi/ < >/pernyataan <i>sessionId</i>	Menjalankan pernyataan dalam sesi.
DAPATKAN /sesi/ < >/pernyataan/< > <i>sessionId statementId</i>	Mengembalikan rincian pernyataan yang ditentukan dalam sesi.
POST/sesi/ < >/pernyataan/< >/batalkan <i>sessionId statementId</i>	Membatalkan pernyataan yang ditentukan dalam sesi ini.

Mengirim permintaan ke titik akhir Apache Livy

Anda juga dapat mengirim permintaan langsung ke titik akhir Apache Livy dari klien HTTP. Melakukannya memungkinkan Anda menjalankan kode dari jarak jauh untuk kasus penggunaan di luar buku catatan.

Sebelum Anda mulai mengirim permintaan ke titik akhir, pastikan Anda telah menginstal pustaka berikut:

```
pip3 install botocore awscrt requests
```

Berikut ini adalah contoh script Python untuk mengirim permintaan HTTP langsung ke endpoint:

```
from botocore import crt
import requests
from botocore.awsrequest import AWSRequest
from botocore.credentials import Credentials
import botocore.session
import json, pprint, textwrap

endpoint = 'https://<application_id>.livy.emr-serverless-
services.<AWS_REGION>.amazonaws.com'
headers = {'Content-Type': 'application/json'}

session = botocore.session.Session()
signer = crt.auth.CrtS3SigV4Auth(session.get_credentials(), 'emr-serverless',
    '<AWS_REGION>')

### Create session request

data = {'kind': 'pyspark', 'heartbeatTimeoutInSeconds': 60, 'conf': { 'emr-
serverless.session.executionRoleArn': 'arn:aws:iam::123456789012:role/role1'}}

request = AWSRequest(method='POST', url=endpoint + "/sessions", data=json.dumps(data),
    headers=headers)

request.context["payload_signing_enabled"] = False

signer.add_auth(request)

prepped = request.prepare()
```

```
r = requests.post(prepped.url, headers=prepped.headers, data=json.dumps(data))

pprint.pprint(r.json())

### List Sessions Request

request = AWSRequest(method='GET', url=endpoint + "/sessions", headers=headers)

request.context["payload_signing_enabled"] = False

signer.add_auth(request)

prepped = request.prepare()

r2 = requests.get(prepped.url, headers=prepped.headers)
pprint.pprint(r2.json())

### Get session state

session_url = endpoint + r.headers['location']

request = AWSRequest(method='GET', url=session_url, headers=headers)

request.context["payload_signing_enabled"] = False

signer.add_auth(request)

prepped = request.prepare()

r3 = requests.get(prepped.url, headers=prepped.headers)

pprint.pprint(r3.json())

### Submit Statement

data = {
    'code': "1 + 1"
}

statements_url = endpoint + r.headers['location'] + "/statements"
```

```
request = AWSRequest(method='POST', url=statements_url, data=json.dumps(data),
    headers=headers)

request.context["payload_signing_enabled"] = False

signer.add_auth(request)

prepped = request.prepare()

r4 = requests.post(prepped.url, headers=prepped.headers, data=json.dumps(data))

pprint.pprint(r4.json())

### Check statements results

specific_statement_url = endpoint + r4.headers['location']

request = AWSRequest(method='GET', url=specific_statement_url, headers=headers)

request.context["payload_signing_enabled"] = False

signer.add_auth(request)

prepped = request.prepare()

r5 = requests.get(prepped.url, headers=prepped.headers)

pprint.pprint(r5.json())

### Delete session

session_url = endpoint + r.headers['location']

request = AWSRequest(method='DELETE', url=session_url, headers=headers)

request.context["payload_signing_enabled"] = False

signer.add_auth(request)

prepped = request.prepare()

r6 = requests.delete(prepped.url, headers=prepped.headers)
```

```
pprint.pprint(r6.json())
```

Kernel Sparkmagic

Sebelum Anda menginstal sparkmagic, pastikan bahwa Anda telah mengonfigurasi AWS kredensial dalam contoh di mana Anda ingin menginstal sparkmagic

1. Instal sparkmagic dengan mengikuti langkah-langkah [instalasi](#). Perhatikan bahwa Anda hanya melakukan empat langkah pertama.
2. Kernel sparkmagic mendukung autentikator khusus, sehingga Anda dapat mengintegrasikan autentikator dengan kernel sparkmagic sehingga setiap permintaan ditandatangani SigV4.
3. Instal autentikator khusus EMR Tanpa Server.

```
pip install emr-serverless-customauth
```

4. Sekarang berikan path ke authenticator kustom dan URL endpoint Apache Livy di file json konfigurasi sparkmagic. Gunakan perintah berikut untuk membuka file konfigurasi.

```
vim ~/.sparkmagic/config.json
```

Berikut ini adalah config.json file sampel.

```
{
  "kernel_python_credentials" : {
    "username": "",
    "password": "",
    "url": "https://<application-id>.livy.emr-serverless-
services.<AWS_REGION>.amazonaws.com",
    "auth": "Custom_Auth"
  },

  "kernel_scala_credentials" : {
    "username": "",
    "password": "",
    "url": "https://<application-id>.livy.emr-serverless-
services.<AWS_REGION>.amazonaws.com",
    "auth": "Custom_Auth"
  },
  "authenticators": {
    "None": "sparkmagic.auth.customauth.Authenticator",
```

```

    "Basic_Access": "sparkmagic.auth.basic.Basic",
    "Custom_Auth":
"emr_serverless_customauth.customauthenticator.EMRServerlessCustomSigV4Signer"
  },
  "livy_session_startup_timeout_seconds": 600,
  "ignore_ssl_errors": false
}

```

5. Mulai lab Jupyter. Ini harus menggunakan otentikasi khusus yang Anda atur pada langkah terakhir.
6. Anda kemudian dapat menjalankan perintah notebook berikut dan kode Anda untuk memulai.

```
%info //Returns the information about the current sessions.
```

```

%%configure -f //Configure information specific to a session. We supply
executionRoleArn in this example. Change it for your use case.
{
  "driverMemory": "4g",
  "conf": {
    "emr-serverless.session.executionRoleArn":
"arn:aws:iam::123456789012:role/JobExecutionRole"
  }
}

```

```
<your code>//Run your code to start the session
```

Secara internal, setiap instruksi memanggil setiap operasi Apache Livy API melalui URL endpoint Apache Livy yang dikonfigurasi. Anda kemudian dapat menulis instruksi Anda sesuai dengan kasus penggunaan Anda.

Pertimbangan-pertimbangan

Pertimbangkan pertimbangan berikut saat menjalankan beban kerja interaktif melalui titik akhir Apache Livy.

- EMR Tanpa Server mempertahankan isolasi tingkat sesi menggunakan prinsipal pemanggil. Prinsipal penelepon yang membuat sesi adalah satu-satunya yang dapat mengakses sesi itu. Untuk isolasi yang lebih terperinci, konfigurasi identitas sumber saat Anda mengasumsikan kredensial. Dalam hal ini, EMR Tanpa Server memberlakukan isolasi tingkat sesi berdasarkan

prinsip penelepon dan identitas sumber. Untuk informasi lebih lanjut tentang identitas sumber, lihat [Memantau dan mengontrol tindakan yang diambil dengan peran yang diasumsikan](#).


- Endpoint Apache Livy didukung dengan rilis EMR Serverless 6.14.0 dan yang lebih tinggi.
- Endpoint Apache Livy hanya didukung untuk mesin Apache Spark.
- Titik akhir Apache Livy mendukung Scala Spark dan. PySpark
- Secara default, `autoStopConfig` diaktifkan di aplikasi Anda. Ini berarti bahwa aplikasi dimatikan setelah 15 menit menganggur. Anda dapat mengubah konfigurasi ini sebagai bagian dari `update-application` permintaan `create-application` atau permintaan Anda.
- Anda dapat menjalankan hingga 25 sesi bersamaan pada satu aplikasi berkemampuan endpoint Apache Livy.
- Untuk pengalaman startup terbaik, kami sarankan Anda mengonfigurasi kapasitas pra-inisialisasi untuk driver dan pelaksana.
- Anda harus memulai aplikasi secara manual sebelum menghubungkan ke titik akhir Apache Livy.
- Anda harus memiliki kuota layanan vCPU yang cukup Akun AWS untuk menjalankan beban kerja interaktif dengan endpoint Apache Livy. Kami menyarankan setidaknya 24 vCPU.
- Batas waktu sesi Apache Livy default adalah 1 jam. Jika Anda tidak menjalankan pernyataan selama satu jam, maka Apache Livy menghapus sesi dan melepaskan driver dan pelaksana. Dari rilis `emr-7.8.0`, nilai ini dapat diatur dengan menentukan `ttl` parameter sebagai bagian dari `/sessions` POST permintaan Livy, misalnya, 2h (jam), (menit), (detik), 120m (milidetik). 7200s 7200000ms

Note

Konfigurasi ini tidak dapat diubah sebelum `emr-7.8.0`. Berikut ini adalah contoh dari badan POST `/sessions` permintaan.

```
{
  "kind": "pyspark",
  "heartbeatTimeoutInSeconds": 60,
  "conf": {
    "emr-serverless.session.executionRoleArn": "executionRoleArn"
  },
  "ttl": "2h"
}
```

- Dimulai dengan rilis EMR Amazon emr-7.8.0 untuk Aplikasi dengan kontrol akses berbutir halus melalui LakeFormation diaktifkan, pengaturan dapat dinonaktifkan per sesi. [Untuk informasi lebih lanjut tentang mengaktifkan kontrol akses berbutir halus untuk aplikasi EMR Tanpa Server, lihat Metode untuk kontrol akses berbutir halus.](#)

 Note

Lake Formation tidak dapat diaktifkan untuk Sesi ketika belum diaktifkan untuk Aplikasi. Berikut ini adalah contoh dari badan POST /sessions permintaan.

```
{
  "kind": "pyspark",
  "heartbeatTimeoutInSeconds": 60,
  "conf": {
    "emr-serverless.session.executionRoleArn": "executionRoleArn"
  },
  "spark.emr-serverless.lakeformation.enabled" : "false"
}
```

- Hanya sesi aktif yang dapat berinteraksi dengan titik akhir Apache Livy. Setelah sesi selesai, membatalkan, atau berakhir, Anda tidak dapat mengaksesnya melalui titik akhir Apache Livy.

Pencatatan log dan pemantauan

Pemantauan merupakan bagian penting dari menjaga keandalan, ketersediaan, dan kinerja aplikasi dan pekerjaan EMR Tanpa Server. Anda harus mengumpulkan data pemantauan dari semua bagian solusi EMR Tanpa Server Anda sehingga kegagalan multipoint dapat di-debug lebih mudah jika terjadi.

Topik

- [Menyimpan log](#)
- [Memutar log](#)
- [Mengenkripsi log](#)
- [Konfigurasi properti Apache Log4j2 untuk Amazon EMR Tanpa Server](#)
- [Pemantauan EMR Tanpa Server](#)
- [Mengotomatisasi EMR Tanpa Server dengan Amazon EventBridge](#)

Menyimpan log

Untuk memantau kemajuan pekerjaan Anda di EMR Tanpa Server dan memecahkan masalah kegagalan pekerjaan, pilih cara EMR Tanpa Server menyimpan dan menyajikan log aplikasi. Saat Anda mengirimkan pekerjaan, tentukan penyimpanan terkelola, Amazon S3, dan Amazon CloudWatch sebagai opsi pencatatan Anda.

Dengan CloudWatch, tentukan jenis log dan lokasi log yang ingin Anda gunakan, atau terima tipe dan lokasi default. Untuk informasi lebih lanjut tentang CloudWatch log, lihat [the section called “Amazon CloudWatch”](#). Dengan penyimpanan terkelola dan pencatatan S3, tabel berikut mencantumkan lokasi log dan ketersediaan UI yang dapat Anda harapkan jika Anda memilih [penyimpanan terkelola](#), bucket [Amazon S3](#), atau keduanya.

Opsi	Log peristiwa	Log Kontainer	UI Aplikasi
Penyimpanan terkelola	Disimpan dalam penyimpanan terkelola	Disimpan dalam penyimpanan terkelola	Didukung

Opsi	Log peristiwa	Log Kontainer	UI Aplikasi
Penyimpanan terkelola dan bucket S3	Disimpan di kedua tempat	Disimpan dalam ember S3	Didukung
Buket Amazon S3	Disimpan dalam ember S3	Disimpan dalam ember S3	Tidak didukung ¹

¹ Kami menyarankan agar Anda tetap memilih opsi Penyimpanan Terkelola. Jika tidak, Anda tidak dapat menggunakan aplikasi bawaan UIs.

Logging untuk EMR Tanpa Server dengan penyimpanan terkelola

Secara default, EMR Tanpa Server menyimpan log aplikasi dengan aman di penyimpanan terkelola Amazon EMR selama maksimal 30 hari.

Note

Jika Anda menonaktifkan opsi default, Amazon EMR tidak dapat memecahkan masalah pekerjaan Anda atas nama Anda. Contoh: Anda tidak dapat mengakses Spark-UI dari Konsol Tanpa Server EMR.

Untuk menonaktifkan opsi ini dari EMR Studio, batalkan centang kotak Izinkan AWS untuk menyimpan log selama 30 hari di bagian Pengaturan tambahan pada halaman Kirim pekerjaan.

Untuk mematikan opsi ini dari AWS CLI, gunakan `managedPersistenceMonitoringConfiguration` konfigurasi saat Anda mengirimkan pekerjaan yang dijalankan.

```
{
  "monitoringConfiguration": {
    "managedPersistenceMonitoringConfiguration": {
      "enabled": false
    }
  }
}
```

Jika aplikasi EMR Tanpa Server Anda berada dalam subnet pribadi dengan titik akhir VPC untuk Amazon S3 dan Anda melampirkan kebijakan titik akhir untuk mengontrol akses, tambahkan izin berikut untuk EMR Tanpa Server untuk menyimpan dan menyajikan log aplikasi. Ganti Resource dengan AppInfo bucket dari tabel wilayah yang tersedia dalam [Kebijakan sampel untuk subnet pribadi yang mengakses Amazon S3](#).

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EMRServerlessManagedLogging",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:PutObjectAcl"
      ],
      "Resource": [
        "arn:aws:s3:::prod.us-east-1.appinfo.src",
        "arn:aws:s3:::prod.us-east-1.appinfo.src/*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:PrincipalServiceName": "emr-serverless.amazonaws.com",
          "aws:SourceVpc": "vpc-12345678"
        }
      }
    }
  ]
}
```

Selain itu, gunakan tombol `aws:SourceVpc` kondisi untuk memastikan bahwa permintaan berjalan melalui VPC tempat titik akhir VPC dilampirkan.

Logging untuk EMR Tanpa Server dengan bucket Amazon S3

Sebelum pekerjaan Anda dapat mengirim data log ke Amazon S3, sertakan izin berikut dalam kebijakan izin untuk peran runtime pekerjaan. Ganti *amzn-s3-demo-logging-bucket* dengan nama bucket logging Anda.

JSON

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

Untuk menyiapkan bucket Amazon S3 untuk menyimpan log dari AWS CLI, gunakan `s3MonitoringConfiguration` konfigurasi saat Anda memulai menjalankan pekerjaan. Untuk melakukan ini, berikan yang berikut `--configuration-overrides` dalam konfigurasi.

```
{
  "monitoringConfiguration": {
    "s3MonitoringConfiguration": {
      "logUri": "s3://amzn-s3-demo-logging-bucket/logs/"
    }
  }
}
```

Untuk pekerjaan batch yang tidak mengaktifkan percobaan ulang, EMR Serverless mengirimkan log ke jalur berikut:

```
'/applications/<applicationId>/jobs/<jobId>'
```

Log driver percikan disimpan di jalur berikut oleh EMR Tanpa Server

```
'/applications/<applicationId>/jobs/<jobId>/SPARK_DRIVER/'
```

Log pelaksana Spark disimpan di jalur berikut oleh EMR Tanpa Server

```
'/applications/<applicationId>/jobs/<jobId>/SPARK_EXECUTOR/<EXECUTOR-ID>'
```

<EXECUTOR-ID>Ini adalah bilangan bulat.

EMR Serverless merilis 7.1.0 dan mendukung upaya coba lagi yang lebih tinggi untuk pekerjaan streaming dan pekerjaan batch. Jika Anda menjalankan pekerjaan dengan percobaan ulang diaktifkan, EMR Serverless secara otomatis menambahkan nomor percobaan ke awalan jalur log, sehingga Anda dapat membedakan dan melacak log dengan lebih baik.

```
'/applications/<applicationId>/jobs/<jobId>/attempts/<attemptNumber>/'
```

Logging untuk EMR Tanpa Server dengan Amazon CloudWatch

Saat Anda mengirimkan pekerjaan ke aplikasi EMR Tanpa Server, pilih Amazon CloudWatch sebagai opsi untuk menyimpan log aplikasi Anda. Ini memungkinkan Anda untuk menggunakan fitur analisis CloudWatch log seperti Wawasan CloudWatch Log dan Live Tail. Anda juga dapat melakukan streaming log dari CloudWatch sistem lain seperti OpenSearch untuk analisis lebih lanjut.

EMR Tanpa Server menyediakan pencatatan waktu nyata untuk log driver. Anda dapat mengakses log secara real time dengan kemampuan CloudWatch live tail, atau melalui perintah ekor CloudWatch CLI.

Secara default, CloudWatch logging dinonaktifkan untuk EMR Tanpa Server. Untuk mengaktifkannya, gunakan konfigurasi di [AWS CLI](#).

Note

Amazon CloudWatch menerbitkan log secara real time, sehingga menghasilkan lebih banyak sumber daya dari pekerja. Jika Anda memilih kapasitas pekerja yang rendah, dampaknya terhadap waktu kerja Anda mungkin meningkat. Jika Anda mengaktifkan CloudWatch pencatatan, kami sarankan Anda memilih kapasitas pekerja yang lebih besar. Mungkin juga publikasi log dapat menghambat jika tingkat transaksi per detik (TPS) terlalu rendah untuk PutLogEvents Konfigurasi CloudWatch throttling bersifat global untuk semua layanan, termasuk EMR Tanpa Server. Untuk informasi selengkapnya, lihat [Bagaimana cara menentukan pembatasan di log saya? CloudWatch](#) pada AWS re:post.

Izin yang diperlukan untuk login dengan CloudWatch

Sebelum pekerjaan Anda dapat mengirim data log ke Amazon CloudWatch, sertakan izin berikut dalam kebijakan izin untuk peran runtime pekerjaan.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogGroups"
      ],
      "Resource": [
        "arn:aws:logs:*:123456789012:*"
      ],
      "Sid": "AllowLOGSDescribeLogGroups"
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:PutLogEvents",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:DescribeLogStreams"
      ],
      "Resource": [
        "arn:aws:logs:*:123456789012:log-group:my-log-group-name:*"
      ],
      "Sid": "AllowLOGSPutLogEvents"
    }
  ]
}
```

AWS CLI

Untuk mengatur Amazon CloudWatch untuk menyimpan log untuk EMR Tanpa Server dari AWS CLI, gunakan `cloudWatchLoggingConfiguration` konfigurasi saat Anda memulai menjalankan

pekerjaan. Untuk melakukan ini, berikan penggantian konfigurasi berikut. Secara opsional, berikan juga nama grup log, nama awalan aliran log, jenis log, dan ARN kunci enkripsi.

Jika Anda tidak menentukan nilai opsional, maka CloudWatch menerbitkan log ke grup log default/aws/emr-serverless, dengan aliran `/applications/applicationId/jobs/jobId/worker-type` log default.

EMR Serverless merilis 7.1.0 dan mendukung upaya coba lagi yang lebih tinggi untuk pekerjaan streaming dan pekerjaan batch. Jika Anda mengaktifkan percobaan ulang untuk suatu pekerjaan, EMR Tanpa Server secara otomatis menambahkan nomor percobaan ke awalan jalur log, sehingga Anda dapat membedakan dan melacak log dengan lebih baik.

```
'/applications/<applicationId>/jobs/<jobId>/attempts/<attemptNumber>/worker-type'
```

Berikut ini menunjukkan konfigurasi minimum yang diperlukan untuk mengaktifkan CloudWatch pencatatan Amazon dengan pengaturan default untuk EMR Tanpa Server:

```
{
  "monitoringConfiguration": {
    "cloudWatchLoggingConfiguration": {
      "enabled": true
    }
  }
}
```

Contoh berikut menunjukkan semua konfigurasi wajib dan opsional yang menentukan saat Anda mengaktifkan CloudWatch pencatatan Amazon untuk EMR Tanpa Server. `logTypes` nilai yang didukung juga tercantum dalam contoh berikut ini.

```
{
  "monitoringConfiguration": {
    "cloudWatchLoggingConfiguration": {
      "enabled": true, // Required
      "logGroupName": "Example_logGroup", // Optional
      "logStreamNamePrefix": "Example_logStream", // Optional
      "encryptionKeyArn": "key-arn", // Optional
      "logTypes": {
        "SPARK_DRIVER": ["stdout", "stderr"] //List of values
      }
    }
  }
}
```

```
}  
}  
}
```

Secara default, EMR Tanpa Server hanya menerbitkan stdout driver dan stderr log ke CloudWatch. Jika Anda menginginkan log lain, tentukan peran kontainer dan jenis log yang sesuai dengan LogTypes bidang tersebut.

Daftar berikut menunjukkan tipe pekerja yang didukung yang menentukan LogTypes konfigurasi:

Spark

- SPARK_DRIVER : ["STDERR", "STDOUT"]
- SPARK_EXECUTOR : ["STDERR", "STDOUT"]

Hive

- HIVE_DRIVER : ["STDERR", "STDOUT", "HIVE_LOG", "TEZ_AM"]
- TEZ_TASK : ["STDERR", "STDOUT", "SYSTEM_LOGS"]

Memutar log

Amazon EMR Serverless dapat memutar log aplikasi Spark dan log peristiwa. Rotasi log membantu masalah pekerjaan yang berjalan lama menghasilkan file log besar yang dapat menghabiskan semua ruang disk Anda. Memutar log membantu Anda menghemat penyimpanan disk dan mengurangi jumlah kegagalan pekerjaan karena Anda tidak memiliki lebih banyak ruang tersisa di disk Anda.

Rotasi log diaktifkan secara default dan hanya tersedia untuk pekerjaan Spark.

Log peristiwa percikan

Note

Rotasi log peristiwa percikan tersedia di semua label rilis Amazon EMR.

Alih-alih menghasilkan satu file log peristiwa, EMR Serverless memutar log peristiwa pada interval waktu reguler dan menghapus file log peristiwa yang lebih lama. Memutar log tidak memengaruhi log yang diunggah ke bucket S3.

Log aplikasi percikan

Note

Rotasi log aplikasi percikan tersedia di semua label rilis Amazon EMR.

EMR Serverless juga memutar log aplikasi spark untuk driver dan pelaksana, seperti dan file. `stdout stderr` Anda dapat mengakses file log terbaru dengan memilih tautan log di Studio dengan menggunakan tautan Spark History Server dan Live UI. File log adalah versi terpotong dari log terbaru. Untuk merujuk ke log yang diputar yang lebih lama, tentukan lokasi Amazon S3 saat menyimpan log. Lihat [Logging untuk EMR Tanpa Server dengan bucket Amazon S3 untuk informasi selengkapnya](#).

Anda dapat menemukan file log terbaru di lokasi berikut. EMR Serverless menyegarkan file setiap 15 detik. File-file ini dapat berkisar dari 0 MB hingga 128 MB.

```
<example-S3-logUri>/applications/<application-id>/jobs/<job-id>/SPARK_DRIVER/stderr.gz
```

Lokasi berikut berisi file yang diputar yang lebih lama. Setiap file berukuran 128 MB.

```
<example-S3-logUri>/applications/<application-id>/jobs/<job-id>/SPARK_DRIVER/archived/  
stderr_<index>.gz
```

Perilaku yang sama berlaku untuk pelaksana Spark juga. Perubahan ini hanya berlaku untuk logging S3. Rotasi log tidak memperkenalkan perubahan apa pun pada aliran log yang diunggah ke Amazon CloudWatch

EMR Serverless merilis 7.1.0 dan mendukung upaya coba lagi yang lebih tinggi untuk streaming dan pekerjaan batch. Jika Anda mengaktifkan upaya coba lagi dengan pekerjaan Anda, EMR Serverless menambahkan awalan ke jalur log untuk pekerjaan tersebut sehingga Anda dapat melacak dan membedakan log satu sama lain dengan lebih baik. Jalur ini berisi semua log yang diputar.

```
'/applications/<applicationId>/jobs/<jobId>/attempts/<attemptNumber>/'.
```

Mengenkripsi log

Mengenkripsi log EMR Tanpa Server dengan penyimpanan terkelola

Untuk mengenkripsi log dalam penyimpanan terkelola dengan kunci KMS Anda sendiri, gunakan `managedPersistenceMonitoringConfiguration` konfigurasi saat Anda mengirimkan pekerjaan.

```
{
  "monitoringConfiguration": {
    "managedPersistenceMonitoringConfiguration": {
      "encryptionKeyArn": "key-arn"
    }
  }
}
```

Mengenkripsi log EMR Tanpa Server dengan ember Amazon S3

Untuk mengenkripsi log di bucket Amazon S3 Anda dengan kunci KMS Anda sendiri, gunakan `s3MonitoringConfiguration` konfigurasi saat Anda mengirimkan pekerjaan.

```
{
  "monitoringConfiguration": {
    "s3MonitoringConfiguration": {
      "logUri": "s3://amzn-s3-demo-logging-bucket/logs/",
      "encryptionKeyArn": "key-arn"
    }
  }
}
```

Mengenkripsi log EMR Tanpa Server dengan Amazon CloudWatch

Untuk mengenkripsi log di Amazon CloudWatch dengan kunci KMS Anda sendiri, gunakan `cloudWatchLoggingConfiguration` konfigurasi saat Anda mengirimkan pekerjaan.

```
{
  "monitoringConfiguration": {
    "cloudWatchLoggingConfiguration": {
      "enabled": true,

```

```

        "encryptionKeyArn": "key-arn"
      }
    }
  }
}

```

Izin yang diperlukan untuk enkripsi log

Dalam bagian ini

- [Izin pengguna yang diperlukan](#)
- [Izin kunci enkripsi untuk Amazon S3 dan penyimpanan terkelola](#)
- [Izin kunci enkripsi untuk Amazon CloudWatch](#)

Izin pengguna yang diperlukan

Pengguna yang mengirimkan pekerjaan atau melihat log atau aplikasi UIs harus memiliki izin untuk menggunakan kunci. Anda dapat menentukan izin dalam kebijakan kunci KMS atau kebijakan IAM untuk pengguna, grup, atau peran. Jika pengguna yang mengirimkan pekerjaan tidak memiliki izin kunci KMS, EMR Tanpa Server menolak pengiriman job run.

Contoh kebijakan kunci

Kebijakan utama berikut memberikan izin untuk `kms:GenerateDataKey` dan `kms:Decrypt`:

```

{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:user/user-name"
  },
  "Action": [
    "kms:GenerateDataKey",
    "kms:Decrypt"
  ],
  "Resource": "*"
}

```

Contoh kebijakan IAM

Kebijakan IAM berikut memberikan izin untuk `kms:GenerateDataKey` dan `kms:Decrypt`

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ],
      "Resource": [
        "arn:aws:kms:*:123456789012:key/12345678-1234-1234-1234-123456789012"
      ],
      "Sid": "AllowKMSGeneratedatakey"
    }
  ]
}
```

Untuk meluncurkan UI Spark atau Tez, berikan izin kepada pengguna, grup, atau peran Anda untuk mengakses `emr-serverless:GetDashboardForJobRun` API sebagai berikut:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-serverless:GetDashboardForJobRun"
      ],
      "Resource": [
        "*"
      ],
      "Sid": "AllowEMRSERVERLESSGetdashboardforjobrun"
    }
  ]
}
```

Izin kunci enkripsi untuk Amazon S3 dan penyimpanan terkelola

Saat Anda mengenkripsi log dengan kunci enkripsi Anda sendiri baik di penyimpanan terkelola atau di bucket S3 Anda, konfigurasi izin kunci KMS sebagai berikut.

`emr-serverless.amazonaws.com` Kepala sekolah harus memiliki izin berikut dalam kebijakan untuk kunci KMS:

```
{
  "Effect": "Allow",
  "Principal": {
    "Service": "emr-serverless.amazonaws.com"
  },
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey"
  ],
  "Resource": "*"
  "Condition": {
    "StringLike": {
      "aws:SourceArn": "arn:aws:emr-serverless:region:aws-account-id:/
applications/application-id"
    }
  }
}
```

Sebagai praktik keamanan terbaik, kami menyarankan Anda menambahkan kunci `aws:SourceArn` kondisi ke kebijakan kunci KMS. Kunci kondisi global IAM `aws:SourceArn` membantu memastikan bahwa EMR Tanpa Server menggunakan kunci KMS hanya untuk ARN aplikasi.

Peran runtime pekerjaan harus memiliki izin berikut dalam kebijakan IAM-nya:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:GenerateDataKey",
```

```

    "kms:Decrypt"
  ],
  "Resource": [
    "arn:aws:kms:*:123456789012:key/12345678-1234-1234-1234-123456789012"
  ],
  "Sid": "AllowKMSGeneratedatakey"
}
]
}

```

Izin kunci enkripsi untuk Amazon CloudWatch

Untuk mengaitkan ARN kunci KMS ke grup log Anda, gunakan kebijakan IAM berikut untuk peran runtime pekerjaan.

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:AssociateKmsKey"
      ],
      "Resource": [
        "arn:aws:logs:*:123456789012:log-group:my-log-group-name:*"
      ],
      "Sid": "AllowLOGSAssociatekmskey"
    }
  ]
}

```

Konfigurasi kebijakan kunci KMS untuk memberikan izin KMS ke Amazon: CloudWatch

JSON

```

{
  "Version": "2012-10-17",

```

```
"Id": "key-default-1",
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "kms:Decrypt",
      "kms:GenerateDataKey"
    ],
    "Resource": [
      "*"
    ],
    "Condition": {
      "ArnLike": {
        "kms:EncryptionContext:aws:logs:arn": "arn:aws:logs:*:123456789012:*"
      }
    },
    "Sid": "AllowKMSDecrypt"
  }
]
}
```

Konfigurasi properti Apache Log4j2 untuk Amazon EMR Tanpa Server

Halaman ini menjelaskan cara mengonfigurasi [properti Apache Log4j 2.x](#) kustom untuk pekerjaan EMR Tanpa Server di. StartJobRun Jika Anda ingin mengonfigurasi klasifikasi Log4j di tingkat aplikasi, lihat. [Konfigurasi aplikasi default untuk EMR Serverless](#)

Konfigurasi properti Spark Log4j2 untuk Amazon EMR Tanpa Server

Dengan Amazon EMR merilis 6.8.0 dan yang lebih tinggi, Anda dapat menyesuaikan properti [Apache Log4j 2.x untuk menentukan konfigurasi log](#) butir halus. Ini menyederhanakan pemecahan masalah pekerjaan Spark Anda di EMR Tanpa Server. Untuk mengkonfigurasi properti ini, gunakan `spark-driver-log4j2` dan `spark-executor-log4j2` klasifikasi.

Topik

- [Klasifikasi Log4j2 untuk Spark](#)
- [Contoh konfigurasi Log4j2 untuk Spark](#)
- [Log4j2 dalam contoh pekerjaan Spark](#)

- [Pertimbangan Log4j2 untuk Spark](#)

Klasifikasi Log4j2 untuk Spark

Untuk menyesuaikan konfigurasi log Spark, gunakan klasifikasi berikut dengan.

[applicationConfiguration](#) Untuk mengkonfigurasi properti Log4j 2.x, gunakan yang berikut ini.
[properties](#)

spark-driver-log4j2

Klasifikasi ini menetapkan nilai dalam `log4j2.properties` file untuk driver.

spark-executor-log4j2

Klasifikasi ini menetapkan nilai dalam `log4j2.properties` file untuk pelaksana.

Contoh konfigurasi Log4j2 untuk Spark

Contoh berikut menunjukkan cara mengirimkan pekerjaan Spark dengan untuk menyesuaikan konfigurasi Log4j2 `applicationConfiguration` untuk driver dan eksekutor Spark.

Untuk mengonfigurasi klasifikasi Log4j di tingkat aplikasi, bukan saat Anda mengirimkan pekerjaan, lihat. [Konfigurasi aplikasi default untuk EMR Serverless](#)

```
aws emr-serverless start-job-run \  
  --application-id application-id \  
  --execution-role-arn job-role-arn \  
  --job-driver '{  
    "sparkSubmit": {  
      "entryPoint": "/usr/lib/spark/examples/jars/spark-examples.jar",  
      "entryPointArguments": ["1"],  
      "sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi --conf  
spark.executor.cores=4 --conf spark.executor.memory=20g --conf spark.driver.cores=4 --  
conf spark.driver.memory=8g --conf spark.executor.instances=1"  
    }  
  }'  
  --configuration-overrides '{  
    "applicationConfiguration": [  
      {  
        "classification": "spark-driver-log4j2",  
        "properties": {
```

```

        "rootLogger.level":"error", // will only display Spark error logs
        "logger.IdentifierForClass.name": "classpath for setting logger",
        "logger.IdentifierForClass.level": "info"
    }
},
{
    "classification": "spark-executor-log4j2",
    "properties": {
        "rootLogger.level":"error", // will only display Spark error logs
        "logger.IdentifierForClass.name": "classpath for setting logger",
        "logger.IdentifierForClass.level": "info"
    }
}
]
}'

```

Log4j2 dalam contoh pekerjaan Spark

Contoh kode berikut menunjukkan cara membuat aplikasi Spark saat Anda menginisialisasi konfigurasi Log4j2 kustom untuk aplikasi.

Python

Example- Menggunakan Log4j2 untuk pekerjaan Spark dengan Python

```

import os
import sys

from pyspark import SparkConf, SparkContext
from pyspark.sql import SparkSession

app_name = "PySparkApp"
if __name__ == "__main__":
    spark = SparkSession\
        .builder\
        .appName(app_name)\
        .getOrCreate()

    spark.sparkContext._conf.getAll()
    sc = spark.sparkContext
    log4jLogger = sc._jvm.org.apache.log4j
    LOGGER = log4jLogger.LogManager.getLogger(app_name)

```

```
LOGGER.info("pyspark script logger info")
LOGGER.warn("pyspark script logger warn")
LOGGER.error("pyspark script logger error")

// your code here

spark.stop()
```

Untuk menyesuaikan Log4j2 untuk driver saat Anda menjalankan pekerjaan Spark, gunakan konfigurasi berikut:

```
{
  "classification": "spark-driver-log4j2",
  "properties": {
    "rootLogger.level": "error", // only display Spark error logs
    "logger.PySparkApp.level": "info",
    "logger.PySparkApp.name": "PySparkApp"
  }
}
```

Scala

Example- Menggunakan Log4j2 untuk pekerjaan Spark dengan Scala

```
import org.apache.log4j.Logger
import org.apache.spark.sql.Session

object ExampleClass {
  def main(args: Array[String]): Unit = {
    val spark = SparkSession
      .builder
      .appName(this.getClass.getName)
      .getOrCreate()

    val logger = Logger.getLogger(this.getClass);
    logger.info("script logging info logs")
    logger.warn("script logging warn logs")
    logger.error("script logging error logs")

    // your code here
    spark.stop()
  }
}
```

```
}  
}
```

Untuk menyesuaikan Log4j2 untuk driver saat Anda menjalankan pekerjaan Spark, gunakan konfigurasi berikut:

```
{  
  "classification": "spark-driver-log4j2",  
  "properties": {  
    "rootLogger.level": "error", // only display Spark error logs  
    "logger.ExampleClass.level": "info",  
    "logger.ExampleClass.name": "ExampleClass"  
  }  
}
```

Pertimbangan Log4j2 untuk Spark

Properti Log4j2.x berikut tidak dapat dikonfigurasi untuk proses Spark:

- `rootLogger.appenderRef.stdout.ref`
- `appender.console.type`
- `appender.console.name`
- `appender.console.target`
- `appender.console.layout.type`
- `appender.console.layout.pattern`

[Untuk informasi rinci tentang properti Log4j2.x yang dikonfigurasi, lihat file di `log4j2.properties.template` GitHub](#)

Pemantauan EMR Tanpa Server

Bagian ini mencakup cara-cara yang memantau aplikasi dan pekerjaan Tanpa Server Amazon EMR Anda.

Topik

- [Memantau aplikasi dan pekerjaan EMR Tanpa Server](#)

- [Pantau metrik Spark dengan Amazon Managed Service untuk Prometheus](#)
- [Metrik penggunaan EMR Tanpa Server](#)

Memantau aplikasi dan pekerjaan EMR Tanpa Server

Dengan CloudWatch metrik Amazon untuk EMR Tanpa Server, Anda dapat menerima metrik CloudWatch 1 menit dan CloudWatch mengakses dasbor untuk mengakses near-real-time operasi dan kinerja aplikasi EMR Tanpa Server Anda.

EMR Tanpa Server mengirimkan metrik ke setiap menit. CloudWatch EMR Tanpa Server memancarkan metrik ini di tingkat aplikasi serta pekerjaan, tipe pekerja, dan level. `capacity-allocation-type`

Untuk memulai, gunakan templat CloudWatch dasbor EMR Tanpa Server yang disediakan di repositori [EMR](#) Tanpa Server dan terapkan. [GitHub](#)

Note

[EMR Beban kerja interaktif tanpa server](#) hanya mengaktifkan pemantauan tingkat aplikasi, dan memiliki dimensi tipe pekerja baru, `Spark_Kernel` Untuk memantau dan men-debug beban kerja interaktif Anda, akses log dan Apache Spark UI dari dalam [EMR Studio Workspace Anda](#).

Metrik pemantauan

Important

Kami merestrukturisasi tampilan metrik kami untuk ditambahkan `ApplicationName` dan `JobName` sebagai dimensi. Untuk rilis 7.10 dan yang lebih baru, metrik lama tidak akan lagi diperbarui. Untuk rilis EMR di bawah 7.10, metrik lama masih tersedia.

Dimensi saat ini

Tabel di bawah ini menjelaskan dimensi EMR Tanpa Server yang tersedia dalam namespace. `AWS/EMR_Serverless`

Dimensi untuk metrik EMR Tanpa Server

Dimensi	Deskripsi	
ApplicationId	Filter untuk semua metrik aplikasi EMR Tanpa Server menggunakan ID aplikasi.	
ApplicationName	Filter untuk semua metrik aplikasi EMR Tanpa Server menggunakan nama. Jika nama tidak diberikan, atau berisi karakter non-ASCII, itu diterbitkan sebagai [Tidak ditentukan].	
JobId	Filter untuk semua metrik EMR Tanpa Server ID job run.	
JobName	Filter untuk semua metrik pekerjaan EMR Tanpa Server yang dijalankan menggunakan nama. Jika nama tidak diberikan, atau berisi karakter non-ASCII, itu diterbitkan sebagai [Tidak ditentukan].	
WorkerType	Filter untuk semua metrik dari jenis pekerja tertentu. Misalnya, Anda dapat memfilter untuk SPARK_DRIVER dan SPARK_EXECUTORS untuk pekerjaan Spark.	
CapacityAllocationType	Filter untuk semua metrik dari jenis alokasi kapasitas tertentu. Misalnya, Anda dapat memfilter PreInitCa	

Dimensi	Deskripsi
	<p><code>capacity</code> untuk kapasitas pra-inisialisasi dan <code>OnDemandCapacity</code> untuk yang lainnya.</p>

Pemantauan tingkat aplikasi

Anda dapat memantau penggunaan kapasitas di tingkat aplikasi EMR Tanpa Server dengan metrik Amazon CloudWatch. Anda juga dapat mengatur satu tampilan untuk memantau penggunaan kapasitas aplikasi di CloudWatch dasbor.

Metrik aplikasi EMR Tanpa Server

Metrik	Deskripsi	Unit	Dimensi
<code>MaxCPUAllowed</code>	CPU maksimum yang diizinkan untuk aplikasi.	vCPU	<code>ApplicationId</code> , <code>ApplicationName</code>
<code>MaxMemoryAllowed</code>	Memori maksimum dalam GB diperbolehkan untuk aplikasi.	Gigabyte (GB)	<code>ApplicationId</code> , <code>ApplicationName</code>
<code>MaxStorageAllowed</code>	Penyimpanan maksimum dalam GB diperbolehkan untuk aplikasi.	Gigabyte (GB)	<code>ApplicationId</code> , <code>ApplicationName</code>
<code>CPULocated</code>	Jumlah total v yang CPUs dialokasikan.	vCPU	<code>ApplicationId</code> , <code>ApplicationName</code> , <code>WorkerType</code> , <code>CapacityAllocationType</code>
<code>IdleWorkerCount</code>	Jumlah total pekerja yang menganggur.	Hitungan	<code>ApplicationId</code> , <code>ApplicationName</code> , <code>WorkerType</code>

Metrik	Deskripsi	Unit	Dimensi
			ApplicationId , CapacityAllocationType
MemoryAllocated	Total memori dalam GB dialokasikan.	Gigabyte (GB)	ApplicationId , ApplicationName , WorkerType , CapacityAllocationType
PendingCreationWorkerCount	Jumlah total pekerja yang menunggu penciptaan.	Hitungan	ApplicationId , ApplicationName , WorkerType , CapacityAllocationType
RunningWorkerCount	Jumlah total pekerja yang digunakan oleh aplikasi.	Hitungan	ApplicationId , ApplicationName , WorkerType , CapacityAllocationType
StorageAllocated	Total penyimpanan disk dalam GB dialokasikan.	Gigabyte (GB)	ApplicationId , ApplicationName , WorkerType , CapacityAllocationType
TotalWorkerCount	Jumlah total pekerja yang tersedia.	Hitungan	ApplicationId , ApplicationName , WorkerType , CapacityAllocationType

Pemantauan tingkat pekerjaan

Amazon EMR Tanpa Server mengirimkan metrik tingkat pekerjaan berikut ke setiap satu menit. Amazon CloudWatch Anda dapat mengakses nilai metrik untuk pekerjaan agregat yang dijalankan berdasarkan status menjalankan pekerjaan. Unit untuk setiap metrik adalah hitungan.

EMR Metrik tingkat pekerjaan tanpa server

Metrik	Deskripsi	Dimensi
SubmittedJobs	Jumlah pekerjaan di negara bagian yang Dikirim.	ApplicationId , ApplicationName
PendingJobs	Jumlah pekerjaan dalam keadaan Tertunda.	ApplicationId , ApplicationName
ScheduledJobs	Jumlah pekerjaan dalam keadaan Terjadwal.	ApplicationId , ApplicationName
RunningJobs	Jumlah pekerjaan dalam keadaan Running.	ApplicationId , ApplicationName
SuccessJobs	Jumlah pekerjaan dalam keadaan Sukses.	ApplicationId , ApplicationName
FailedJobs	Jumlah pekerjaan dalam keadaan Gagal.	ApplicationId , ApplicationName
CancellingJobs	Jumlah pekerjaan di negara Membatalkan.	ApplicationId , ApplicationName
CancelledJobs	Jumlah pekerjaan di negara yang Dibatalkan.	ApplicationId , ApplicationName

Anda dapat memantau metrik khusus mesin untuk menjalankan dan menyelesaikan pekerjaan EMR Tanpa Server dengan aplikasi khusus mesin. UI Saat Anda mengakses UI untuk pekerjaan yang sedang berjalan, UI aplikasi langsung akan ditampilkan dengan pembaruan waktu nyata. Saat Anda mengakses UI untuk pekerjaan yang diselesaikan, UI aplikasi persisten akan ditampilkan.

Menjalankan pekerjaan

Untuk menjalankan tugas EMR Tanpa Server Anda, akses antarmuka real-time yang menyediakan metrik khusus mesin. Anda dapat menggunakan Apache Spark UI atau Hive Tez UI untuk memantau dan men-debug pekerjaan Anda. Untuk mengakses ini UIs, gunakan konsol EMR Studio atau minta titik akhir URL aman dengan. AWS Command Line Interface

Pekerjaan yang diselesaikan

Untuk pekerjaan EMR Tanpa Server Anda yang telah selesai, gunakan Spark History Server atau Persistent Hive Tez UI untuk mengakses detail pekerjaan, tahapan, tugas, dan metrik untuk menjalankan pekerjaan Spark atau Hive. Untuk mengakses ini UIs, gunakan konsol EMR Studio, atau minta titik akhir URL aman dengan. AWS Command Line Interface

Pemantauan tingkat pekerja Job

Amazon EMR Tanpa Server mengirimkan metrik tingkat pekerja kerja berikut yang tersedia di `AWS/EMRServerless` namespace dan grup metrik ke Amazon. `Job Worker Metrics` CloudWatch EMR Tanpa Server mengumpulkan poin data dari pekerja individu selama pekerjaan berjalan di tingkat pekerjaan, tipe pekerja, dan tingkat. `capacity-allocation-type` Anda dapat menggunakan `ApplicationId` sebagai dimensi untuk memantau beberapa pekerjaan yang termasuk dalam aplikasi yang sama.

Note

Untuk melihat total CPU dan Memori yang digunakan oleh pekerjaan EMR Tanpa Server saat melihat metrik di konsol CloudWatch Amazon, gunakan Statistik sebagai Jumlah dan Periode sebagai 1 menit.

EMR Metrik tingkat pekerja pekerjaan tanpa server

Metrik	Deskripsi	Unit	Dimensi
<code>WorkerCpuAllocated</code>	Jumlah total inti vCPU yang dialokasikan untuk pekerja dalam menjalankan pekerjaan.	vCPU	<code>JobId</code> , <code>JobName</code> , <code>ApplicationId</code> , <code>ApplicationName</code> , <code>WorkerType</code> ,

Metrik	Deskripsi	Unit	Dimensi
			dan CapacityAllocation Type
WorkerCpuUsed	Jumlah total core vCPU yang digunakan oleh pekerja dalam menjalankan pekerjaan.	vCPU	JobId, JobName, ApplicationId, ApplicationName, WorkerType, dan CapacityAllocation Type
WorkerMemoryAllocated	Total memori dalam GB dialokasikan untuk pekerja dalam menjalankan pekerjaan.	Gigabyte (GB)	JobId, JobName, ApplicationId, ApplicationName, WorkerType, dan CapacityAllocation Type

Metrik	Deskripsi	Unit	Dimensi
WorkerMemoryUsed	Total memori dalam GB yang digunakan oleh pekerja dalam menjalankan pekerjaan.	Gigabyte (GB)	JobId, JobName, ApplicationId, ApplicationName, WorkerType, dan CapacityAllocationType
WorkerEphemeralStorageAllocated	Jumlah byte penyimpanan sementara yang dialokasikan untuk pekerja dalam menjalankan pekerjaan.	Gigabyte (GB)	JobId, JobName, ApplicationId, ApplicationName, WorkerType, dan CapacityAllocationType
WorkerEphemeralStorageUsed	Jumlah byte penyimpanan sementara yang digunakan oleh pekerja dalam menjalankan pekerjaan.	Gigabyte (GB)	JobId, JobName, ApplicationId, ApplicationName, WorkerType, dan CapacityAllocationType

Metrik	Deskripsi	Unit	Dimensi
WorkerStorageReadBytes	Jumlah byte yang dibaca dari penyimpanan oleh pekerja dalam menjalankan pekerjaan.	Byte	JobId, JobName, ApplicationId, ApplicationName, WorkerType, dan CapacityAllocationType
WorkerStorageWriteBytes	Jumlah byte yang ditulis ke penyimpanan dari pekerja dalam menjalankan pekerjaan.	Byte	JobId, JobName, ApplicationId, ApplicationName, WorkerType, dan CapacityAllocationType

Langkah-langkah di bawah ini menjelaskan cara mengakses berbagai jenis metrik.

Console

Untuk mengakses UI aplikasi Anda dengan konsol

1. [Arahkan ke aplikasi EMR Tanpa Server Anda di EMR Studio dengan petunjuk di Memulai dari konsol.](#)
2. Untuk mengakses aplikasi UIs dan log khusus mesin untuk pekerjaan yang sedang berjalan:
 - a. Pilih pekerjaan dengan RUNNING status.
 - b. Pilih pekerjaan di halaman Detail aplikasi, atau navigasikan ke halaman Detail Pekerjaan untuk pekerjaan Anda.

- c. Di bawah menu tarik-turun Display UI, pilih Spark UI atau Hive Tez UI untuk menavigasi ke UI aplikasi untuk jenis pekerjaan Anda.
 - d. Untuk mengakses log mesin Spark, navigasikan ke tab Executors di UI Spark, dan pilih tautan Log untuk driver. Untuk mengakses log mesin Hive, pilih tautan Log untuk DAG yang sesuai di UI Hive Tez.
3. Untuk mengakses aplikasi UIs dan log khusus mesin untuk pekerjaan yang diselesaikan:
- a. Pilih pekerjaan dengan SUCCESS status.
 - b. Pilih pekerjaan di halaman detail Aplikasi lamaran Anda atau navigasikan ke halaman detail Pekerjaan.
 - c. Di bawah menu tarik-turun Display UI, pilih Spark History Server atau Persistent Hive Tez UI untuk menavigasi ke UI aplikasi untuk jenis pekerjaan Anda.
 - d. Untuk mengakses log mesin Spark, navigasikan ke tab Executors di UI Spark, dan pilih tautan Log untuk driver. Untuk mengakses log mesin Hive, pilih tautan Log untuk DAG yang sesuai di UI Hive Tez.

AWS CLI

Untuk mengakses UI aplikasi Anda dengan AWS CLI

- Untuk menghasilkan URL yang digunakan untuk mengakses UI aplikasi Anda untuk menjalankan dan menyelesaikan pekerjaan, hubungi GetDashboardForJobRun API.

```
aws emr-serverless get-dashboard-for-job-run /  
--application-id <application-id> /  
--job-run-id <job-id>
```

URL yang Anda hasilkan valid selama satu jam.

Pantau metrik Spark dengan Amazon Managed Service untuk Prometheus

Dengan Amazon EMR rilis 7.1.0 dan yang lebih tinggi, Anda dapat mengintegrasikan EMR Tanpa Server dengan Amazon Managed Service untuk Prometheus guna mengumpulkan metrik Apache Spark untuk pekerjaan dan aplikasi EMR Tanpa Server. Integrasi ini tersedia saat Anda mengirimkan pekerjaan atau membuat aplikasi menggunakan AWS konsol, EMR Serverless API, atau AWS CLI

Prasyarat

Sebelum Anda dapat mengirimkan metrik Spark Anda ke Amazon Managed Service untuk Prometheus, lengkapi prasyarat berikut.

- [Buat Layanan Terkelola Amazon untuk ruang kerja Prometheus](#). Ruang kerja ini berfungsi sebagai titik akhir konsumsi. Catat URL yang ditampilkan untuk Endpoint - URL tulis jarak jauh. Anda harus menentukan URL saat membuat aplikasi EMR Tanpa Server.
- Untuk memberikan akses pekerjaan Anda ke Amazon Managed Service untuk Prometheus untuk tujuan pemantauan, tambahkan kebijakan berikut ke peran pelaksanaan pekerjaan Anda.

```
{
  "Sid": "AccessToPrometheus",
  "Effect": "Allow",
  "Action": ["aps:RemoteWrite"],
  "Resource": "arn:aws:aps:<AWS_REGION>:<AWS_ACCOUNT_ID>:workspace/<WORKSPACE_ID>"
}
```

Pengaturan

Untuk menggunakan AWS konsol untuk membuat aplikasi yang terintegrasi dengan Amazon Managed Service untuk Prometheus

1. Lihat [Memulai Amazon EMR Tanpa Server untuk membuat aplikasi](#).
2. Saat Anda membuat aplikasi, pilih Gunakan pengaturan khusus, lalu konfigurasi aplikasi Anda dengan menentukan informasi ke dalam bidang yang ingin Anda konfigurasi.
3. Di bawah Log dan metrik aplikasi, pilih Mengirimkan metrik engine ke Amazon Managed Service for Prometheus, lalu tentukan URL penulisan jarak jauh Anda.
4. Tentukan pengaturan konfigurasi lain yang Anda inginkan, lalu pilih Buat dan mulai aplikasi.

Gunakan API Tanpa AWS CLI Server EMR atau EMR

Anda juga dapat menggunakan API Tanpa Server EMR AWS CLI atau EMR untuk mengintegrasikan aplikasi EMR Tanpa Server Anda dengan Amazon Managed Service for Prometheus saat menjalankan atau perintah. `create-application start-job-run`

create-application

```
aws emr-serverless create-application \  
--release-label emr-7.1.0 \  
--type "SPARK" \  
--monitoring-configuration '{  
    "prometheusMonitoringConfiguration": {  
        "remoteWriteUrl": "https://aps-workspaces.<AWS_REGION>.amazonaws.com/  
workspaces/<WORKSPACE_ID>/api/v1/remote_write"  
    }  
'
```

start-job-run

```
aws emr-serverless start-job-run \  
--application-id <APPLICATION_ID> \  
--execution-role-arn <JOB_EXECUTION_ROLE> \  
--job-driver '{  
    "sparkSubmit": {  
        "entryPoint": "local:///usr/lib/spark/examples/src/main/python/pi.py",  
        "entryPointArguments": ["10000"],  
        "sparkSubmitParameters": "--conf spark.dynamicAllocation.maxExecutors=10"  
    }  
' \  
--configuration-overrides '{  
    "monitoringConfiguration": {  
        "prometheusMonitoringConfiguration": {  
            "remoteWriteUrl": "https://aps-workspaces.<AWS_REGION>.amazonaws.com/  
workspaces/<WORKSPACE_ID>/api/v1/remote_write"  
        }  
    }  
'
```

Termasuk `prometheusMonitoringConfiguration` dalam perintah Anda menunjukkan bahwa EMR Tanpa Server harus menjalankan pekerjaan Spark dengan agen yang mengumpulkan metrik Spark dan menuliskannya ke titik `remoteWriteUrl` akhir Anda untuk Amazon Managed Service for Prometheus. Anda kemudian dapat menggunakan metrik Spark di Amazon Managed Service for Prometheus untuk visualisasi, peringatan, dan analisis.

Properti konfigurasi lanjutan

EMR Tanpa Server menggunakan komponen dalam nama Spark `PrometheusServlet` untuk mengumpulkan metrik Spark dan menerjemahkan data kinerja ke dalam data yang kompatibel dengan Amazon Managed Service for Prometheus. Secara default, EMR Tanpa Server menetapkan nilai default di Spark dan mengurai metrik driver dan pelaksana saat Anda mengirimkan pekerjaan menggunakan `PrometheusMonitoringConfiguration`

Tabel berikut menjelaskan semua properti yang dikonfigurasi saat mengirimkan pekerjaan Spark yang mengirimkan metrik ke Amazon Managed Service untuk Prometheus.

Properti percikan	Nilai default	Deskripsi
<code>spark.metrics.conf</code> <code>.*.sink.prometheusServlet.class</code>	<code>org.apache.spark.metrics.sink.PrometheusServlet</code>	Kelas yang digunakan Spark untuk mengirim metrik ke Amazon Managed Service untuk Prometheus. Untuk mengganti perilaku default, tentukan kelas kustom Anda sendiri.
<code>spark.metrics.conf</code> <code>.*.source.jvm.class</code>	<code>org.apache.spark.metrics.source.JvmSource</code>	Kelas Spark digunakan untuk mengumpulkan dan mengirim metrik penting dari mesin virtual Java yang mendasarinya. Untuk berhenti mengumpulkan metrik JVM, nonaktifkan properti ini dengan menyetelnya ke string kosong, seperti. "" Untuk mengganti perilaku default, tentukan kelas kustom Anda sendiri.
<code>spark.metrics.conf</code> <code>.driver.sink.prometheusServlet.path</code>	<code>/metrik/prometheus</code>	URL berbeda yang digunakan Amazon Managed Service untuk Prometheus untuk mengumpulkan metrik dari driver. Untuk mengganti

Properti percikan	Nilai default	Deskripsi
		perilaku default, tentukan jalur Anda sendiri. Untuk berhenti mengumpulkan metrik driver, nonaktifkan properti ini dengan menyetelnya ke string kosong, seperti "".
<code>spark.metrics.conf.executor.sink.prometheusServlet.path</code>	<code>/metrics/executor/prometheus</code>	URL berbeda yang digunakan Amazon Managed Service untuk Prometheus untuk mengumpulkan metrik dari pelaksana. Untuk mengganti perilaku default, tentukan jalur Anda sendiri. Untuk berhenti mengumpulkan metrik pelaksana, nonaktifkan properti ini dengan menyetelnya ke string kosong, seperti "".

Untuk informasi selengkapnya tentang metrik Spark, lihat metrik [Apache Spark](#).

Pertimbangan dan batasan

Saat menggunakan Amazon Managed Service for Prometheus untuk mengumpulkan metrik dari EMR Tanpa Server, pertimbangkan pertimbangan dan batasan berikut.

- Dukungan untuk menggunakan Amazon Managed Service untuk Prometheus dengan EMR Serverless hanya tersedia di [tempat Amazon Managed Service untuk Prometheus Wilayah AWS umumnya](#) tersedia.
- Menjalankan agen untuk mengumpulkan metrik Spark di Amazon Managed Service untuk Prometheus membutuhkan lebih banyak sumber daya dari pekerja. Jika Anda memilih ukuran pekerja yang lebih kecil, seperti satu pekerja vCPU, waktu kerja Anda mungkin meningkat.
- Dukungan untuk menggunakan Amazon Managed Service untuk Prometheus dengan EMR Tanpa Server hanya tersedia untuk Amazon EMR rilis 7.1.0 dan yang lebih tinggi.

- Layanan Terkelola Amazon untuk Prometheus harus diterapkan di akun yang sama tempat Anda menjalankan EMR Tanpa Server untuk mengumpulkan metrik.

Metrik penggunaan EMR Tanpa Server

Anda dapat menggunakan metrik CloudWatch penggunaan Amazon untuk memberikan visibilitas ke sumber daya yang digunakan akun Anda. Gunakan metrik ini untuk memvisualisasikan penggunaan layanan Anda pada CloudWatch grafik dan dasbor.

Metrik penggunaan EMR Tanpa Server sesuai dengan Service Quotas. Anda dapat mengonfigurasi alarm yang memberi tahu Anda saat penggunaan mendekati kuota layanan. Untuk informasi selengkapnya, lihat [Service Quotas dan CloudWatch alarm Amazon](#) di Panduan Pengguna Service Quotas.

Untuk informasi lebih lanjut tentang kuota layanan EMR Tanpa Server, lihat [Titik akhir dan kuota untuk EMR Serverless](#)

Metrik penggunaan kuota layanan untuk EMR Tanpa Server

EMR Tanpa Server menerbitkan metrik penggunaan kuota layanan berikut di namespace. `AWS/Usage`

Metrik	Deskripsi
<code>ResourceCount</code>	Jumlah total sumber daya yang ditentukan yang berjalan di akun Anda. Sumber daya ditentukan oleh dimensi yang terkait dengan metrik.

Dimensi untuk metrik penggunaan kuota layanan EMR Tanpa Server

Anda dapat menggunakan dimensi berikut untuk menyempurnakan metrik penggunaan yang diterbitkan EMR Tanpa Server.

Dimensi	Nilai	Deskripsi
<code>Service</code>	EMR Tanpa Server	Nama yang Layanan AWS berisi sumber daya.

Dimensi	Nilai	Deskripsi
Type	Sumber daya	Jenis entitas yang EMR Serverless melaporkan.
Resource	vCPU	Jenis sumber daya yang dilacak EMR Tanpa Server.
Class	Tidak ada	Kelas sumber daya yang dilacak EMR Tanpa Server.

Mengotomatisasi EMR Tanpa Server dengan Amazon EventBridge

Anda dapat menggunakannya Amazon EventBridge untuk mengotomatiskan Layanan AWS dan merespons secara otomatis peristiwa sistem, seperti masalah ketersediaan aplikasi atau perubahan sumber daya. EventBridge memberikan aliran peristiwa sistem yang mendekati real-time yang menggambarkan perubahan dalam AWS sumber daya Anda. Anda dapat menulis aturan sederhana untuk menunjukkan kejadian mana yang sesuai kepentingan Anda, dan tindakan otomatis apa yang diambil ketika suatu kejadian sesuai dengan suatu aturan. Dengan EventBridge, Anda dapat secara otomatis:

- Memanggil fungsi AWS Lambda
- Relay peristiwa ke Amazon Kinesis Data Streams
- Aktifkan mesin AWS Step Functions negara
- Beri tahu topik Amazon SNS atau antrian Amazon SQS

Misalnya, saat Anda menggunakan EventBridge EMR Tanpa Server, Anda dapat mengaktifkan AWS Lambda fungsi saat pekerjaan ETL berhasil atau memberi tahu topik Amazon SNS saat pekerjaan ETL gagal.

EMR Tanpa Server memancarkan empat jenis acara:

- Peristiwa perubahan status aplikasi — Peristiwa yang memancarkan setiap perubahan status aplikasi. Untuk informasi lebih lanjut tentang status aplikasi, lihat [Status aplikasi](#).
- Job run state change events — Peristiwa yang memancarkan setiap perubahan status dari pekerjaan yang dijalankan. Untuk informasi lebih lanjut tentang, lihat [Status tugas berjalan](#).

- Job run retry events - Acara yang memancarkan setiap percobaan ulang pekerjaan yang dijalankan dari Amazon EMR Serverless rilis 7.1.0 dan yang lebih tinggi.
- Acara pembaruan pemanfaatan sumber daya pekerjaan - Acara yang memancarkan pembaruan pemanfaatan sumber daya untuk pekerjaan yang dijalankan dengan interval hampir 30 menit.

Contoh acara EMR Tanpa Server EventBridge

Peristiwa yang dilaporkan oleh EMR Tanpa Server memiliki nilai yang `aws.emr-serverless` ditetapkan `source`, seperti pada contoh berikut.

Acara perubahan status aplikasi

Contoh peristiwa berikut menunjukkan aplikasi di CREATING negara bagian.

```
{
  "version": "0",
  "id": "9fd3cf79-1ff1-b633-4dd9-34508dc1e660",
  "detail-type": "EMR Serverless Application State Change",
  "source": "aws.emr-serverless",
  "account": "123456789012",
  "time": "2022-05-31T21:16:31Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
    "applicationId": "00f1cb5c6anuij25",
    "applicationName": "3965ad00-8fba-4932-a6c8-ded32786fd42",
    "arn": "arn:aws:emr-serverless:us-east-1:111122223333:/
applications/00f1cb5c6anuij25",
    "releaseLabel": "emr-6.6.0",
    "state": "CREATING",
    "type": "HIVE",
    "createdAt": "2022-05-31T21:16:31.547953Z",
    "updatedAt": "2022-05-31T21:16:31.547970Z",
    "autoStopConfig": {
      "enabled": true,
      "idleTimeout": 15
    },
    "autoStartConfig": {
      "enabled": true
    }
  }
}
```

```
}
```

Acara perubahan status Job run

Contoh peristiwa berikut menunjukkan job run yang berpindah dari SCHEDULED state ke RUNNING state.

```
{
  "version": "0",
  "id": "00df3ec6-5da1-36e6-ab71-20f0de68f8a0",
  "detail-type": "EMR Serverless Job Run State Change",
  "source": "aws.emr-serverless",
  "account": "123456789012",
  "time": "2022-05-31T21:07:42Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
    "jobRunId": "00f1cbn5g4bb0c01",
    "applicationId": "00f1982r1uukb925",
    "arn": "arn:aws:emr-serverless:us-east-1:123456789012:/
applications/00f1982r1uukb925/jobruns/00f1cbn5g4bb0c01",
    "releaseLabel": "emr-6.6.0",
    "state": "RUNNING",
    "previousState": "SCHEDULED",
    "createdBy": "arn:aws:sts::123456789012:assumed-role/
TestRole-402dcef3ad14993c15d28263f64381e4cda34775/6622b6233b6d42f59c25dd2637346242",
    "updatedAt": "2022-05-31T21:07:42.299487Z",
    "createdAt": "2022-05-31T21:07:25.325900Z"
  }
}
```

Job run acara coba lagi

Berikut ini adalah contoh acara coba ulang job run.

```
{
  "version": "0",
  "id": "00df3ec6-5da1-36e6-ab71-20f0de68f8a0",
  "detail-type": "EMR Serverless Job Run Retry",
  "source": "aws.emr-serverless",
  "account": "123456789012",
  "time": "2022-05-31T21:07:42Z",
  "region": "us-east-1",
```

```

"resources": [],
"detail": {
  "jobRunId": "00f1cbn5g4bb0c01",
  "applicationId": "00f1982r1uukb925",
  "arn": "arn:aws:emr-serverless:us-east-1:123456789012:/
applications/00f1982r1uukb925/jobruns/00f1cbn5g4bb0c01",
  "releaseLabel": "emr-6.6.0",
  "createdBy": "arn:aws:sts::123456789012:assumed-role/
TestRole-402dcef3ad14993c15d28263f64381e4cda34775/6622b6233b6d42f59c25dd2637346242",
  "updatedAt": "2022-05-31T21:07:42.299487Z",
  "createdAt": "2022-05-31T21:07:25.325900Z",
  //Attempt Details
  "previousAttempt": 1,
  "previousAttemptState": "FAILED",
  "previousAttemptCreatedAt": "2022-05-31T21:07:25.325900Z",
  "previousAttemptEndedAt": "2022-05-31T21:07:30.325900Z",
  "newAttempt": 2,
  "newAttemptCreatedAt": "2022-05-31T21:07:30.325900Z"
}
}

```

Pembaruan Pemanfaatan Sumber Daya Pekerjaan

Contoh peristiwa berikut menunjukkan pembaruan pemanfaatan sumber daya akhir untuk pekerjaan yang dipindahkan ke status terminal setelah dijalankan.

```

{
  "version": "0",
  "id": "00df3ec6-5da1-36e6-ab71-20f0de68f8a0",
  "detail-type": "EMR Serverless Job Resource Utilization Update",
  "source": "aws.emr-serverless",
  "account": "123456789012",
  "time": "2022-05-31T21:07:42Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:emr-serverless:us-east-1:123456789012:/applications/00f1982r1uukb925/
jobruns/00f1cbn5g4bb0c01"
  ],
  "detail": {
    "applicationId": "00f1982r1uukb925",
    "jobRunId": "00f1cbn5g4bb0c01",
    "attempt": 1,
    "mode": "BATCH",

```

```
"createdAt": "2022-05-31T21:07:25.325900Z",
"startedAt": "2022-05-31T21:07:26.123Z",
"calculatedFrom": "2022-05-31T21:07:42.299487Z",
"calculatedTo": "2022-05-31T21:07:30.325900Z",
"resourceUtilizationFinal": true,
"resourceUtilizationForInterval": {
  "vCPUHour": 0.023,
  "memoryGBHour": 0.114,
  "storageGBHour": 0.228
},
"billedResourceUtilizationForInterval": {
  "vCPUHour": 0.067,
  "memoryGBHour": 0.333,
  "storageGBHour": 0
},
"totalResourceUtilization": {
  "vCPUHour": 0.023,
  "memoryGBHour": 0.114,
  "storageGBHour": 0.228
},
"totalBilledResourceUtilization": {
  "vCPUHour": 0.067,
  "memoryGBHour": 0.333,
  "storageGBHour": 0
}
}
```

Bidang `startedAt` hanya akan hadir dalam acara jika pekerjaan telah pindah ke status berjalan.

Penandaan pada sumber daya

Tetapkan metadata Anda sendiri ke setiap sumber daya menggunakan tag untuk membantu Anda mengelola sumber daya EMR Tanpa Server. Bagian ini memberikan ikhtisar fungsi tag dan menunjukkan cara membuat tag.

Topik

- [Apa itu tag?](#)
- [Menandai sumber daya Anda](#)
- [Batasan penandaan](#)
- [Bekerja dengan tag menggunakan API AWS CLI Tanpa Server EMR Amazon dan Amazon](#)

Apa itu tag?

Tag adalah label yang Anda tetapkan ke AWS sumber daya. Setiap tanda terdiri atas sebuah kunci dan sebuah nilai, yang keduanya Anda tentukan. Tag memungkinkan Anda untuk mengkategorikan AWS sumber daya berdasarkan atribut seperti tujuan, pemilik, dan lingkungan. Bila Anda memiliki banyak sumber daya dari jenis yang sama, cepat mengidentifikasi sumber daya tertentu berdasarkan tag yang ditetapkan untuk itu. Misalnya, tentukan satu set tag untuk aplikasi Amazon EMR Tanpa Server Anda untuk membantu melacak setiap pemilik aplikasi dan tingkat tumpukan. Kami menyarankan Anda merancang satu set kunci tag yang konsisten untuk setiap jenis sumber daya.

Selain itu, tag tidak dapat ditetapkan secara otomatis ke sumber daya Anda. Setelah Anda menambahkan tag ke sumber daya, ubah nilai tag atau hapus tag dari sumber daya kapan saja. Tag tidak memiliki arti semantik untuk Amazon EMR Tanpa Server dan ditafsirkan secara ketat sebagai string karakter. Jika Anda menambahkan tanda yang memiliki kunci yang sama dengan tanda yang ada pada sumber daya tersebut, nilai baru akan menimpa nilai sebelumnya.

Jika Anda menggunakan IAM, Anda dapat mengontrol pengguna mana di AWS akun Anda yang memiliki izin untuk mengelola tag. Untuk contoh kebijakan kontrol akses berbasis tag, lihat. [Kebijakan untuk kendali akses berbasis tanda](#)

Menandai sumber daya Anda

Anda dapat menandai aplikasi baru atau yang sudah ada dan menjalankan pekerjaan. Jika Anda menggunakan Amazon EMR Serverless API, SDK, atau AWS SDK AWS CLI, Anda dapat

menerapkan tag ke sumber daya baru menggunakan `tags` parameter 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 pembuatan, Anda tidak perlu menjalankan skrip penandaan khusus setelah membuat sumber daya.

Tabel berikut menjelaskan sumber daya Amazon EMR Tanpa Server yang dapat ditandai.

Sumber daya yang dapat diberi tag

Sumber daya	Mendukung tag	Mendukung penyebaran tag	Mendukung penandaan pada pembuatan (Amazon EMR Serverless API AWS CLI,, dan SDK) AWS	API untuk pembuatan (tanda dapat ditambahkan selama pembuatan)
Aplikasi	Ya	Tidak. Tag yang terkait dengan aplikasi tidak menyebar ke pekerjaan yang dikirimkan ke aplikasi itu.	Ya	CreateApplication
Tugas berjalan	Ya	Tidak	Ya	StartJobRun

Batasan penandaan

Batasan dasar berikut berlaku untuk tag:

- Setiap sumber daya dapat memiliki maksimal 50 tag yang dibuat pengguna.

- Untuk setiap sumber daya, setiap kunci tanda harus unik, dan setiap kunci tanda hanya dapat memuat satu nilai.
- Panjang kunci maksimum adalah 128 karakter Unicode di UTF-8.
- Panjang nilai maksimum adalah 256 karakter Unicode di UTF-8.
- Karakter yang diizinkan adalah huruf, angka, spasi yang dapat direpresentasikan dalam UTF-8, dan karakter berikut: `._:/= + - @`.
- Kunci tanda tidak bisa berupa string kosong. Nilai tag dapat berupa string kosong, tetapi bukan nol.
- Kunci dan nilai tanda peka huruf besar/kecil.
- Jangan gunakan `AWS :` atau kombinasi huruf besar atau kecil seperti awalan untuk kunci atau nilai. Ini disimpan hanya untuk penggunaan AWS .

Bekerja dengan tag menggunakan API AWS CLI Tanpa Server EMR Amazon dan Amazon

Gunakan AWS CLI perintah berikut atau operasi API Tanpa Server Amazon EMR untuk menambahkan, memperbarui, membuat daftar, dan menghapus tag untuk sumber daya Anda.

Perintah CLI dan operasi API untuk tag

Sumber daya	Mendukung tag	Mendukung penyebaran tag
Menambahkan atau menimpa satu tanda atau lebih	<code>tag-resource</code>	<code>TagResource</code>
Membuat daftar tanda untuk sumber daya	<code>list-tags-for-resource</code>	<code>ListTagsForResource</code>
Menghapus satu tanda atau lebih	<code>untag-resource</code>	<code>UntagResource</code>

Contoh berikut menunjukkan cara menandai atau menghapus tag sumber daya menggunakan AWS CLI

Tandai aplikasi yang ada

Perintah berikut menandai aplikasi yang ada.

```
aws emr-serverless tag-resource --resource-arn resource_ARN --tags team=devs
```

Hapus tag aplikasi yang ada

Perintah berikut menghapus tag dari aplikasi yang ada.

```
aws emr-serverless untag-resource --resource-arn resource_ARN --tag-keys tag_key
```

Daftar tag untuk sumber daya

Perintah berikut membuat daftar tanda yang terkait dengan sumber daya yang ada.

```
aws emr-serverless list-tags-for-resource --resource-arn resource_ARN
```

Tutorial untuk EMR Tanpa Server

Bagian ini menjelaskan kasus penggunaan umum saat Anda bekerja dengan aplikasi EMR Tanpa Server. Ini termasuk berbagai alat termasuk Hudi dan Iceberg untuk mengerjakan kumpulan data besar dan menggunakan pustaka Python dan Python untuk mengirimkan pekerjaan Spark.

Topik

- [Menggunakan Java 17 dengan Amazon EMR Tanpa Server](#)
- [Menggunakan Apache Hudi dengan EMR Serverless](#)
- [Menggunakan Apache Iceberg dengan EMR Serverless](#)
- [Menggunakan pustaka Python dengan EMR Tanpa Server](#)
- [Menggunakan versi Python yang berbeda dengan EMR Serverless](#)
- [Menggunakan Delta Lake OSS dengan EMR Tanpa Server](#)
- [Mengirimkan pekerjaan EMR Tanpa Server dari Airflow](#)
- [Menggunakan fungsi yang ditentukan pengguna Hive dengan EMR Tanpa Server](#)
- [Menggunakan gambar kustom dengan EMR Serverless](#)
- [Menggunakan integrasi Amazon Redshift untuk Apache Spark di Amazon EMR Tanpa Server](#)
- [Menghubungkan ke DynamoDB dengan Amazon EMR Tanpa Server](#)

Menggunakan Java 17 dengan Amazon EMR Tanpa Server

Dengan Amazon EMR merilis 6.11.0 dan yang lebih tinggi, konfigurasi pekerjaan EMR Serverless Spark untuk menggunakan runtime Java 17 untuk Java Virtual Machine (JVM). Gunakan salah satu metode berikut untuk mengkonfigurasi Spark dengan Java 17.

JAVA_HOME

Untuk mengganti pengaturan JVM untuk EMR Serverless 6.11.0 dan yang lebih tinggi, berikan pengaturan ke klasifikasi dan lingkungannya. `JAVA_HOME spark.emr-serverless.driverEnv`
`spark.executorEnv`

x86_64

Tetapkan properti yang diperlukan untuk menentukan Java 17 sebagai `JAVA_HOME` konfigurasi untuk driver dan pelaksana Spark:

```
--conf spark.emr-serverless.driverEnv.JAVA_HOME=/usr/lib/jvm/java-17-amazon-
corretto.x86_64/
--conf spark.executorEnv.JAVA_HOME=/usr/lib/jvm/java-17-amazon-corretto.x86_64/
```

arm_64

Tetapkan properti yang diperlukan untuk menentukan Java 17 sebagai JAVA_HOME konfigurasi untuk driver dan pelaksana Spark:

```
--conf spark.emr-serverless.driverEnv.JAVA_HOME=/usr/lib/jvm/java-17-amazon-
corretto.aarch64/
--conf spark.executorEnv.JAVA_HOME=/usr/lib/jvm/java-17-amazon-corretto.aarch64/
```

spark-defaults

Atau, Anda dapat menentukan Java 17 dalam spark-defaults klasifikasi untuk mengganti pengaturan JVM untuk EMR Tanpa Server 6.11.0 dan yang lebih tinggi.

x86_64

Tentukan Java 17 dalam spark-defaults klasifikasi:

```
{
  "applicationConfiguration": [
    {
      "classification": "spark-defaults",
      "properties": {
        "spark.emr-serverless.driverEnv.JAVA_HOME" : "/usr/lib/jvm/java-17-
amazon-corretto.x86_64/",
        "spark.executorEnv.JAVA_HOME": "/usr/lib/jvm/java-17-amazon-
corretto.x86_64/"
      }
    }
  ]
}
```

arm_64

Tentukan Java 17 dalam spark-defaults klasifikasi:

```
{
```

```
"applicationConfiguration": [  
  {  
    "classification": "spark-defaults",  
    "properties": {  
      "spark.emr-serverless.driverEnv.JAVA_HOME" : "/usr/lib/jvm/java-17-  
amazon-corretto.aarch64/",  
      "spark.executorEnv.JAVA_HOME": "/usr/lib/jvm/java-17-amazon-  
corretto.aarch64/"  
    }  
  }  
]  
}
```

Menggunakan Apache Hudi dengan EMR Serverless

Bagian ini menjelaskan penggunaan Apache Hudi dengan aplikasi EMR Tanpa Server. Hudi adalah kerangka kerja manajemen data yang membuat pemrosesan data lebih sederhana.

Untuk menggunakan Apache Hudi dengan aplikasi EMR Serverless

1. Setel properti Spark yang diperlukan dalam menjalankan pekerjaan Spark yang sesuai.

```
spark.jars=/usr/lib/hudi/hudi-spark-bundle.jar,/usr/lib/hudi/hudi-utilities-  
bundle.jar,/usr/lib/hudi/hudi-aws-bundle.jar  
spark.serializer=org.apache.spark.serializer.KryoSerializer
```

2. Untuk menyinkronkan tabel Hudi ke katalog yang dikonfigurasi, tentukan Katalog Data AWS Glue sebagai metastore Anda, atau konfigurasi metastore eksternal. EMR Serverless mendukung hms sebagai mode sinkronisasi untuk tabel Hive untuk beban kerja Hudi. EMR Tanpa Server mengaktifkan properti ini sebagai default. Untuk mempelajari lebih lanjut tentang cara mengatur metastore Anda, lihat [Konfigurasi metastore untuk EMR Tanpa Server](#)

Important

EMR Tanpa Server tidak mendukung HIVEQL atau JDBC sebagai opsi mode sinkronisasi untuk tabel Hive untuk menangani beban kerja Hudi. Untuk mempelajari lebih lanjut, lihat [Mode sinkronisasi](#).

Saat Anda menggunakan AWS Glue Data Catalog sebagai metastore Anda, tentukan properti konfigurasi berikut untuk pekerjaan Hudi Anda.

```
--conf spark.jars=/usr/lib/hudi/hudi-spark-bundle.jar,  
--conf spark.serializer=org.apache.spark.serializer.KryoSerializer,  
--conf  
spark.hadoop.hive.metastore.client.factory.class=com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveMetastore
```

[Untuk mempelajari lebih lanjut tentang rilis Apache Hudi dari Amazon EMR, lihat riwayat rilis Hudi.](#)

Menggunakan Apache Iceberg dengan EMR Serverless

Bagian ini menjelaskan cara menggunakan Apache Iceberg dengan aplikasi EMR Tanpa Server. Apache Iceberg adalah format tabel yang membantu bekerja dengan kumpulan data besar di danau data.

Untuk menggunakan Apache Iceberg dengan aplikasi EMR Serverless

1. Setel properti Spark yang diperlukan dalam menjalankan pekerjaan Spark yang sesuai.

```
spark.jars=/usr/share/aws/iceberg/lib/iceberg-spark3-runtime.jar
```

2. Tentukan Katalog Data AWS Glue sebagai metastore Anda atau konfigurasi metastore eksternal. Untuk mempelajari lebih lanjut tentang menyiapkan metastore Anda, lihat [Konfigurasi metastore untuk EMR Tanpa Server](#)

Konfigurasi properti metastore yang ingin Anda gunakan untuk Iceberg. Misalnya, jika Anda ingin menggunakan AWS Glue Data Catalog, atur properti berikut dalam konfigurasi aplikasi.

```
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.hadoop.hive.metastore.client.factory.class=com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveMetastore
```

Saat Anda menggunakan AWS Glue Data Catalog sebagai metastore Anda, tentukan properti konfigurasi berikut untuk pekerjaan Iceberg Anda.

```
--conf spark.jars=/usr/share/aws/iceberg/lib/iceberg-spark3-runtime.jar,
--conf
  spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions,
--conf spark.sql.catalog.dev=org.apache.iceberg.spark.SparkCatalog,
--conf spark.sql.catalog.dev.catalog-impl=org.apache.iceberg.aws.glue.GlueCatalog,
--conf spark.sql.catalog.dev.warehouse=s3://amzn-s3-demo-bucket/EXAMPLE-PREFIX/
--conf
  spark.hadoop.hive.metastore.client.factory.class=com.amazonaws.glue.catalog.metastore.AWSG
```

[Untuk mempelajari lebih lanjut tentang rilis Apache Iceberg dari Amazon EMR, lihat riwayat rilis Iceberg.](#)

Menggunakan pustaka Python dengan EMR Tanpa Server

Saat Anda menjalankan PySpark pekerjaan di aplikasi Amazon EMR Tanpa Server, paketkan berbagai pustaka Python sebagai dependensi. Untuk melakukan ini, gunakan fitur Python asli, bangun lingkungan virtual, atau langsung konfigurasi PySpark pekerjaan Anda untuk menggunakan pustaka Python. Halaman ini mencakup setiap pendekatan.

Menggunakan fitur Python asli

Saat Anda mengatur konfigurasi berikut, gunakan PySpark untuk mengunggah file Python (), paket Python zip (.py), dan file .egg Egg .zip () ke pelaksana Spark.

```
--conf spark.submit.pyFiles=s3://amzn-s3-demo-bucket/EXAMPLE-PREFIX/<.py|.egg|.zip
file>
```

Untuk detail selengkapnya tentang cara menggunakan lingkungan virtual Python untuk PySpark pekerjaan, lihat [Menggunakan Fitur PySpark Asli](#).

Saat menggunakan Notebook EMR, Anda dapat membuat dependensi Python tersedia di Notebook Anda dengan menjalankan kode berikut:

```
%%configure -f
{
  "conf": {
    "spark.submit.pyFiles": "s3:///amzn-s3-demo-bucket/EXAMPLE-PREFIX/<.py|.egg|.zip
file>
```

```
}  
}
```

Membangun lingkungan virtual Python

Untuk mengemas beberapa pustaka Python untuk suatu PySpark pekerjaan, buat lingkungan virtual Python yang terisolasi.

1. Untuk membangun lingkungan virtual Python, gunakan perintah berikut. Contoh yang ditampilkan menginstal paket `scipy` dan `matplotlib` ke dalam paket lingkungan virtual dan menyalin arsip ke lokasi Amazon S3.

Important

Anda harus menjalankan perintah berikut di lingkungan Amazon Linux 2 yang serupa dengan versi Python yang sama seperti yang Anda gunakan di EMR Tanpa Server, yaitu, Python 3.7.10 untuk Amazon EMR rilis 6.6.0. Anda dapat menemukan contoh Dockerfile di repositori [EMR Serverless Sampel](#). GitHub

```
# initialize a python virtual environment  
python3 -m venv pyspark_venvsource  
source pyspark_venvsource/bin/activate  
  
# optionally, ensure pip is up-to-date  
pip3 install --upgrade pip  
  
# install the python packages  
pip3 install scipy  
pip3 install matplotlib  
  
# package the virtual environment into an archive  
pip3 install venv-pack  
venv-pack -f -o pyspark_venv.tar.gz  
  
# copy the archive to an S3 location  
aws s3 cp pyspark_venv.tar.gz s3://amzn-s3-demo-bucket/EXAMPLE-PREFIX/  
  
# optionally, remove the virtual environment directory  
rm -fr pyspark_venvsource
```

2. Kirimkan pekerjaan Spark dengan properti Anda yang disetel untuk menggunakan lingkungan virtual Python.

```
--conf spark.archives=s3://amzn-s3-demo-bucket/EXAMPLE-PREFIX/  
pyspark_venv.tar.gz#environment  
--conf spark.emr-serverless.driverEnv.PYSPARK_DRIVER_PYTHON=./environment/bin/  
python  
--conf spark.emr-serverless.driverEnv.PYSPARK_PYTHON=./environment/bin/python  
--conf spark.executorEnv.PYSPARK_PYTHON=./environment/bin/python
```

Perhatikan bahwa jika Anda tidak mengganti biner Python asli, konfigurasi kedua dalam urutan pengaturan sebelumnya adalah. `--conf spark.executorEnv.PYSPARK_PYTHON=python`

Untuk informasi lebih lanjut tentang cara menggunakan lingkungan virtual Python untuk PySpark pekerjaan, lihat [Menggunakan Virtualenv](#). Untuk contoh lebih lanjut tentang cara mengirimkan pekerjaan Spark, lihat. [Menggunakan konfigurasi Spark saat Anda menjalankan pekerjaan EMR Tanpa Server](#)

Mengkonfigurasi PySpark pekerjaan untuk menggunakan pustaka Python

[Dengan Amazon EMR rilis 6.12.0 dan yang lebih tinggi, Anda dapat langsung mengonfigurasi pekerjaan EMR Tanpa Server PySpark untuk menggunakan pustaka Python ilmu data populer seperti panda,, dan tanpa pengaturan tambahan apa pun. NumPyPyArrow](#)

Contoh berikut menunjukkan cara mengemas setiap pustaka Python untuk suatu PySpark pekerjaan.

NumPy

NumPy adalah perpustakaan Python untuk komputasi ilmiah yang menawarkan array multidimensi dan operasi untuk matematika, penyortiran, simulasi acak, dan statistik dasar. Untuk menggunakan NumPy, jalankan perintah berikut:

```
import numpy
```

pandas

panda adalah pustaka Python yang dibangun di atasnya. NumPy Perpustakaan panda menyediakan para ilmuwan data dengan struktur [DataFrame](#) data dan alat analisis data. Untuk menggunakan panda, jalankan perintah berikut:

```
import pandas
```

PyArrow

PyArrow adalah pustaka Python yang mengelola data kolumnar dalam memori untuk meningkatkan kinerja pekerjaan. PyArrow didasarkan pada spesifikasi pengembangan lintas bahasa Apache Arrow, yang merupakan cara standar untuk mewakili dan bertukar data dalam format kolumnar. Untuk menggunakan PyArrow, jalankan perintah berikut:

```
import pyarrow
```

Menggunakan versi Python yang berbeda dengan EMR Serverless

Selain kasus penggunaan di [Menggunakan pustaka Python dengan EMR Tanpa Server](#), Anda juga dapat menggunakan lingkungan virtual Python untuk bekerja dengan versi Python yang berbeda dari versi yang dikemas dalam rilis EMR Amazon untuk aplikasi Amazon EMR Tanpa Server Anda. Untuk melakukan ini, bangun lingkungan virtual Python dengan versi Python yang ingin Anda gunakan.

Untuk mengirimkan pekerjaan dari lingkungan virtual Python

1. Bangun lingkungan virtual Anda dengan perintah dalam contoh berikut. Contoh ini menginstal Python 3.9.9 ke dalam paket lingkungan virtual dan menyalin arsip ke lokasi Amazon S3.

Important

Jika Anda menggunakan Amazon EMR rilis 7.0.0 dan yang lebih tinggi, jalankan perintah Anda di lingkungan Amazon Linux 2023 yang mirip dengan yang Anda gunakan untuk aplikasi EMR Tanpa Server.

Jika Anda menggunakan rilis 6.15.0 atau yang lebih rendah, jalankan perintah berikut di lingkungan Amazon Linux 2 yang serupa.

```
# install Python 3.9.9 and activate the venv
yum install -y gcc openssl-devel bzip2-devel libffi-devel tar gzip wget make
wget https://www.python.org/ftp/python/3.9.9/Python-3.9.9.tgz && \
tar xzf Python-3.9.9.tgz && cd Python-3.9.9 && \
./configure --enable-optimizations --enable-shared && \
make altinstall
```

```
# create python venv with Python 3.9.9
python3.9 -m venv pyspark_venv_python_3.9.9 --copies
source pyspark_venv_python_3.9.9/bin/activate

# copy system python3 libraries and shared libraries to venv
cp -r /usr/local/lib/python3.9/* ./pyspark_venv_python_3.9.9/lib/python3.9/
cp /usr/local/lib/libpython3.9* ./pyspark_venv_python_3.9.9/lib/

# package venv to archive.
# Note that you have to supply --python-prefix option
# to make sure python starts with the path where your
# copied libraries are present.
# Copying the python binary to the "environment" directory.
pip3 install venv-pack
venv-pack -f -o pyspark_venv_python_3.9.9.tar.gz --python-prefix /home/hadoop/
environment

# stage the archive in S3
aws s3 cp pyspark_venv_python_3.9.9.tar.gz s3://<path>

# optionally, remove the virtual environment directory
rm -fr pyspark_venv_python_3.9.9
```

2. Atur properti Anda untuk menggunakan lingkungan virtual Python dan kirimkan pekerjaan Spark.

```
# note that the archive suffix "environment" is the same as the directory where you
# copied the Python binary.
--conf spark.archives=s3://amzn-s3-demo-bucket/EXAMPLE-PREFIX/
pyspark_venv_python_3.9.9.tar.gz#environment
--conf spark.emr-serverless.driverEnv.PYSPARK_DRIVER_PYTHON=./environment/bin/
python
--conf spark.emr-serverless.driverEnv.PYSPARK_PYTHON=./environment/bin/python
--conf spark.executorEnv.PYSPARK_PYTHON=./environment/bin/python
--conf spark.emr-serverless.driverEnv.LD_LIBRARY_PATH=./environment/lib
--conf spark.executorEnv.LD_LIBRARY_PATH=./environment/lib
```

Untuk informasi lebih lanjut tentang cara menggunakan lingkungan virtual Python untuk PySpark pekerjaan, lihat [Menggunakan Virtualenv](#). Untuk contoh lebih lanjut tentang cara mengirimkan pekerjaan Spark, lihat [Menggunakan konfigurasi Spark saat Anda menjalankan pekerjaan EMR Tanpa Server](#)

Menggunakan Delta Lake OSS dengan EMR Tanpa Server

Amazon EMR versi 6.9.0 dan lebih tinggi

Note

Amazon EMR 7.0.0 dan yang lebih tinggi menggunakan Delta Lake 3.0.0, yang mengganti nama file menjadi `delta-core.jar` dan `delta-spark.jar`. Jika Anda menggunakan Amazon EMR 7.0.0 atau yang lebih tinggi, pastikan untuk menentukan `delta-spark.jar` dalam konfigurasi Anda.

Amazon EMR 6.9.0 dan yang lebih tinggi termasuk Delta Lake, jadi Anda tidak perlu lagi mengemas Delta Lake sendiri atau memberikan bendera `--packages` dengan pekerjaan EMR Tanpa Server Anda.

1. Saat Anda mengirimkan pekerjaan EMR Tanpa Server, pastikan Anda memiliki properti konfigurasi berikut dan sertakan parameter berikut di bidang `sparkSubmitParameters`

```
--conf spark.jars=/usr/share/aws/delta/lib/delta-core.jar,/usr/share/aws/delta/lib/delta-storage.jar
--conf spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension
--conf
spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog
```

2. Buat lokal `delta_sample.py` untuk menguji pembuatan dan membaca tabel Delta.

```
# delta_sample.py
from pyspark.sql import SparkSession

import uuid

url = "s3://amzn-s3-demo-bucket/delta-lake/output/%s/" % str(uuid.uuid4())
spark = SparkSession.builder.appName("DeltaSample").getOrCreate()

## creates a Delta table and outputs to target S3 bucket
spark.range(5).write.format("delta").save(url)

## reads a Delta table and outputs to target S3 bucket
spark.read.format("delta").load(url).show
```

3. Dengan menggunakan AWS CLI, unggah `delta_sample.py` file ke bucket Amazon S3 Anda. Kemudian gunakan `start-job-run` perintah untuk mengirimkan pekerjaan ke aplikasi EMR Serverless yang ada.

```
aws s3 cp delta_sample.py s3://amzn-s3-demo-bucket/code/

aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --name emr-delta \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "s3://amzn-s3-demo-bucket/code/delta_sample.py",
      "sparkSubmitParameters": "--conf spark.jars=/usr/share/
aws/delta/lib/delta-core.jar,/usr/share/aws/delta/lib/delta-storage.jar --
conf spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension --conf
spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog"
    }
  }'
```

Untuk menggunakan pustaka Python dengan Delta Lake, tambahkan `delta-core` pustaka dengan [mengemasnya sebagai dependensi atau dengan menggunakannya sebagai](#) gambar kustom.

Atau, Anda dapat menggunakan `SparkContext.addPyFile` untuk menambahkan pustaka Python dari file JAR: `delta-core`

```
import glob
from pyspark.sql import SparkSession

spark = SparkSession.builder.getOrCreate()
spark.sparkContext.addPyFile(glob.glob("/usr/share/aws/delta/lib/delta-core_*.jar")[0])
```

Amazon EMR versi 6.8.0 dan lebih rendah

Jika Anda menggunakan Amazon EMR 6.8.0 atau lebih rendah, ikuti langkah-langkah berikut untuk menggunakan Delta Lake OSS dengan aplikasi EMR Tanpa Server Anda.

1. [Untuk membangun Delta Lake versi open source yang kompatibel dengan versi Spark di aplikasi Amazon EMR Serverless Anda, navigasikan ke Delta dan ikuti petunjuknya. GitHub](#)
2. Unggah perpustakaan Delta Lake ke ember Amazon S3 di ember Anda. Akun AWS

3. Saat Anda mengirimkan pekerjaan EMR Tanpa Server dalam konfigurasi aplikasi, sertakan file JAR Delta Lake yang sekarang ada di ember Anda.

```
--conf spark.jars=s3://amzn-s3-demo-bucket/jars/delta-core_2.12-1.1.0.jar
```

4. Untuk memastikan bahwa Anda dapat membaca dan menulis dari tabel Delta, jalankan PySpark tes sampel.

```
from pyspark import SparkConf, SparkContext
    from pyspark.sql import HiveContext, SparkSession

import uuid

conf = SparkConf()
sc = SparkContext(conf=conf)
sqlContext = HiveContext(sc)

url = "s3://amzn-s3-demo-bucket/delta-lake/output/1.0.1/%s/" %
str(uuid.uuid4())

## creates a Delta table and outputs to target S3 bucket
session.range(5).write.format("delta").save(url)

## reads a Delta table and outputs to target S3 bucket
session.read.format("delta").load(url).show
```

Mengirimkan pekerjaan EMR Tanpa Server dari Airflow

Penyedia Amazon di Apache Airflow menyediakan operator EMR Tanpa Server. Untuk informasi selengkapnya tentang operator, lihat Operator [Tanpa Server Amazon EMR di dokumentasi Apache Airflow](#).

Anda dapat menggunakan `EmrServerlessCreateApplicationOperator` untuk membuat aplikasi Spark atau Hive. Anda juga dapat menggunakan `EmrServerlessStartJobOperator` untuk memulai satu atau lebih pekerjaan dengan aplikasi baru Anda.

Untuk menggunakan operator dengan Amazon Managed Workflows for Apache Airflow (MWAA) dengan Airflow 2.2.2, tambahkan baris berikut ke file Anda dan perbarui lingkungan MWAA `requirements.txt` Anda untuk menggunakan file baru.

```
apache-airflow-providers-amazon==6.0.0
boto3>=1.23.9
```

Perhatikan bahwa dukungan EMR Tanpa Server ditambahkan untuk merilis 5.0.0 dari penyedia Amazon. Rilis 6.0.0 adalah versi terakhir yang kompatibel dengan Airflow 2.2.2. Anda dapat menggunakan versi yang lebih baru dengan Airflow 2.4.3 di MWAA.

Contoh singkat berikut menunjukkan cara membuat aplikasi, menjalankan beberapa pekerjaan Spark, dan kemudian menghentikan aplikasi. Contoh lengkap tersedia di repositori [EMR Serverless Sampel](#). GitHub Untuk detail sparkSubmit konfigurasi tambahan, lihat [Menggunakan konfigurasi Spark saat Anda menjalankan pekerjaan EMR Tanpa Server](#).

```
from datetime import datetime

from airflow import DAG
from airflow.providers.amazon.aws.operators.emr import (
    EmrServerlessCreateApplicationOperator,
    EmrServerlessStartJobOperator,
    EmrServerlessDeleteApplicationOperator,
)

# Replace these with your correct values
JOB_ROLE_ARN = "arn:aws:iam::account-id:role/emr_serverless_default_role"
S3_LOGS_BUCKET = "amzn-s3-demo-bucket"

DEFAULT_MONITORING_CONFIG = {
    "monitoringConfiguration": {
        "s3MonitoringConfiguration": {"logUri": f"s3://amzn-s3-demo-bucket/logs/"}
    },
}

with DAG(
    dag_id="example_endtoend_emr_serverless_job",
    schedule_interval=None,
    start_date=datetime(2021, 1, 1),
    tags=["example"],
    catchup=False,
) as dag:
    create_app = EmrServerlessCreateApplicationOperator(
        task_id="create_spark_app",
        job_type="SPARK",
        release_label="emr-6.7.0",
```

```
        config={"name": "airflow-test"},
    )

    application_id = create_app.output

    job1 = EmrServerlessStartJobOperator(
        task_id="start_job_1",
        application_id=application_id,
        execution_role_arn=JOB_ROLE_ARN,
        job_driver={
            "sparkSubmit": {
                "entryPoint": "local:///usr/lib/spark/examples/src/main/python/
pi_fail.py",
            }
        },
        configuration_overrides=DEFAULT_MONITORING_CONFIG,
    )

    job2 = EmrServerlessStartJobOperator(
        task_id="start_job_2",
        application_id=application_id,
        execution_role_arn=JOB_ROLE_ARN,
        job_driver={
            "sparkSubmit": {
                "entryPoint": "local:///usr/lib/spark/examples/src/main/python/pi.py",
                "entryPointArguments": ["1000"]
            }
        },
        configuration_overrides=DEFAULT_MONITORING_CONFIG,
    )

    delete_app = EmrServerlessDeleteApplicationOperator(
        task_id="delete_app",
        application_id=application_id,
        trigger_rule="all_done",
    )

    (create_app >> [job1, job2] >> delete_app)
```

Menggunakan fungsi yang ditentukan pengguna Hive dengan EMR Tanpa Server

Hive user-defined functions (UDFs) memungkinkan Anda membuat fungsi kustom untuk memproses catatan atau grup rekaman. Dalam tutorial ini, Anda akan menggunakan contoh UDF dengan aplikasi Amazon EMR Tanpa Server yang sudah ada sebelumnya untuk menjalankan pekerjaan yang menghasilkan hasil kueri. Untuk mempelajari cara mengatur aplikasi, lihat [Memulai dengan Amazon EMR Tanpa Server](#).

Untuk menggunakan UDF dengan EMR Serverless

1. Arahkan ke [GitHub](#) untuk sampel UDF. Kloning repo dan beralih ke cabang git yang ingin Anda gunakan. Perbarui pom.xml file maven-compiler-plugin dalam repositori untuk memiliki sumber. Juga perbarui konfigurasi versi java target ke 1.8. Jalankan `mvn package -DskipTests` untuk membuat file JAR yang berisi sampel Anda UDFs.
2. Setelah Anda membuat file JAR, unggah ke bucket S3 Anda dengan perintah berikut.

```
aws s3 cp brickhouse-0.8.2-JS.jar s3://amzn-s3-demo-bucket/jars/
```

3. Buat file contoh untuk menggunakan salah satu contoh fungsi UDF. Simpan kueri ini sebagai `udf_example.q` dan unggah ke bucket S3 Anda.

```
add jar s3://amzn-s3-demo-bucket/jars/brickhouse-0.8.2-JS.jar;
CREATE TEMPORARY FUNCTION from_json AS 'brickhouse.udf.json.FromJsonUDF';
select from_json('{"key1":[0,1,2], "key2":[3,4,5,6], "key3":[7,8,9]}', map("",
array(cast(0 as int))));
select from_json('{"key1":[0,1,2], "key2":[3,4,5,6], "key3":[7,8,9]}', map("",
array(cast(0 as int))))["key1"][2];
```

4. Kirimkan pekerjaan Hive berikut.

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "hive": {
      "query": "s3://amzn-s3-demo-bucket/queries/udf_example.q",
      "parameters": "--hiveconf hive.exec.scratchdir=s3://amzn-s3-demo-bucket/
emr-serverless-hive/scratch --hiveconf hive.metastore.warehouse.dir=s3://'$BUCKET'/
emr-serverless-hive/warehouse"
```

```

    }
  }' --configuration-overrides '{
    "applicationConfiguration": [{
      "classification": "hive-site",
      "properties": {
        "hive.driver.cores": "2",
        "hive.driver.memory": "6G"
      }
    }],
    "monitoringConfiguration": {
      "s3MonitoringConfiguration": {
        "logUri": "s3://amzn-s3-demo-bucket/logs/"
      }
    }
  }'

```

- Gunakan `get-job-run` perintah untuk memeriksa status pekerjaan Anda. Tunggu sampai negara berubah `SUCCESS`.

```
aws emr-serverless get-job-run --application-id application-id --job-run-id job-id
```

- Unduh file output dengan perintah berikut.

```
aws s3 cp --recursive s3://amzn-s3-demo-bucket/logs/applications/application-id/
jobs/job-id/HIVE_DRIVER/ .
```

`stdout.gzFile` tersebut menyerupai yang berikut ini.

```
{"key1": [0, 1, 2], "key2": [3, 4, 5, 6], "key3": [7, 8, 9]}
2
```

Menggunakan gambar kustom dengan EMR Serverless

Topik

- [Gunakan versi Python khusus](#)
- [Gunakan versi Java kustom](#)
- [Membangun citra ilmu data](#)
- [Memproses data geospasial dengan Apache Sedona](#)

- [Informasi lisensi untuk menggunakan gambar kustom](#)

Gunakan versi Python khusus

Anda dapat membuat gambar khusus untuk menggunakan versi Python yang berbeda. Untuk menggunakan Python versi 3.10 untuk pekerjaan Spark, misalnya, jalankan perintah berikut:

```
FROM public.ecr.aws/emr-serverless/spark/emr-6.9.0:latest

USER root

# install python 3
RUN yum install -y gcc openssl-devel bzip2-devel libffi-devel tar gzip wget make
RUN wget https://www.python.org/ftp/python/3.10.0/Python-3.10.0.tgz && \
tar xzf Python-3.10.0.tgz && cd Python-3.10.0 && \
./configure --enable-optimizations && \
make altinstall

# EMRS runs the image as hadoop
USER hadoop:hadoop
```

Sebelum Anda mengirimkan pekerjaan Spark, atur properti Anda untuk menggunakan lingkungan virtual Python, sebagai berikut.

```
--conf spark.emr-serverless.driverEnv.PYSPARK_DRIVER_PYTHON=/usr/local/bin/python3.10
--conf spark.emr-serverless.driverEnv.PYSPARK_PYTHON=/usr/local/bin/python3.10
--conf spark.executorEnv.PYSPARK_PYTHON=/usr/local/bin/python3.10
```

Gunakan versi Java kustom

Contoh berikut menunjukkan cara membuat gambar kustom untuk menggunakan Java 11 untuk pekerjaan Spark Anda.

```
FROM public.ecr.aws/emr-serverless/spark/emr-6.9.0:latest

USER root

# install JDK 11
RUN amazon-linux-extras install java-openjdk11
```

```
# EMRS runs the image as hadoop
USER hadoop:hadoop
```

Sebelum Anda mengirimkan pekerjaan Spark, atur properti Spark untuk menggunakan Java 11, sebagai berikut.

```
--conf spark.executorEnv.JAVA_HOME=/usr/lib/jvm/java-11-
openjdk-11.0.16.0.8-1.amzn2.0.1.x86_64
--conf spark.emr-serverless.driverEnv.JAVA_HOME=/usr/lib/jvm/java-11-
openjdk-11.0.16.0.8-
```

Membangun citra ilmu data

Contoh berikut menunjukkan cara memasukkan paket Python ilmu data umum, seperti Pandas dan NumPy

```
FROM public.ecr.aws/emr-serverless/spark/emr-6.9.0:latest

USER root

# python packages
RUN pip3 install boto3 pandas numpy
RUN pip3 install -U scikit-learn==0.23.2 scipy
RUN pip3 install sk-dist
RUN pip3 install xgboost

# EMR Serverless runs the image as hadoop
USER hadoop:hadoop
```

Memproses data geospasial dengan Apache Sedona

Contoh berikut menunjukkan bagaimana membangun gambar untuk menyertakan Apache Sedona untuk pemrosesan geospasial.

```
FROM public.ecr.aws/emr-serverless/spark/emr-6.9.0:latest

USER root

RUN yum install -y wget
RUN wget https://repo1.maven.org/maven2/org/apache/sedona/sedona-core-3.0_2.12/1.3.0-
incubating/sedona-core-3.0_2.12-1.3.0-incubating.jar -P /usr/lib/spark/jars/
```

```
RUN pip3 install apache-sedona

# EMRS runs the image as hadoop
USER hadoop:hadoop
```

Informasi lisensi untuk menggunakan gambar kustom

Anda dapat membangun gambar kustom dengan EMR Serverless untuk melakukan tugas-tugas tertentu atau menggunakan versi tertentu dari paket perangkat lunak. Modifikasi dan distribusi gambar kustom dapat tunduk pada aturan dan ketentuan lisensi. Teks lisensi muncul di ayat berikut.

Lisensi yang berlaku untuk gambar kustom

Hak Cipta Amazon.com dan afiliasinya; semua hak dilindungi undang-undang. Perangkat lunak ini adalah AWS Konten berdasarkan [Perjanjian AWS Pelanggan](#) dan tidak boleh didistribusikan tanpa izin. Selain izin dalam Lisensi [AWS Kekayaan Intelektual](#), [AWS Pemberi Lisensi](#) memberi Anda izin tambahan ini:

Membuat, Menyalin, dan Menggunakan Derivatif AWS Konten diizinkan asalkan kondisi berikut terpenuhi:

- Anda tidak memodifikasi AWS Konten itu sendiri, dan Derivatif apa pun secara ketat merupakan hasil dari penambahan konten baru Anda.
- Reproduksi internal harus mempertahankan pemberitahuan hak cipta di atas.
- Distribusi eksternal, dalam bentuk sumber atau biner, dengan atau tanpa modifikasi, tidak diizinkan berdasarkan ketentuan lisensi ini.

Untuk informasi lebih lanjut tentang menggunakan gambar kustom, lihat [Menggunakan gambar kustom dengan EMR](#) Tanpa Server.

Menggunakan integrasi Amazon Redshift untuk Apache Spark di Amazon EMR Tanpa Server

Dengan Amazon EMR rilis 6.9.0 dan yang lebih baru, setiap gambar rilis menyertakan konektor antara [Apache](#) Spark dan Amazon Redshift. Dengan konektor ini, gunakan Spark di Amazon EMR Serverless untuk memproses data yang disimpan di Amazon Redshift. Integrasi ini didasarkan pada [konektor spark-redshift open-source](#). Untuk Amazon EMR Tanpa Server, integrasi Amazon Redshift [untuk Apache Spark disertakan sebagai integrasi](#) asli.

Topik

- [Meluncurkan aplikasi Spark dengan 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 dengan integrasi Amazon Redshift untuk Apache Spark

Untuk menggunakan integrasi dengan EMR Serverless 6.9.0, teruskan dependensi Spark-Redshift yang diperlukan dengan pekerjaan Spark Anda. Gunakan `--jars` untuk menyertakan pustaka terkait konektor Redshift. Untuk mengakses 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`

Amazon EMR merilis 6.10.0 dan yang lebih tinggi tidak memerlukan ketergantungan, dan secara otomatis menginstal `minimal-json.jar` dependensi lain ke setiap cluster secara default. Contoh berikut menunjukkan cara meluncurkan aplikasi Spark dengan integrasi Amazon Redshift untuk Apache Spark.

Amazon EMR 6.10.0 +

Luncurkan pekerjaan Spark di Amazon EMR Tanpa Server dengan integrasi Amazon Redshift untuk Apache Spark pada rilis EMR Tanpa Server 6.10.0 dan yang lebih tinggi.

```
spark-submit my_script.py
```

Amazon EMR 6.9.0

Untuk meluncurkan pekerjaan Spark di Amazon EMR Tanpa Server dengan integrasi Amazon Redshift untuk Apache Spark pada rilis EMR Tanpa Server 6.9.0, gunakan opsi seperti yang

ditunjukkan pada contoh berikut. `--jars` Perhatikan bahwa jalur yang tercantum dengan `--jars` opsi adalah jalur default untuk file JAR.

```
--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
```

```
spark-submit \
  --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 \
  my_script.py
```

Mengautentikasi dengan integrasi Amazon Redshift untuk Apache Spark

Gunakan AWS Secrets Manager untuk mengambil kredensial dan terhubung ke Amazon Redshift

Anda dapat melakukan autentikasi dengan aman ke Amazon Redshift dengan menyimpan kredensialnya di Secrets Manager dan meminta `GetSecretValue` pekerjaan Spark memanggil API untuk mengambilnya:

```
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
```

Otentikasi ke Amazon Redshift dengan driver JDBC

Tetapkan nama pengguna dan kata sandi di dalam URL JDBC

Anda dapat mengautentikasi pekerjaan Spark ke cluster Amazon Redshift dengan menentukan nama database Amazon Redshift dan kata sandi 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 aplikasi Anda, gunakan format berikut untuk mengatur nama pengguna dan kata sandi di URL JDBC:

```
jdbc:redshift://redshifthost:5439/database?user=username&password=password
```

Gunakan otentikasi berbasis IAM dengan peran eksekusi pekerjaan Amazon EMR Tanpa Server

Dimulai dengan rilis Amazon EMR Tanpa Server 6.9.0, driver Amazon Redshift JDBC 2.1 atau yang 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.

Sebaliknya, tentukan `jdbc:redshift:iam://` skema. Ini memerintahkan driver JDBC untuk menggunakan peran eksekusi pekerjaan EMR Tanpa Server Anda 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 ini adalah:

```
jdbc:redshift:iam://examplecluster.abc123xyz789.us-west-2.redshift.amazonaws.com:5439/
dev
```

Izin berikut diperlukan untuk peran pelaksanaan pekerjaan Anda ketika kondisi yang disediakan terpenuhi:

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 cluster 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

Menghubungkan ke Amazon Redshift dalam VPC yang berbeda

Saat Anda menyiapkan kluster Amazon Redshift yang disediakan atau workgroup Amazon Redshift Tanpa Server di bawah VPC, konfigurasi konektivitas VPC untuk aplikasi Amazon EMR Tanpa Server untuk mengakses sumber daya. Untuk informasi lebih lanjut tentang cara mengkonfigurasi konektivitas VPC pada aplikasi EMR Tanpa Server, lihat. [Mengkonfigurasi akses VPC untuk aplikasi EMR Tanpa Server untuk terhubung ke data](#)

- Jika kluster Amazon Redshift atau grup kerja Amazon Redshift Serverless yang disediakan dapat diakses publik, tentukan satu atau beberapa subnet pribadi yang memiliki gateway NAT terpasang saat Anda membuat aplikasi EMR Tanpa Server.
- Jika kluster Amazon Redshift atau grup kerja Tanpa Server Amazon Redshift yang disediakan tidak dapat diakses publik, Anda harus membuat titik akhir VPC dikelola Amazon Redshift untuk kluster Amazon Redshift seperti yang dijelaskan dalam. [Mengkonfigurasi akses VPC untuk aplikasi EMR Tanpa Server untuk terhubung ke data](#) Atau, Anda dapat membuat grup kerja Amazon Redshift Tanpa Server seperti yang dijelaskan dalam [Menghubungkan ke Amazon Redshift Tanpa Server di Panduan Manajemen Pergeseran Merah Amazon](#). Anda harus mengaitkan kluster atau subgrup Anda ke subnet pribadi yang Anda tentukan saat membuat aplikasi EMR Tanpa Server.

Note

Jika Anda menggunakan otentikasi berbasis IAM, dan subnet pribadi Anda untuk aplikasi EMR Tanpa Server tidak memiliki gateway NAT yang terpasang, maka Anda juga harus membuat titik akhir VPC pada subnet tersebut untuk Amazon Redshift atau Amazon Redshift Tanpa Server. Dengan cara ini, driver JDBC dapat mengambil kredensialnya.

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::account-id:role/role-name"

df = sql_context.read \
    .format("io.github.spark_redshift_community.spark.redshift") \
    .option("url", url) \
    .option("dbtable", "table-name") \
    .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", "table-name-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 dengan 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::account-id:role/role-name"  
  
bucket = "s3://path/for/temp/data"  
tableName = "table-name" # Redshift table name  
  
s = f"""CREATE TABLE IF NOT EXISTS {table-name} (country string, data string)  
    USING io.github.spark_redshift_community.spark.redshift  
    OPTIONS (dbtable '{table-name}', 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(table-name, overwrite=False)  
df = spark.sql(f"SELECT * FROM {table-name}")  
df.show()
```

Pertimbangan dan batasan saat menggunakan konektor Spark

- 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 agar 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](#).
 - Kami menyarankan Anda mengaktifkan enkripsi saat [istirahat Amazon Redshift](#).
- Pertimbangkan rekomendasi berikut untuk Amazon S3:
 - Kami menyarankan Anda [memblokir akses publik ke bucket Amazon S3](#).
 - Kami menyarankan Anda menggunakan [enkripsi sisi server Amazon S3 untuk mengenkripsi bucket](#) Amazon S3 yang digunakan.
 - Sebaiknya gunakan [kebijakan siklus hidup Amazon S3](#) untuk menentukan aturan retensi bucket Amazon S3.
 - Amazon EMR selalu memverifikasi kode yang diimpor dari sumber terbuka ke dalam gambar. Demi keamanan, kami tidak mendukung metode otentikasi berikut dari Spark ke Amazon S3:
 - Mengatur kunci AWS akses dalam klasifikasi `hadoop-env` konfigurasi
 - Pengkodean kunci AWS akses di URI `tempdir`

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

Menghubungkan ke DynamoDB dengan Amazon EMR Tanpa Server

Dalam tutorial ini, Anda mengunggah subset data dari [Dewan Amerika Serikat tentang Nama Geografis](#) ke bucket Amazon S3 dan kemudian menggunakan Hive atau Spark di Amazon EMR Tanpa Server untuk menyalin data ke tabel Amazon DynamoDB untuk kueri.

Langkah 1: Unggah data ke bucket Amazon S3

Untuk membuat bucket Amazon S3, ikuti petunjuk dalam [Membuat](#) bucket di Panduan Pengguna Amazon Simple Storage Service Console. Ganti referensi *amzn-s3-demo-bucket* dengan nama bucket yang baru Anda buat. Sekarang aplikasi EMR Serverless Anda siap menjalankan pekerjaan.

1. Unduh arsip data sampel `features.zip` dengan perintah berikut.

```
wget https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/features.zip
```

2. Ekstrak `features.txt` file dari arsip dan akses yang pertama beberapa baris dalam file:

```
unzip features.zip  
head features.txt
```

Hasilnya akan tampak mirip dengan yang berikut ini.

```
1535908|Big Run|Stream|WV|38.6370428|-80.8595469|794  
875609|Constable Hook|Cape|NJ|40.657881|-74.0990309|7  
1217998|Gooseberry Island|Island|RI|41.4534361|-71.3253284|10  
26603|Boone Moore Spring|Spring|AZ|34.0895692|-111.410065|3681  
1506738|Missouri Flat|Flat|WA|46.7634987|-117.0346113|2605  
1181348|Minnow Run|Stream|PA|40.0820178|-79.3800349|1558  
1288759|Hunting Creek|Stream|TN|36.343969|-83.8029682|1024  
533060|Big Charles Bayou|Bay|LA|29.6046517|-91.9828654|0  
829689|Greenwood Creek|Stream|NE|41.596086|-103.0499296|3671  
541692|Button Willow Island|Island|LA|31.9579389|-93.0648847|98
```

Bidang di setiap baris di sini menunjukkan pengidentifikasi unik, nama, jenis fitur alami, keadaan, garis lintang dalam derajat, bujur dalam derajat, dan tinggi dalam kaki.

3. Unggah data Anda ke Amazon S3

```
aws s3 cp features.txt s3://amzn-s3-demo-bucket/features/
```

Langkah 2: Buat tabel Hive

Gunakan Apache Spark atau Hive untuk membuat tabel Hive baru yang berisi data yang diunggah di Amazon S3.

Spark

Untuk membuat tabel Hive dengan Spark, jalankan perintah berikut.

```
import org.apache.spark.sql.SparkSession

val sparkSession = SparkSession.builder().enableHiveSupport().getOrCreate()

sparkSession.sql("CREATE TABLE hive_features \
  (feature_id BIGINT, \
  feature_name STRING, \
  feature_class STRING, \
  state_alpha STRING, \
  prim_lat_dec DOUBLE, \
  prim_long_dec DOUBLE, \
  elev_in_ft BIGINT) \
  ROW FORMAT DELIMITED \
  FIELDS TERMINATED BY '|' \
  LINES TERMINATED BY '\n' \
  LOCATION 's3://amzn-s3-demo-bucket/features';")
```

Anda sekarang memiliki tabel Hive terisi dengan data dari file. `features.txt` Untuk memverifikasi bahwa data Anda ada dalam tabel, jalankan query Spark SQL seperti yang ditunjukkan pada contoh berikut.

```
sparkSession.sql(
  "SELECT state_alpha, COUNT(*) FROM hive_features GROUP BY state_alpha;")
```

Hive

Untuk membuat tabel Hive dengan Hive, jalankan perintah berikut.

```
CREATE TABLE hive_features
```

```
(feature_id          BIGINT,
feature_name        STRING ,
feature_class       STRING ,
state_alpha         STRING,
prim_lat_dec        DOUBLE ,
prim_long_dec       DOUBLE ,
elev_in_ft          BIGINT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
LINES TERMINATED BY '\n'
LOCATION 's3://amzn-s3-demo-bucket/features';
```

Anda sekarang memiliki tabel Hive yang berisi data dari `features.txt` file. Untuk memverifikasi bahwa data Anda ada dalam tabel, jalankan kueri HiveQL, seperti yang ditunjukkan pada contoh berikut.

```
SELECT state_alpha, COUNT(*) FROM hive_features GROUP BY state_alpha;
```

Langkah 3: Salin data ke DynamoDB

Gunakan Spark atau Hive untuk menyalin data ke tabel DynamoDB baru.

Spark

Untuk menyalin data dari tabel Hive yang Anda buat pada langkah sebelumnya ke DynamoDB, ikuti Langkah 1-3 di [Salin data ke DynamoDB](#). Ini menciptakan tabel DynamoDB baru yang disebut. Features Anda kemudian dapat membaca data langsung dari file teks dan menyalinnya ke tabel DynamoDB Anda, seperti contoh berikut menunjukkan.

```
import com.amazonaws.services.dynamodbv2.model.AttributeValue
import org.apache.hadoop.dynamodb.DynamoDBItemWritable
import org.apache.hadoop.dynamodb.read.DynamoDBInputFormat
import org.apache.hadoop.io.Text
import org.apache.hadoop.mapred.JobConf
import org.apache.spark.SparkContext

import scala.collection.JavaConverters._

object EmrServerlessDynamoDbTest {
```

```
def main(args: Array[String]): Unit = {

    jobConf.set("dynamodb.input.tableName", "Features")
    jobConf.set("dynamodb.output.tableName", "Features")
    jobConf.set("dynamodb.region", "region")

    jobConf.set("mapred.output.format.class",
"org.apache.hadoop.dynamodb.write.DynamoDBOutputFormat")
    jobConf.set("mapred.input.format.class",
"org.apache.hadoop.dynamodb.read.DynamoDBInputFormat")

    val rdd = sc.textFile("s3://amzn-s3-demo-bucket/ddb-connector/")
        .map(row => {
            val line = row.split("\\|")
            val item = new DynamoDBItemWritable()

            val elevInFt = if (line.length > 6) {
                new AttributeValue().withN(line(6))
            } else {
                new AttributeValue().withNULL(true)
            }

            item.setItem(Map(
                "feature_id" -> new AttributeValue().withN(line(0)),
                "feature_name" -> new AttributeValue(line(1)),
                "feature_class" -> new AttributeValue(line(2)),
                "state_alpha" -> new AttributeValue(line(3)),
                "prim_lat_dec" -> new AttributeValue().withN(line(4)),
                "prim_long_dec" -> new AttributeValue().withN(line(5)),
                "elev_in_ft" -> elevInFt)
                .asJava)
                (new Text(""), item)
            ))
        rdd.saveAsHadoopDataset(jobConf)
    }
```

Hive

Untuk menyalin data dari tabel Hive yang Anda buat pada langkah sebelumnya ke DynamoDB, ikuti petunjuk di [Salin](#) data ke DynamoDB.

Langkah 4: Kueri data dari DynamoDB

Gunakan Spark atau Hive untuk menanyakan tabel DynamoDB Anda.

Spark

Untuk kueri data dari tabel DynamoDB yang Anda buat pada langkah sebelumnya, gunakan Spark SQL atau Spark API. MapReduce

Example— Kueri tabel DynamoDB Anda dengan Spark SQL

Query Spark SQL berikut mengembalikan daftar semua jenis fitur dalam urutan abjad.

```
val dataframe = sparkSession.sql("SELECT DISTINCT feature_class \
FROM ddb_features \
ORDER BY feature_class;")
```

Query Spark SQL berikut mengembalikan daftar semua danau yang dimulai dengan huruf M.

```
val dataframe = sparkSession.sql("SELECT feature_name, state_alpha \
FROM ddb_features \
WHERE feature_class = 'Lake' \
AND feature_name LIKE 'M%' \
ORDER BY feature_name;")
```

Kueri SQL Spark berikut mengembalikan daftar semua negara bagian dengan setidaknya tiga fitur yang lebih tinggi dari satu mil.

```
val dataframe = sparkSession.dql("SELECT state_alpha, feature_class, COUNT(*) \
FROM ddb_features \
WHERE elev_in_ft > 5280 \
GROUP by state_alpha, feature_class \
HAVING COUNT(*) >= 3 \
ORDER BY state_alpha, feature_class;")
```

Example— Kueri tabel DynamoDB Anda dengan Spark API MapReduce

MapReduce Query berikut mengembalikan daftar semua jenis fitur dalam urutan abjad.

```
val df = sc.hadoopRDD(jobConf, classOf[DynamoDBInputFormat], classOf[Text],
classOf[DynamoDBItemWritable])
.map(pair => (pair._1, pair._2.getItem))
```

```
.map(pair => pair._2.get("feature_class").getS)
.distinct()
.sortBy(value => value)
.toDF("feature_class")
```

MapReduce Query berikut mengembalikan daftar semua danau yang dimulai dengan huruf M.

```
val df = sc.hadoopRDD(jobConf, classOf[DynamoDBInputFormat], classOf[Text],
  classOf[DynamoDBItemWritable])
  .map(pair => (pair._1, pair._2.getItem))
  .filter(pair => "Lake".equals(pair._2.get("feature_class").getS))
  .filter(pair => pair._2.get("feature_name").getS.startsWith("M"))
  .map(pair => (pair._2.get("feature_name").getS,
    pair._2.get("state_alpha").getS))
  .sortBy(_._1)
  .toDF("feature_name", "state_alpha")
```

MapReduce Kueri berikut mengembalikan daftar semua negara bagian dengan setidaknya tiga fitur yang lebih tinggi dari satu mil.

```
val df = sc.hadoopRDD(jobConf, classOf[DynamoDBInputFormat], classOf[Text],
  classOf[DynamoDBItemWritable])
  .map(pair => pair._2.getItem)
  .filter(pair => pair.get("elev_in_ft").getN != null)
  .filter(pair => Integer.parseInt(pair.get("elev_in_ft").getN) > 5280)
  .groupBy(pair => (pair.get("state_alpha").getS, pair.get("feature_class").getS))
  .filter(pair => pair._2.size >= 3)
  .map(pair => (pair._1._1, pair._1._2, pair._2.size))
  .sortBy(pair => (pair._1, pair._2))
  .toDF("state_alpha", "feature_class", "count")
```

Hive

Untuk kueri data dari tabel DynamoDB yang Anda buat pada langkah sebelumnya, ikuti petunjuk di [Kueri data dalam tabel DynamoDB](#).

Menyiapkan akses lintas akun

Untuk mengatur akses lintas akun untuk EMR Tanpa Server, selesaikan langkah-langkah berikut. Dalam contoh, AccountA adalah akun tempat Anda membuat aplikasi Amazon EMR Tanpa Server, dan AccountB merupakan akun tempat Amazon DynamoDB Anda berada.

1. Buat tabel DynamoDB di AccountB Untuk informasi selengkapnya, lihat [Langkah 1: Buat tabel](#).
2. Buat peran Cross-Account-Role-B IAM AccountB yang dapat mengakses tabel DynamoDB.
 - a. Masuk ke Konsol Manajemen AWS dan buka konsol IAM di <https://console.aws.amazon.com/iam/>.
 - b. Pilih Peran, dan buat peran baru yang disebut Cross-Account-Role-B. Untuk informasi selengkapnya tentang cara membuat peran IAM, lihat [Membuat peran IAM](#) di Panduan pengguna.
 - c. Buat kebijakan IAM yang memberikan izin untuk mengakses tabel DynamoDB lintas akun. Kemudian lampirkan kebijakan IAM ke Cross-Account-Role-B.

Berikut ini adalah kebijakan yang memberikan akses ke tabel DynamoDB.
CrossAccountTable

- d. Cara mengedit hubungan kepercayaan untuk peran Cross-Account-Role-B.

Untuk mengonfigurasi hubungan kepercayaan untuk peran tersebut, pilih tab Trust Relationships di konsol IAM untuk peran yang Anda buat di Langkah 2: Cross-Account-Role-B.

Pilih Edit Trust Relationship dan kemudian tambahkan dokumen kebijakan berikut. Dokumen ini memungkinkan Job-Execution-Role-A AccountA untuk mengambil Cross-Account-Role-B peran 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"
    }
  ]
}
```

```
}

```

- e. Berikan Job-Execution-Role-A - STS Assume role izin untuk berasumsiCross-Account-Role-B. AccountA

Di konsol IAM untuk Akun AWS AccountA, pilihJob-Execution-Role-A. 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::123456789012:role/Cross-Account-Role-B"
      ],
      "Sid": "AllowSTSAssumerole"
    }
  ]
}
```

- f. Tetapkan dynamodb.customAWSCredentialsProvider properti dengan nilai seperti com.amazonaws.emr.AssumeRoleAWSCredentialsProvider dalam klasifikasi inti-situs. Atur variabel lingkungan ASSUME_ROLE_CREDENTIALS_ROLE_ARN dengan nilai ARN dari Cross-Account-Role-B
3. Jalankan Spark atau Hive job menggunakan. Job-Execution-Role-A

Pertimbangan-pertimbangan

Perhatikan perilaku dan batasan ini saat Anda menggunakan konektor DynamoDB dengan Apache Spark atau Apache Hive.

Pertimbangan saat menggunakan konektor DynamoDB dengan Apache Spark

- Spark SQL tidak mendukung pembuatan tabel Hive dengan opsi penanganan penyimpanan. Untuk informasi selengkapnya, lihat [Menentukan format penyimpanan untuk tabel Hive dalam dokumentasi](#) Apache Spark.
- Spark SQL tidak mendukung STORED BY operasi dengan handler penyimpanan. Jika Anda ingin berinteraksi dengan tabel DynamoDB melalui tabel Hive eksternal, gunakan Hive untuk membuat tabel terlebih dahulu.
- Untuk menerjemahkan kueri ke query DynamoDB, konektor DynamoDB menggunakan pushdown predikat. Predikat pushdown memfilter data dengan kolom yang dipetakan ke kunci partisi dari tabel DynamoDB. Predikat pushdown hanya beroperasi saat Anda menggunakan konektor dengan Spark SQL, dan bukan dengan API. MapReduce

Pertimbangan saat menggunakan konektor DynamoDB dengan Apache Hive

Menyetel jumlah maksimum mapper

- Jika Anda menggunakan SELECT kueri untuk membaca data dari tabel Hive eksternal yang memetakan ke DynamoDB, jumlah tugas peta di EMR Tanpa Server dihitung sebagai total throughput baca yang dikonfigurasi untuk tabel DynamoDB, dibagi dengan throughput per tugas peta. Throughput default per tugas peta adalah 100.
- Pekerjaan Hive dapat menggunakan jumlah tugas peta di luar jumlah maksimum kontainer yang dikonfigurasi per aplikasi EMR Tanpa Server, tergantung pada throughput baca yang dikonfigurasi untuk DynamoDB. Selain itu, kueri Hive yang berjalan lama dapat menggunakan semua kapasitas baca yang disediakan dari tabel DynamoDB. Ini berdampak negatif pada pengguna lain.
- Anda dapat menggunakan `dynamodb.max.map.tasks` properti untuk menetapkan batas atas untuk tugas peta. Anda juga dapat menggunakan properti ini untuk menyetel jumlah data yang dibaca oleh setiap tugas peta berdasarkan ukuran wadah tugas.
- Anda dapat mengatur `dynamodb.max.map.tasks` properti di tingkat kueri Hive, atau dalam `hive-site` klasifikasi `start-job-run` perintah. Nilai ini harus lebih besar atau sama dengan 1. Saat Hive memproses kueri Anda, pekerjaan Hive yang dihasilkan menggunakan tidak lebih dari nilai `dynamodb.max.map.tasks` saat membaca dari tabel DynamoDB.

Menyetel throughput tulis per tugas

- Tulis throughput per tugas pada EMR Tanpa Server dihitung sebagai total throughput tulis yang dikonfigurasi untuk tabel DynamoDB, dibagi dengan nilai properti `mapreduce.job.maps`. Untuk Hive, nilai default properti ini adalah 2. Oleh karena itu, dua tugas pertama di tahap akhir pekerjaan Hive dapat menghabiskan semua throughput penulisan. Hal ini menyebabkan pembatasan penulisan tugas lain dalam pekerjaan yang sama atau pekerjaan lain.
- Untuk menghindari pembatasan penulisan, tetapkan nilai `mapreduce.job.maps` properti berdasarkan jumlah tugas di tahap akhir atau throughput tulis yang ingin Anda alokasikan per tugas. Tetapkan properti ini dalam `mapred-site` klasifikasi `start-job-run` perintah pada EMR Serverless.

Keamanan

Keamanan cloud di AWS adalah prioritas tertinggi. Sebagai AWS pelanggan, Anda mendapat manfaat dari pusat data dan arsitektur jaringan yang dibangun untuk memenuhi persyaratan organisasi yang paling sensitif terhadap keamanan.

Keamanan adalah tanggung jawab bersama antara Anda AWS dan Anda. [Model tanggung jawab bersama](#) menggambarkan hal ini sebagai keamanan dari cloud dan keamanan di cloud:

- Keamanan cloud — AWS bertanggung jawab untuk melindungi infrastruktur yang menjalankan AWS layanan di AWS Cloud. AWS juga memberi Anda layanan yang Anda gunakan dengan aman. Auditor pihak ketiga secara berkala menguji dan memverifikasi efektivitas keamanan kami sebagai bagian dari [Program kepatuhan AWS](#). Untuk mempelajari tentang program kepatuhan yang berlaku untuk Amazon EMR Tanpa Server, lihat [AWS layanan dalam cakupan](#) berdasarkan program kepatuhan.
- Keamanan di cloud — Tanggung jawab Anda ditentukan oleh AWS layanan yang Anda gunakan. Anda juga bertanggung jawab atas faktor lain, termasuk sensitivitas data 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 Tanpa Server. Topik menjelaskan cara mengonfigurasi Amazon EMR Tanpa Server dan menggunakan AWS layanan lain untuk memenuhi tujuan keamanan dan kepatuhan Anda.

Topik

- [Praktik terbaik keamanan untuk Amazon EMR Tanpa Server](#)
- [Perlindungan data](#)
- [Identity and Access Management \(IAM\) di Amazon EMR Tanpa Server](#)
- [Propagasi Identitas Tepercaya](#)
- [Menggunakan Lake Formation dengan EMR Serverless](#)
- [Enkripsi antar pekerja](#)
- [Enkripsi Disk dengan KMS CMK](#)
- [Secrets Manager untuk perlindungan data dengan EMR Serverless](#)
- [Menggunakan Hibah Akses Amazon S3 dengan EMR Tanpa Server](#)

- [Mencatat panggilan API Tanpa Server Amazon EMR menggunakan AWS CloudTrail](#)
- [Validasi kepatuhan untuk Amazon EMR Tanpa Server](#)
- [Ketahanan di Amazon EMR Tanpa Server](#)
- [Keamanan infrastruktur di Amazon EMR Tanpa Server](#)
- [Analisis konfigurasi dan kerentanan di Amazon EMR Tanpa Server](#)

Praktik terbaik keamanan untuk Amazon EMR Tanpa Server

Amazon EMR Tanpa Server menyediakan sejumlah fitur keamanan untuk dipertimbangkan saat Anda mengembangkan dan menerapkan kebijakan keamanan Anda sendiri. Praktik terbaik berikut adalah pedoman umum dan tidak mewakili solusi keamanan yang lengkap. Karena praktik terbaik ini mungkin tidak sesuai atau cukup untuk lingkungan Anda, anggap sebagai pertimbangan yang membantu dan bukan sebagai resep.

Terapkan prinsip hak istimewa paling rendah

EMR Tanpa Server menyediakan kebijakan akses terperinci untuk aplikasi yang menggunakan peran IAM, seperti peran eksekusi. Kami menyarankan agar peran eksekusi diberikan hanya set minimum hak istimewa yang diperlukan oleh pekerjaan, 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.

Mengisolasi kode aplikasi yang tidak tepercaya

EMR Tanpa Server menciptakan isolasi jaringan penuh antara pekerjaan milik aplikasi EMR Tanpa Server yang berbeda. Dalam kasus di mana isolasi tingkat pekerjaan diinginkan, pertimbangkan untuk mengisolasi pekerjaan ke dalam aplikasi EMR Tanpa Server yang berbeda.

Izin kontrol akses berbasis peran (RBAC)

Administrator harus secara ketat mengontrol izin kontrol akses berbasis peran (RBAC) untuk aplikasi EMR Tanpa Server.

Perlindungan data

[Model tanggung jawab AWS bersama](#) berlaku untuk perlindungan data di Amazon EMR Tanpa Server. Seperti yang dijelaskan dalam model AWS ini, bertanggung jawab untuk melindungi

infrastruktur global yang menjalankan semua AWS Cloud. Anda bertanggung jawab untuk mempertahankan kendali atas konten yang di-host pada infrastruktur ini. Konten ini mencakup konfigurasi keamanan dan tugas manajemen untuk AWS layanan yang Anda gunakan. Untuk informasi lebih lanjut 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 agar Anda 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 menyarankan TLS 1.2 atau 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 lanjutan seperti Amazon Macie, yang membantu menemukan dan mengamankan data pribadi yang disimpan di Amazon S3.
- Gunakan opsi enkripsi Amazon EMR Tanpa Server untuk mengenkripsi data saat istirahat dan dalam perjalanan.
- 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 [Federal Information Processing Standard \(FIPS\) 140-2](#).

Kami sangat menyarankan agar Anda tidak pernah memasukkan informasi identifikasi sensitif, seperti nomor akun pelanggan Anda, ke dalam bidang bentuk bebas seperti bidang Nama. Ini termasuk saat Anda bekerja dengan Amazon EMR Tanpa Server atau AWS layanan lain menggunakan konsol, API, AWS CLI atau SDK. AWS Data apa pun yang Anda masukkan ke Amazon EMR Tanpa Server atau layanan lain mungkin diambil untuk dimasukkan 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

Setiap aplikasi EMR Tanpa Server menggunakan versi rilis tertentu, yang mencakup EMRFS (EMR File System). Enkripsi Amazon S3 bekerja dengan objek EMR File System (EMRFS) yang dibaca dari dan ditulis ke Amazon S3. Anda dapat menentukan enkripsi sisi server Amazon S3 (SSE) atau enkripsi sisi klien (CSE) sebagai mode enkripsi Default saat Anda mengaktifkan enkripsi saat istirahat. Secara opsional, tentukan metode enkripsi yang berbeda untuk masing-masing bucket menggunakan penggantian enkripsi Per 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. Jika Anda menggunakan Amazon S3 CSE dengan kunci yang dikelola pelanggan, peran eksekusi yang digunakan untuk menjalankan pekerjaan di aplikasi EMR Tanpa Server harus memiliki akses ke kunci tersebut. Untuk informasi mendalam tentang enkripsi Amazon S3, lihat [Melindungi data menggunakan enkripsi di Panduan Pengembang Layanan Penyimpanan Sederhana Amazon](#).

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

Semua bucket Amazon S3 memiliki enkripsi yang dikonfigurasi secara default, dan semua objek baru yang diunggah ke bucket S3 secara otomatis dienkripsi saat istirahat, Amazon S3 mengenkripsi data pada tingkat objek saat menulis data ke disk dan mendekripsi data saat diakses. Untuk informasi selengkapnya tentang SSE, lihat [Melindungi data menggunakan enkripsi sisi server di Panduan Pengembangan Layanan Penyimpanan Sederhana Amazon](#).

Anda dapat memilih antara dua sistem manajemen kunci yang berbeda saat Anda menentukan SSE di Amazon EMR Tanpa Server:

- SSE-S3- Amazon S3 mengelola kunci untuk Anda. Tidak ada pengaturan tambahan yang diperlukan di EMR Tanpa Server.
- SSE-KMS- Anda menggunakan sebuah AWS KMS key untuk mengatur dengan kebijakan yang sesuai untuk EMR Tanpa Server. Tidak ada pengaturan tambahan yang diperlukan di EMR Tanpa Server.

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.

Untuk menggunakan AWS KMS enkripsi untuk data yang Anda tulis ke Amazon S3, Anda memiliki dua opsi saat menggunakan API. `StartJobRun` Anda dapat mengaktifkan enkripsi untuk semua yang Anda tulis ke Amazon S3, atau mengaktifkan enkripsi untuk data yang Anda tulis ke bucket tertentu. Untuk informasi selengkapnya tentang `StartJobRun` API, lihat Referensi API [EMR Tanpa Server](#).

Untuk mengaktifkan AWS KMS enkripsi untuk semua data yang Anda tulis ke Amazon S3, gunakan perintah berikut saat Anda memanggil API. `StartJobRun`

```
--conf spark.hadoop.fs.s3.enableServerSideEncryption=true
--conf spark.hadoop.fs.s3.serverSideEncryption.kms.keyId=<kms_id>
```

Untuk mengaktifkan AWS KMS enkripsi data yang Anda tulis ke bucket tertentu, gunakan perintah berikut saat memanggil StartJobRun API.

```
--conf spark.hadoop.fs.s3.bucket.<amzn-s3-demo-bucket1>.enableServerSideEncryption=true
--conf spark.hadoop.fs.s3.bucket.<amzn-s3-demo-bucket1>.serverSideEncryption.kms.keyId=<kms-id>
```

SSE dengan kunci yang disediakan pelanggan (SSE-C) tidak tersedia untuk digunakan dengan EMR Tanpa Server.

Enkripsi di sisi klien Amazon S3

Dengan enkripsi sisi klien Amazon S3, enkripsi dan dekripsi Amazon S3 berlangsung di klien EMRFS yang tersedia di setiap rilis EMR Amazon. 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. Jika Anda menggunakan Amazon S3 CSE dengan kunci yang dikelola pelanggan, peran eksekusi yang digunakan untuk menjalankan pekerjaan di aplikasi EMR Tanpa Server harus memiliki akses ke kunci tersebut. Biaya tambahan KMS mungkin berlaku. Untuk informasi selengkapnya tentang perbedaan ini, lihat [Melindungi data menggunakan enkripsi sisi klien di Panduan](#) Pengembang Layanan Penyimpanan Sederhana Amazon.

Enkripsi disk lokal

Data yang disimpan dalam penyimpanan sementara dienkripsi dengan kunci yang dimiliki layanan menggunakan algoritma kriptografi standar industri. AES-256

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 kunci yang dikelola pelanggan](#).

Enkripsi saat bergerak

Fitur enkripsi khusus aplikasi berikut tersedia dengan Amazon EMR Tanpa Server:

- Spark
 - Secara default, komunikasi antara driver Spark dan pelaksana diautentikasi dan internal. Komunikasi RPC antara driver dan pelaksana dienkripsi.
- Hive
 - Komunikasi antara aplikasi AWS Glue metastore dan EMR Serverless terjadi melalui TLS.

Anda harus mengizinkan hanya koneksi terenkripsi melalui HTTPS (TLS) menggunakan [aws:SecureTransport condition pada kebijakan IAM](#) bucket Amazon S3.

Identity and Access Management (IAM) di Amazon EMR Tanpa Server

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 Tanpa Server. IAM adalah Layanan AWS yang dapat Anda gunakan tanpa biaya tambahan.

Topik

- [Audiens](#)
- [Mengautentikasi dengan identitas](#)
- [Mengelola akses menggunakan kebijakan](#)
- [Bagaimana EMR Serverless bekerja dengan IAM](#)
- [Menggunakan peran terkait layanan untuk EMR Tanpa Server](#)
- [Peran runtime Job untuk Amazon EMR Tanpa Server](#)
- [Contoh kebijakan akses pengguna untuk EMR Tanpa Server](#)
- [Kebijakan untuk kendali akses berbasis tanda](#)
- [Identity-based contoh kebijakan untuk EMR Tanpa Server](#)
- [Amazon EMR Pembaruan tanpa server ke kebijakan terkelola AWS](#)
- [Memecahkan masalah Amazon EMR Identitas dan akses tanpa server](#)

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 Amazon EMR Identitas dan akses tanpa server](#))
- Administrator layanan - tentukan akses pengguna dan mengirimkan permintaan izin (lihat [Identity and Access Management \(IAM\) di Amazon EMR Tanpa Server](#))
- Administrator IAM - tulis kebijakan untuk mengelola akses (lihat [Contoh kebijakan berbasis identitas untuk EMR Tanpa Server](#))

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 kredensial dari sumber identitas. Identitas terfederasi mengambil peran yang memberikan kredensial sementara.

Untuk manajemen akses terpusat, kami menyarankan AWS IAM Identity Center. Untuk informasi selengkapnya, lihat [Apa itu Pusat Identitas IAM?](#) dalam Panduan Pengguna AWS IAM Identity Center

Pengguna dan grup IAM

[Pengguna IAM](#) adalah identitas dengan izin khusus untuk satu orang atau aplikasi. Sebaiknya gunakan kredensial sementara alih-alih pengguna IAM dengan kredensial jangka panjang. Untuk informasi selengkapnya, lihat [Mewajibkan pengguna manusia untuk menggunakan federasi dengan penyedia identitas untuk mengakses AWS menggunakan kredensi sementara](#) di Panduan Pengguna IAM.

[Grup IAM](#) menentukan kumpulan pengguna IAM dan mempermudah pengelolaan izin untuk pengguna dalam jumlah besar. Untuk mempelajari selengkapnya, lihat [Kasus penggunaan untuk pengguna IAM](#) dalam Panduan Pengguna IAM.

Peran IAM

[Peran IAM](#) adalah identitas dengan izin khusus yang menyediakan kredensial sementara. Anda dapat mengambil peran dengan [beralih dari pengguna ke peran IAM \(konsol\)](#) atau dengan memanggil operasi AWS CLI atau AWS API. Untuk informasi selengkapnya, lihat [Metode untuk mengambil peran](#) dalam Panduan Pengguna IAM.

Peran IAM berguna untuk akses pengguna terfederasi, izin pengguna IAM sementara, akses lintas akun, akses lintas layanan, dan aplikasi yang berjalan di Amazon EC2. Untuk informasi selengkapnya, lihat [Akses sumber daya lintas akun di IAM](#) dalam Panduan Pengguna IAM.

Mengelola akses menggunakan kebijakan

Anda mengontrol akses AWS dengan membuat kebijakan dan melampirkannya ke AWS identitas atau sumber daya. Kebijakan menentukan izin saat dikaitkan dengan identitas atau sumber daya. AWS mengevaluasi kebijakan ini ketika kepala sekolah membuat permintaan. Sebagian besar kebijakan disimpan AWS sebagai dokumen JSON. Untuk informasi selengkapnya tentang dokumen kebijakan JSON, lihat [Gambaran umum kebijakan JSON](#) dalam Panduan Pengguna IAM.

Menggunakan kebijakan, administrator menentukan siapa yang memiliki akses ke apa dengan mendefinisikan principal mana yang dapat melakukan tindakan pada sumber daya apa, dan dalam kondisi apa.

Secara default, pengguna dan peran tidak memiliki izin. Administrator IAM membuat kebijakan IAM dan menambahkannya ke peran, yang kemudian dapat diambil oleh pengguna. Kebijakan IAM mendefinisikan izin terlepas dari metode yang Anda gunakan untuk melakukannya.

Identity-based kebijakan

Identity-based kebijakan adalah dokumen kebijakan izin JSON yang Anda lampirkan ke identitas (pengguna, grup, atau peran). Kebijakan ini mengontrol tindakan apa yang bisa dilakukan oleh identitas tersebut, terhadap sumber daya yang mana, dan dalam kondisi apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat [Tentukan izin IAM kustom dengan kebijakan yang dikelola pelanggan](#) dalam Panduan Pengguna IAM.

Identity-based kebijakan dapat berupa kebijakan inline (disematkan langsung ke dalam satu identitas) atau kebijakan terkelola (kebijakan mandiri yang dilampirkan pada beberapa identitas). Untuk mempelajari cara memilih antara kebijakan terkelola dan kebijakan inline, lihat [Pilih antara kebijakan terkelola dan kebijakan inline](#) dalam Panduan Pengguna IAM.

Resource-based kebijakan

Resource-based kebijakan adalah dokumen kebijakan JSON yang Anda lampirkan ke sumber daya. Contohnya termasuk kebijakan kepercayaan peran IAM dan kebijakan bucket Amazon S3. Dalam layanan yang mendukung kebijakan berbasis sumber daya, administrator layanan dapat menggunakannya untuk mengontrol akses ke sumber daya tertentu. Anda harus [menentukan principal](#) dalam kebijakan berbasis sumber daya.

Resource-based kebijakan adalah kebijakan inline yang terletak di layanan tersebut. Anda tidak dapat menggunakan kebijakan AWS terkelola dari IAM dalam kebijakan berbasis sumber daya.

Jenis-jenis kebijakan lain

AWS mendukung jenis kebijakan tambahan yang dapat menetapkan izin maksimum yang diberikan oleh jenis kebijakan yang lebih umum:

- Batasan izin – Menetapkan izin maksimum yang dapat diberikan oleh kebijakan berbasis identitas kepada entitas IAM. Untuk informasi selengkapnya, lihat [Batasan izin untuk entitas IAM](#) dalam Panduan Pengguna IAM.

- Kebijakan kontrol layanan (SCP) – Menentukan izin maksimum untuk organisasi atau unit organisasi di AWS Organizations. Untuk informasi selengkapnya, lihat [Kebijakan kontrol layanan](#) dalam Panduan Pengguna AWS Organizations .
- Kebijakan kontrol sumber daya (RCP) – Menetapkan izin maksimum yang tersedia untuk sumber daya di akun Anda. Untuk informasi selengkapnya, lihat [Kebijakan kontrol sumber daya \(RCP\)](#) dalam Panduan Pengguna AWS Organizations .
- Kebijakan sesi – Kebijakan lanjutan yang diteruskan sebagai parameter saat membuat sesi sementara untuk peran atau pengguna terfederasi. Untuk informasi selengkapnya, lihat [Kebijakan sesi](#) dalam Panduan Pengguna IAM.

Berbagai jenis kebijakan

Ketika beberapa jenis kebijakan berlaku pada suatu permintaan, izin yang dihasilkan lebih rumit untuk dipahami. Untuk mempelajari cara AWS menentukan apakah akan mengizinkan permintaan saat beberapa jenis kebijakan terlibat, lihat [Logika evaluasi kebijakan](#) di Panduan Pengguna IAM.

Bagaimana EMR Serverless bekerja dengan IAM

Sebelum Anda menggunakan IAM untuk mengelola akses ke Amazon EMR Tanpa Server, pelajari fitur IAM apa yang tersedia untuk digunakan dengan Amazon EMR Tanpa Server.

Fitur IAM digunakan dengan EMR Tanpa Server

Fitur IAM	Dukungan Amazon EMR Tanpa Server
Kebijakan Identity-based	Ya
Kebijakan Resource-based	Tidak
Tindakan kebijakan	Ya
Sumber daya kebijakan	Ya
Kunci kondisi kebijakan	Tidak
ACL	Tidak
ABAC (tanda dalam kebijakan)	Ya
Kredensial sementara	Ya

Fitur IAM	Dukungan Amazon EMR Tanpa Server
Izin principal	Ya
Peran layanan	Tidak
Peran Service-linked	Ya

Untuk mendapatkan tampilan tingkat tinggi tentang cara EMR Tanpa Server dan layanan AWS lainnya bekerja dengan sebagian besar fitur IAM, lihat AWS layanan yang [bekerja dengan IAM di Panduan Pengguna](#) IAM.

Identity-based kebijakan untuk EMR Serverless

Mendukung kebijakan berbasis identitas: Ya

Identity-based kebijakan adalah dokumen kebijakan izin JSON yang dapat Anda lampirkan ke identitas, seperti pengguna IAM, grup pengguna, atau peran. Kebijakan ini mengontrol jenis tindakan yang dapat dilakukan oleh pengguna dan peran, di sumber daya mana, dan berdasarkan kondisi seperti apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat [Tentukan izin IAM kustom dengan kebijakan terkelola pelanggan](#) dalam Panduan Pengguna IAM.

Dengan kebijakan berbasis identitas IAM, Anda dapat menentukan secara spesifik apakah tindakan dan sumber daya diizinkan atau ditolak, serta kondisi yang menjadi dasar dikabulkan atau ditolaknya tindakan tersebut. Untuk mempelajari semua elemen yang dapat Anda gunakan dalam kebijakan JSON, lihat [Referensi elemen kebijakan JSON IAM](#) dalam Panduan Pengguna IAM.

Contoh kebijakan berbasis identitas untuk EMR Tanpa Server

Untuk mengakses contoh kebijakan berbasis identitas Amazon EMR Tanpa Server, lihat. [Identity-based contoh kebijakan untuk EMR Tanpa Server](#)

Resource-based kebijakan dalam EMR Tanpa Server

Mendukung kebijakan berbasis sumber daya: Tidak

Resource-based kebijakan adalah dokumen kebijakan JSON yang Anda lampirkan ke sumber daya. Contoh kebijakan berbasis sumber daya adalah kebijakan kepercayaan peran IAM dan kebijakan bucket Amazon S3. Dalam layanan yang mendukung kebijakan berbasis sumber daya, administrator

layanan dapat menggunakannya untuk mengontrol akses ke sumber daya tertentu. Untuk sumber daya tempat kebijakan dilampirkan, kebijakan menentukan tindakan apa yang dapat dilakukan oleh principal tertentu pada sumber daya tersebut dan dalam kondisi apa. Anda harus [menentukan principal](#) dalam kebijakan berbasis sumber daya. Prinsipal dapat mencakup akun, pengguna, peran, pengguna federasi, atau. Layanan AWS

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 EMR Tanpa Server

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 merujuk ke daftar tindakan EMR Tanpa Server, lihat Tindakan, [sumber daya, dan kunci kondisi untuk Amazon EMR Tanpa Server](#) di Referensi Otorisasi Layanan.

Tindakan kebijakan di EMR Tanpa Server menggunakan awalan berikut sebelum tindakan.

```
emr-serverless
```

Untuk menetapkan secara spesifik beberapa tindakan dalam satu pernyataan, pisahkan tindakan tersebut dengan koma.

```
"Action": [  
  "emr-serverless:action1",  
  "emr-serverless:action2"  
]
```

Untuk mengakses contoh kebijakan berbasis identitas Amazon EMR Tanpa Server, lihat. [Identity-based contoh kebijakan untuk EMR Tanpa Server](#)

Sumber daya kebijakan untuk EMR Tanpa Server

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 merujuk ke daftar jenis sumber daya Amazon EMR Tanpa Server dan ARNnya, lihat Sumber daya yang ditentukan [oleh Amazon EMR Tanpa Server](#) dalam Referensi Otorisasi Layanan. Untuk mempelajari tindakan mana yang menentukan ARN dari setiap sumber daya, lihat [Tindakan, sumber daya, dan kunci kondisi untuk Amazon EMR Tanpa Server](#).

Untuk mengakses contoh kebijakan berbasis identitas Amazon EMR Tanpa Server, lihat. [Identity-based contoh kebijakan untuk EMR Tanpa Server](#)

Kunci kondisi kebijakan untuk EMR Tanpa Server

Dukungan kunci kondisi kebijakan

Mendukung kunci kondisi kebijakan khusus layanan	Tidak
--------------------------------------------------	-------

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 merujuk ke daftar kunci kondisi Amazon EMR Tanpa Server dan untuk mempelajari tindakan dan sumber daya yang dapat Anda gunakan kunci kondisi, lihat Tindakan, sumber [daya, dan kunci kondisi untuk Amazon EMR Tanpa Server](#) di Referensi Otorisasi Layanan.

Semua tindakan Amazon EC2 mendukung kunci syarat `aws:RequestedRegion` dan `ec2:Region`. Untuk informasi selengkapnya, lihat [Contoh: Membatasi akses ke wilayah tertentu](#).

Daftar kontrol akses (ACL) di EMR Tanpa Server

Mendukung ACL: Tidak

Daftar kontrol akses (ACL) mengendalikan principal mana (anggota akun, pengguna, atau peran) yang memiliki izin untuk mengakses sumber daya. ACL serupa dengan kebijakan berbasis sumber daya, meskipun kebijakan tersebut tidak menggunakan format dokumen kebijakan JSON.

Attribute-based kontrol akses (ABAC) dengan EMR Tanpa Server

Attribute-based dukungan kontrol akses (ABAC)

Mendukung ABAC (tanda dalam kebijakan)	Ya
----------------------------------------	----

Attribute-based Access Control (ABAC) adalah strategi otorisasi yang mendefinisikan izin berdasarkan atribut yang disebut tag. Anda dapat melampirkan tag ke entitas dan AWS sumber daya IAM, lalu merancang kebijakan ABAC untuk mengizinkan operasi saat tag prinsipal cocok dengan tag pada sumber daya.

Untuk mengendalikan akses berdasarkan tanda, berikan informasi tentang tanda di [elemen kondisi](#) dari kebijakan menggunakan kunci kondisi `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, atau `aws:TagKeys`.

Jika sebuah layanan mendukung ketiga kunci kondisi untuk setiap jenis sumber daya, nilainya adalah Ya untuk layanan tersebut. Jika suatu layanan mendukung ketiga kunci kondisi untuk hanya beberapa jenis sumber daya, nilainya adalah Parsial.

Untuk informasi selengkapnya tentang ABAC, lihat [Tentukan izin dengan otorisasi ABAC](#) dalam Panduan Pengguna IAM. Untuk melihat tutorial yang menguraikan langkah-langkah pengaturan ABAC, lihat [Menggunakan kontrol akses berbasis atribut \(ABAC\)](#) dalam Panduan Pengguna IAM.

Menggunakan kredensial Sementara dengan EMR Serverless

Mendukung kredensial sementara: Ya

Kredensi sementara menyediakan akses jangka pendek ke AWS sumber daya dan secara otomatis dibuat saat Anda menggunakan federasi atau beralih peran. AWS merekomendasikan agar Anda secara dinamis menghasilkan kredensi sementara alih-alih menggunakan kunci akses jangka panjang. Untuk informasi selengkapnya, lihat [Kredensial keamanan sementara di IAM](#) dan [Layanan AWS yang berfungsi dengan IAM](#) dalam Panduan Pengguna IAM.

Cross-service izin utama untuk EMR Tanpa Server

Mendukung sesi akses terusan (FAS): Ya

Sesi akses terusan (FAS) menggunakan izin 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 EMR Tanpa Server

Mendukung peran layanan	Tidak
-------------------------	-------

Service-linked peran untuk EMR Tanpa Server

Mendukung peran terkait layanan	Ya
---------------------------------	----

Untuk detail tentang membuat atau mengelola peran terkait layanan, lihat [AWS layanan yang bekerja dengan IAM](#). Temukan layanan dalam tabel yang Yes menyertakan kolom Service-linked peran. Pilih tautan Ya untuk mengakses dokumentasi peran terkait layanan untuk layanan tersebut.

Menggunakan peran terkait layanan untuk EMR Tanpa Server

[Amazon EMR Tanpa Server menggunakan peran terkait layanan AWS Identity and Access Management \(IAM\)](#). Peran terkait layanan adalah jenis unik peran IAM yang ditautkan langsung ke EMR Tanpa Server. Peran terkait layanan telah ditentukan sebelumnya oleh EMR Tanpa Server dan mencakup semua izin yang diperlukan layanan untuk memanggil layanan lain atas nama Anda. AWS

Peran terkait layanan membuat pengaturan EMR Tanpa Server lebih mudah karena Anda tidak perlu menambahkan izin yang diperlukan secara manual. EMR Tanpa Server mendefinisikan izin peran terkait layanannya, dan kecuali ditentukan lain, hanya EMR Tanpa Server yang dapat mengambil perannya. Izin yang ditentukan mencakup kebijakan kepercayaan dan kebijakan izin, dan kebijakan izin tersebut tidak dapat dilampirkan ke entitas IAM lainnya.

Anda dapat menghapus peran tertaut layanan hanya setelah menghapus sumber daya terkait terlebih dahulu. Ini melindungi sumber daya EMR Tanpa Server karena Anda tidak dapat secara tidak sengaja menghapus izin untuk mengakses sumber daya.

Untuk informasi tentang layanan lain yang mendukung peran terkait layanan, lihat [AWS Layanan yang Bekerja dengan IAM](#) dan periksa layanan yang memiliki Ya di kolom Peran terkait Layanan. Pilih Ya dengan tautan untuk mengakses dokumentasi peran terkait layanan untuk layanan tersebut.

Izin peran terkait layanan untuk EMR Tanpa Server

EMR Tanpa Server menggunakan peran terkait layanan bernama `AWSServiceRoleForAmazonEMRServerless` untuk memungkinkannya memanggil atas nama Anda. AWS APIs

Peran `AWSServiceRoleForAmazonEMRServerless` terkait layanan mempercayai layanan berikut untuk mengambil peran:

- `ops.emr-serverless.amazonaws.com`

Kebijakan izin peran bernama `AmazonEMRServerlessServiceRolePolicy` memungkinkan EMR Tanpa Server menyelesaikan tindakan berikut pada sumber daya yang ditentukan.

Note

Konten kebijakan terkelola berubah, sehingga kebijakan yang ditampilkan di sini mungkin kedaluwarsa. Lihat up-to-date kebijakan [Amazon](#) terbanyak `EMRServerlessServiceRolePolicy` di Konsol Manajemen AWS.

- Tindakan: `ec2:CreateNetworkInterface`
- Tindakan: `ec2>DeleteNetworkInterface`
- Tindakan: `ec2:DescribeNetworkInterfaces`

- Tindakan: `ec2:DescribeSecurityGroups`
- Tindakan: `ec2:DescribeSubnets`
- Tindakan: `ec2:DescribeVpcs`
- Tindakan: `ec2:DescribeDhcpOptions`
- Tindakan: `ec2:DescribeRouteTables`
- Tindakan: `cloudwatch:PutMetricData`

Berikut ini adalah `AmazonEMRServerlessServiceRolePolicy` kebijakan lengkapnya.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EC2PolicyStatement",
      "Effect": "Allow",
      "Action": [
        "ec2:CreateNetworkInterface",
        "ec2>DeleteNetworkInterface",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcs",
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeRouteTables"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "CloudWatchPolicyStatement",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

```

    "Condition": {
      "StringEquals": {
        "cloudwatch:namespace": [
          "AWS/EMRServerless",
          "AWS/Usage"
        ]
      }
    }
  ]
}

```

Kebijakan kepercayaan berikut dilampirkan pada peran ini untuk memungkinkan prinsipal EMR Tanpa Server untuk mengambil peran ini.

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "ops.emr-serverless.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}

```

Anda harus mengonfigurasi izin agar entitas IAM (seperti pengguna, grup, atau peran) dapat membuat, mengedit, atau menghapus peran terkait layanan. Untuk informasi selengkapnya, lihat [izin peran terkait layanan di Panduan Pengguna IAM](#).

Membuat peran terkait layanan untuk EMR Tanpa Server

Anda tidak perlu membuat peran terkait layanan secara manual. Saat Anda membuat aplikasi EMR Tanpa Server baru di (Konsol Manajemen AWS menggunakan EMR Studio), API, AWS CLI atau API

AWS , EMR Serverless membuat peran terkait layanan untuk Anda. Anda harus mengonfigurasi izin agar entitas IAM (seperti pengguna, grup, atau peran) dapat membuat, mengedit, atau menghapus peran terkait layanan.

Untuk membuat peran `AWSService RoleForAmazon EMRServerless` terkait layanan menggunakan IAM

Tambahkan pernyataan berikut ke kebijakan izin untuk entitas IAM yang perlu membuat peran terkait layanan.

```
{
  "Effect": "Allow",
  "Action": [
    "iam:CreateServiceLinkedRole"
  ],
  "Resource": "arn:aws:iam::*:role/aws-service-role/ops.emr-serverless.amazonaws.com/AWSServiceRoleForAmazonEMRServerless*",
  "Condition": {"StringLike": {"iam:AWSServiceName": "ops.emr-serverless.amazonaws.com"}}
}
```

Jika Anda menghapus peran terkait layanan ini, dan kemudian perlu membuatnya lagi, gunakan proses yang sama untuk membuat ulang peran di akun Anda. Saat Anda membuat aplikasi EMR Tanpa Server baru, EMR Serverless membuat peran terkait layanan untuk Anda lagi.

Anda juga dapat menggunakan konsol IAM untuk membuat peran terkait layanan dengan kasus penggunaan EMR Tanpa Server. Di AWS CLI atau AWS API, buat peran terkait layanan dengan nama `ops.emr-serverless.amazonaws.com` layanan. Untuk informasi selengkapnya, lihat [Membuat peran terkait layanan di Panduan Pengguna IAM](#). Jika Anda menghapus peran terkait layanan ini, gunakan proses yang sama ini untuk membuat peran lagi.

Mengedit peran terkait layanan untuk EMR Tanpa Server

EMR Tanpa Server tidak memungkinkan Anda mengedit peran `AWSService RoleForAmazon EMRServerless` terkait layanan karena berbagai entitas mungkin mereferensikan peran tersebut. Anda tidak dapat mengedit kebijakan IAM AWS milik yang digunakan peran terkait layanan EMR Tanpa Server, karena berisi semua izin yang diperlukan EMR Tanpa Server yang diperlukan. Namun, Anda dapat mengedit penjelasan peran menggunakan IAM.

Untuk mengedit deskripsi peran `AWSService RoleForAmazon EMRServerless` terkait layanan menggunakan IAM

Tambahkan pernyataan berikut ke kebijakan izin untuk entitas IAM yang perlu mengedit deskripsi peran terkait layanan.

```
{
  "Effect": "Allow",
  "Action": [
    "iam: UpdateRoleDescription"
  ],
  "Resource": "arn:aws:iam::*:role/aws-service-role/ops.emr-serverless.amazonaws.com/AWSServiceRoleForAmazonEMRServerless*",
  "Condition": {"StringLike": {"iam:AWSServiceName": "ops.emr-serverless.amazonaws.com"}}
}
```

Untuk informasi selengkapnya, lihat [Mengedit peran terkait layanan di Panduan Pengguna IAM](#).

Menghapus peran terkait layanan untuk EMR Tanpa Server

Jika Anda tidak perlu lagi menggunakan fitur atau layanan yang memerlukan peran terkait layanan, sebaiknya hapus peran tersebut. Ini agar Anda tidak memiliki entitas yang tidak terpakai yang tidak dipantau atau dipelihara secara aktif. Namun, hapus semua aplikasi EMR Tanpa Server di semua Wilayah sebelum menghapus peran terkait layanan.

Note

Jika layanan EMR Tanpa Server menggunakan peran saat Anda mencoba menghapus sumber daya yang terkait dengan peran tersebut, maka penghapusan mungkin gagal. Jika hal itu terjadi, tunggu beberapa menit dan coba mengoperasikannya lagi.

Untuk menghapus peran AWSService RoleForAmazon EMRServerless terkait layanan menggunakan IAM

Tambahkan pernyataan berikut ke kebijakan izin untuk entitas IAM yang perlu menghapus peran terkait layanan.

```
{
  "Effect": "Allow",
  "Action": [
    "iam:DeleteServiceLinkedRole",
  ]
}
```

```
    "iam:GetServiceLinkedRoleDeletionStatus"  
  ],  
  "Resource": "arn:aws:iam::*:role/aws-service-role/ops.emr-serverless.amazonaws.com/  
AWSServiceRoleForAmazonEMRServerless*",  
  "Condition": {"StringLike": {"iam:AWSServiceName": "ops.emr-  
serverless.amazonaws.com"}}  
}
```

Untuk menghapus peran tertaut layanan secara manual menggunakan IAM

Gunakan konsol IAM, the AWS CLI, atau AWS API untuk menghapus peran `AWSServiceRoleForAmazonEMRServerless` terkait layanan. Untuk informasi selengkapnya, lihat [Menghapus peran terkait layanan](#) di Panduan Pengguna IAM.

Wilayah yang Didukung untuk peran terkait layanan EMR Tanpa Server

EMR Tanpa Server mendukung penggunaan peran terkait layanan di semua Wilayah tempat layanan tersedia. Untuk informasi lebih lanjut, lihat [AWS Wilayah dan titik akhir](#).

Peran runtime Job untuk Amazon EMR Tanpa Server

Anda dapat menentukan izin peran IAM yang dapat diasumsikan oleh menjalankan pekerjaan EMR Tanpa Server saat memanggil layanan lain atas nama Anda. Ini termasuk akses ke Amazon S3 untuk sumber data, target, serta sumber AWS daya lain seperti cluster Amazon Redshift dan tabel DynamoDB. Untuk mempelajari lebih lanjut tentang cara membuat peran, lihat [Buat peran runtime pekerjaan](#).

Contoh kebijakan runtime

Anda dapat melampirkan kebijakan runtime, seperti berikut ini, ke peran runtime pekerjaan. Kebijakan runtime pekerjaan berikut memungkinkan:

- Baca akses ke bucket Amazon S3 dengan sampel EMR.
- Akses penuh ke ember S3.
- Buat dan baca akses ke AWS Glue Data Catalog.

Untuk menambahkan akses ke AWS sumber daya lain seperti DynamoDB, Anda harus menyertakan izin untuknya dalam kebijakan saat membuat peran runtime.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadAccessForEMRSamples",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3::*.elasticmapreduce",
        "arn:aws:s3::*.elasticmapreduce/*"
      ]
    },
    {
      "Sid": "FullAccessToS3Bucket",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:ListBucket",
        "s3:DeleteObject"
      ],
      "Resource": [
        "arn:aws:s3::amzn-s3-demo-bucket",
        "arn:aws:s3:::amzn-s3-demo-bucket/*"
      ]
    },
    {
      "Sid": "GlueCreateAndReadDataCatalog",
      "Effect": "Allow",
      "Action": [
        "glue:GetDatabase",
        "glue:CreateDatabase",
        "glue:GetDataBases",
        "glue:CreateTable",
        "glue:GetTable",
        "glue:UpdateTable",
        "glue>DeleteTable",
        "glue:GetTables",

```

```

    "glue:GetPartition",
    "glue:GetPartitions",
    "glue:CreatePartition",
    "glue:BatchCreatePartition",
    "glue:GetUserDefinedFunctions"
  ],
  "Resource": [
    "*"
  ]
}
]
}

```

Lulus hak istimewa peran

Anda dapat melampirkan kebijakan izin IAM ke peran pengguna untuk memungkinkan pengguna hanya meneruskan peran yang disetujui. Hal ini memungkinkan administrator untuk mengontrol pengguna mana yang dapat meneruskan peran runtime pekerjaan tertentu ke pekerjaan EMR Tanpa Server. Untuk mempelajari lebih lanjut tentang menyetel izin, lihat [Memberikan izin pengguna untuk meneruskan peran ke layanan](#). AWS

Berikut ini adalah contoh kebijakan yang memungkinkan meneruskan peran runtime pekerjaan ke kepala layanan EMR Tanpa Server.

```

{
  "Effect": "Allow",
  "Action": "iam:PassRole",
  "Resource": "arn:aws:iam::1234567890:role/JobRuntimeRoleForEMRServerless",
  "Condition": {
    "StringLike": {
      "iam:PassedToService": "emr-serverless.amazonaws.com"
    }
  }
}

```

Kebijakan izin terkelola yang terkait dengan peran runtime

Saat Anda mengirimkan pekerjaan berjalan ke EMR tanpa server melalui konsol EMR Studio, ada langkah di mana Anda memilih peran Runtime untuk dikaitkan dengan aplikasi Anda. Ada kebijakan terkelola mendasar yang terkait dengan setiap pilihan di konsol yang penting untuk diperhatikan. Tiga pilihan tersebut adalah sebagai berikut:

1. Semua bucket — Bila Anda memilih ini, ini menentukan kebijakan [AmazonS3FullAccess](#) AWS terkelola, yang menyediakan akses penuh ke semua bucket.
2. Bucket khusus — Ini menentukan pengenal nama sumber daya Amazon (ARN) untuk setiap bucket yang Anda pilih. Tidak ada kebijakan terkelola yang mendasarinya disertakan.
3. Tidak ada - Tidak ada izin kebijakan terkelola yang disertakan.

Kami menyarankan untuk menambahkan ember tertentu. Jika Anda memilih semua ember, perlu diingat bahwa itu menetapkan akses penuh untuk semua ember.

Contoh kebijakan akses pengguna untuk EMR Tanpa Server

Anda dapat menyiapkan kebijakan berbutir halus untuk pengguna Anda tergantung pada tindakan yang ingin dilakukan setiap pengguna saat berinteraksi dengan aplikasi EMR Tanpa Server. Kebijakan berikut adalah contoh yang mungkin membantu dalam menyiapkan izin yang sesuai untuk pengguna Anda. Bagian ini hanya berfokus pada kebijakan EMR Tanpa Server. Untuk contoh kebijakan pengguna EMR Studio, lihat Mengonfigurasi izin pengguna [EMR Studio](#). Untuk informasi tentang cara melampirkan kebijakan kepada pengguna IAM (prinsipal), lihat [Mengelola kebijakan IAM di Panduan Pengguna IAM](#).

Kebijakan pengguna daya

Untuk memberikan semua tindakan yang diperlukan untuk EMR Tanpa Server, buat dan lampirkan `AmazonEMRServerlessFullAccess` kebijakan ke pengguna, peran, atau grup IAM yang diperlukan.

Berikut ini adalah contoh kebijakan yang memungkinkan pengguna daya untuk membuat dan memodifikasi aplikasi EMR Tanpa Server, serta melakukan tindakan lain seperti mengirimkan dan men-debug pekerjaan. Ini mengungkapkan semua tindakan yang diperlukan EMR Tanpa Server untuk layanan lain.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EMRServerlessActions",
```

```

    "Effect": "Allow",
    "Action": [
      "emr-serverless:CreateApplication",
      "emr-serverless:UpdateApplication",
      "emr-serverless>DeleteApplication",
      "emr-serverless:ListApplications",
      "emr-serverless:GetApplication",
      "emr-serverless:StartApplication",
      "emr-serverless:StopApplication",
      "emr-serverless:StartJobRun",
      "emr-serverless:CancelJobRun",
      "emr-serverless:ListJobRuns",
      "emr-serverless:GetJobRun"
    ],
    "Resource": [
      "*"
    ]
  }
]
}

```

Saat Anda mengaktifkan konektivitas jaringan ke VPC, aplikasi EMR Tanpa Server membuat antarmuka jaringan elastis (ENI) Amazon EC2 untuk berkomunikasi dengan sumber daya VPC. Kebijakan berikut memastikan bahwa ENI EC2 baru hanya dibuat dalam konteks aplikasi EMR Tanpa Server.

Note

Kami sangat menyarankan untuk menetapkan kebijakan ini untuk memastikan bahwa pengguna tidak dapat membuat ENI EC2 kecuali dalam konteks peluncuran aplikasi EMR Tanpa Server.

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowEC2ENICreationWithEMRTags",

```

```

    "Effect": "Allow",
    "Action": [
      "ec2:CreateNetworkInterface"
    ],
    "Resource": [
      "arn:aws:ec2:*:*:network-interface/*"
    ],
    "Condition": {
      "StringEquals": {
        "aws:CalledViaLast": "ops.emr-serverless.amazonaws.com"
      }
    }
  }
]
}

```

Jika Anda ingin membatasi akses EMR Tanpa Server ke subnet tertentu, Anda dapat menandai setiap subnet dengan kondisi tag. Kebijakan IAM ini memastikan bahwa aplikasi EMR Tanpa Server hanya dapat membuat ENI EC2 dalam subnet yang diizinkan.

```

{
  "Sid": "AllowEC2ENICreationInSubnetAndSecurityGroupWithEMRTags",
  "Effect": "Allow",
  "Action": [
    "ec2:CreateNetworkInterface"
  ],
  "Resource": [
    "arn:aws:ec2:*:*:subnet/*",
    "arn:aws:ec2:*:*:security-group/*"
  ],
  "Condition": {
    "StringEquals": {
      "aws:ResourceTag/KEY": "VALUE"
    }
  }
}

```

Important

Jika Anda seorang Administrator atau pengguna daya yang membuat aplikasi pertama Anda, Anda harus mengonfigurasi kebijakan izin untuk memungkinkan Anda membuat peran terkait

layanan EMR Tanpa Server. Untuk mempelajari lebih lanjut, lihat [Menggunakan peran terkait layanan untuk EMR Tanpa Server](#).

Kebijakan IAM berikut memungkinkan Anda membuat peran terkait layanan EMR Tanpa Server untuk akun Anda.

```
{
  "Sid": "AllowEMRServerlessServiceLinkedRoleCreation",
  "Effect": "Allow",
  "Action": "iam:CreateServiceLinkedRole",
  "Resource": "arn:aws:iam::account-id:role/aws-service-role/ops.emr-serverless.amazonaws.com/AWSServiceRoleForAmazonEMRServerless"
}
```

Kebijakan insinyur data

Berikut ini adalah contoh kebijakan yang memungkinkan pengguna izin hanya-baca pada aplikasi EMR Tanpa Server, serta kemampuan untuk mengirimkan dan men-debug pekerjaan. Perlu diingat bahwa karena kebijakan ini tidak secara eksplisit menolak tindakan, pernyataan kebijakan yang berbeda masih dapat digunakan untuk memberikan akses ke tindakan tertentu.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EMRServerlessActions",
      "Effect": "Allow",
      "Action": [
        "emr-serverless:ListApplications",
        "emr-serverless:GetApplication",
        "emr-serverless:StartApplication",
        "emr-serverless:StartJobRun",
        "emr-serverless:CancelJobRun",
        "emr-serverless:ListJobRuns",
        "emr-serverless:GetJobRun"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

```

    ]
  }
]
}

```

Menggunakan tag untuk kontrol akses

Anda dapat menggunakan kondisi tag untuk kontrol akses berbutir halus. Misalnya, Anda dapat membatasi pengguna dari satu tim sehingga mereka hanya dapat mengirimkan pekerjaan ke aplikasi EMR Tanpa Server yang ditandai dengan nama tim mereka.

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EMRServerlessActions",
      "Effect": "Allow",
      "Action": [
        "emr-serverless:ListApplications",
        "emr-serverless:GetApplication",
        "emr-serverless:StartApplication",
        "emr-serverless:StartJobRun",
        "emr-serverless:CancelJobRun",
        "emr-serverless:ListJobRuns",
        "emr-serverless:GetJobRun"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}

```

Kebijakan untuk kendali akses berbasis tanda

Anda dapat menggunakan kondisi dalam kebijakan berbasis identitas untuk mengontrol akses ke aplikasi dan pekerjaan berjalan berdasarkan tag.

Contoh berikut menunjukkan skenario dan cara yang berbeda untuk menggunakan operator kondisi dengan kunci kondisi EMR Tanpa Server. 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 lebih lanjut 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 ini menunjukkan kebijakan izin berbasis identitas yang digunakan untuk mengontrol tindakan yang diizinkan dengan aplikasi EMR Tanpa Server.

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 tag belum ditambahkan ke aplikasi, atau tidak berisi `nilaidev`, kebijakan tidak berlaku, dan tindakan tidak diizinkan oleh kebijakan ini. Jika tidak ada pernyataan kebijakan lain yang mengizinkan tindakan, pengguna hanya dapat bekerja dengan aplikasi yang memiliki tag ini dengan nilai ini.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-serverless:GetApplication"
      ],
      "Resource": [
```

```

        "*"
    ],
    "Condition": {
        "StringEquals": {
            "aws:ResourceTag/department": "dev"
        }
    },
    "Sid": "AllowEMRSERVERLESSGetapplication"
}
]
}

```

Anda juga dapat menentukan beberapa nilai tanda menggunakan operator syarat. Misalnya, untuk mengizinkan tindakan pada aplikasi di mana department tag berisi nilai dev atau test, ganti blok kondisi pada contoh sebelumnya dengan yang berikut ini.

```

"Condition": {
    "StringEquals": {
        "aws:ResourceTag/department": ["dev", "test"]
    }
}

```

Memerlukan penandaan ketika sumber daya dibuat

Pada contoh di bawah ini, tag perlu diterapkan saat membuat aplikasi.

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-serverless:CreateApplication"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringEquals": {

```

```

        "aws:RequestedRegion": "us-east-1"
    }
},
    "Sid": "AllowEMRSERVERLESSCreateapplication"
}
]
}

```

Pernyataan kebijakan berikut memungkinkan pengguna untuk membuat aplikasi hanya jika aplikasi memiliki department tag, yang dapat berisi nilai apa pun.

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-serverless:CreateApplication"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": ["us-east-1", "us-west-2"]
        }
      },
      "Sid": "AllowEMRSERVERLESSCreateapplication"
    }
  ]
}

```

Tolak akses untuk menambah dan menghapus tanda

Kebijakan ini mencegah pengguna menambahkan atau menghapus tag pada aplikasi EMR Tanpa Server dengan department tag yang nilainya tidak. dev

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "emr-serverless:TagResource",
        "emr-serverless:UntagResource"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringNotEquals": {
          "aws:PrincipalTag/department": "dev"
        }
      },
      "Sid": "AllowEMRSERVERLESSTagresource"
    }
  ]
}
```

Identity-based contoh kebijakan untuk EMR Tanpa Server

Secara default, pengguna dan peran tidak memiliki izin untuk membuat atau memodifikasi sumber daya Amazon EMR Tanpa Server. 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 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 Tanpa Server, termasuk format ARN untuk setiap jenis sumber daya, lihat Kunci [tindakan, sumber daya, dan kondisi untuk Amazon EMR](#) Tanpa Server di Referensi Otorisasi Layanan.

Topik

- [Praktik terbaik kebijakan](#)
- [Memungkinkan pengguna untuk mengakses izin mereka sendiri](#)

Praktik terbaik kebijakan

Note

EMR Tanpa Server tidak mendukung kebijakan terkelola, jadi praktik pertama yang tercantum di bawah ini tidak berlaku.

Identity-based kebijakan menentukan apakah seseorang dapat membuat, mengakses, atau menghapus sumber daya Amazon EMR Tanpa Server di akun Anda. Tindakan ini membuat Akun AWS Anda dikenai biaya. Ketika Anda membuat atau mengedit kebijakan berbasis identitas, ikuti panduan dan rekomendasi ini:

- Mulailah dengan kebijakan AWS terkelola dan beralih ke izin hak istimewa paling sedikit — Untuk mulai memberikan izin kepada pengguna dan beban kerja Anda, gunakan kebijakan AWS terkelola yang memberikan izin untuk banyak kasus penggunaan umum. Mereka tersedia di Anda Akun AWS. Kami menyarankan Anda mengurangi izin lebih lanjut dengan menentukan kebijakan yang dikelola AWS pelanggan yang khusus untuk kasus penggunaan Anda. Untuk informasi selengkapnya, lihat [Kebijakan yang dikelola AWS](#) atau [Kebijakan yang dikelola AWS untuk fungsi tugas](#) dalam Panduan Pengguna IAM.
- Menerapkan izin dengan hak akses paling rendah – Ketika Anda menetapkan izin dengan kebijakan IAM, hanya berikan izin yang diperlukan untuk melakukan tugas. Anda melakukannya dengan mendefinisikan tindakan yang dapat diambil pada sumber daya tertentu dalam kondisi tertentu, yang juga dikenal sebagai izin dengan hak akses paling rendah. Untuk informasi selengkapnya tentang cara menggunakan IAM untuk mengajukan izin, lihat [Kebijakan dan izin dalam IAM](#) dalam Panduan Pengguna IAM.
- Gunakan kondisi dalam kebijakan IAM untuk membatasi akses lebih lanjut – Anda dapat menambahkan suatu kondisi ke kebijakan Anda untuk membatasi akses ke tindakan dan sumber daya. Sebagai contoh, Anda dapat menulis kondisi kebijakan untuk menentukan bahwa semua permintaan harus dikirim menggunakan SSL. Anda juga dapat menggunakan ketentuan untuk memberikan akses ke tindakan layanan jika digunakan melalui yang spesifik Layanan AWS, seperti CloudFormation. Untuk informasi selengkapnya, lihat [Elemen kebijakan JSON IAM: Kondisi](#) dalam Panduan Pengguna IAM.
- Gunakan IAM Access Analyzer untuk memvalidasi kebijakan IAM Anda untuk memastikan izin yang aman dan fungsional – IAM Access Analyzer memvalidasi kebijakan baru dan yang sudah ada sehingga kebijakan tersebut mematuhi bahasa kebijakan IAM (JSON) dan praktik terbaik IAM. IAM Access Analyzer menyediakan lebih dari 100 pemeriksaan kebijakan dan rekomendasi yang

dapat ditindaklanjuti untuk membantu Anda membuat kebijakan yang aman dan fungsional. Untuk informasi selengkapnya, lihat [Validasi kebijakan dengan IAM Access Analyzer](#) dalam Panduan Pengguna IAM.

- Memerlukan otentikasi multi-faktor (MFA) - Jika Anda memiliki skenario yang mengharuskan pengguna IAM atau pengguna root di Anda, Akun AWS aktifkan MFA untuk keamanan tambahan. Untuk meminta MFA ketika operasi API dipanggil, tambahkan kondisi MFA pada kebijakan Anda. Untuk informasi selengkapnya, lihat [Amankan akses API dengan MFA](#) dalam Panduan Pengguna IAM.

Untuk informasi selengkapnya tentang praktik terbaik dalam IAM, lihat [Praktik terbaik keamanan di IAM](#) dalam Panduan Pengguna IAM.

Memungkinkan pengguna untuk mengakses 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:ListGroupsWithUser",
        "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": "*"
}
]
}

```

Amazon EMR Pembaruan tanpa server ke kebijakan terkelola AWS

Akses detail tentang pembaruan kebijakan AWS terkelola untuk Amazon EMR Tanpa Server sejak layanan ini mulai melacak perubahan ini. [Untuk peringatan otomatis tentang perubahan pada halaman ini, berlangganan umpan RSS di halaman riwayat Dokumen Tanpa Server Amazon EMR.](#)

Ubah	Deskripsi	Date
AmazonEMRServerlessServiceRolePolicy — Perbarui ke kebijakan yang ada	Amazon EMR Tanpa Server menambahkan yang baru SidCloudWatchPolicyStatement dan EC2PolicyStatement ke kebijakan . AmazonEMRServerlessServiceRolePolicy	Januari 25, 2024
AmazonEMRServerlessServiceRolePolicy — Perbarui ke kebijakan yang ada	Amazon EMR Tanpa Server menambahkan izin baru untuk memungkinkan Amazon EMR Tanpa Server mempublikasikan metrik akun agregat untuk penggunaan vCPU di namespace. "AWS/Usage"	20 April 2023

Ubah	Deskripsi	Date
Amazon EMR Serverless mulai melacak perubahan	Amazon EMR Serverless mulai melacak perubahan untuk kebijakan yang dikelola. AWS	20 April 2023

Memecahkan masalah Amazon EMR Identitas dan akses tanpa server

Gunakan informasi berikut untuk membantu Anda mendiagnosis dan memperbaiki masalah umum yang mungkin Anda temui saat bekerja dengan Amazon EMR Tanpa Server dan IAM.

Topik

- [Saya tidak berwenang untuk melakukan tindakan di Amazon EMR Tanpa Server](#)
- [Saya tidak berwenang untuk melakukan iam: PassRole](#)
- [Saya ingin mengizinkan orang di luar AWS akun saya untuk mengakses sumber daya Amazon EMR Tanpa Server saya](#)
- [Saya tidak dapat membuka server UI/Spark riwayat Langsung dari EMR Studio untuk men-debug pekerjaan saya atau kesalahan API terjadi ketika saya mencoba mendapatkan log menggunakan `get-dashboard-for-job-run`](#)

Saya tidak berwenang untuk melakukan tindakan di Amazon EMR Tanpa Server

Jika Konsol Manajemen AWS memberitahu Anda bahwa Anda tidak berwenang untuk melakukan tindakan, maka hubungi administrator Anda untuk bantuan. Administrator Anda adalah orang yang memberikan nama pengguna dan kata sandi Anda.

Contoh kesalahan berikut terjadi ketika `mateojackson` pengguna mencoba menggunakan konsol untuk mengakses detail tentang `my-example-widget` sumber daya fiksi tetapi tidak memiliki izin `emr-serverless:GetWidget` fiksi.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform: emr-serverless: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-serverless:GetWidget`.

Saya tidak berwenang untuk melakukan iam: PassRole

Jika Anda menerima kesalahan yang tidak diizinkan untuk melakukan `iam:PassRole` tindakan, kebijakan Anda harus diperbarui agar Anda dapat meneruskan peran ke Amazon EMR Tanpa Server.

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 Tanpa Server. 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 sumber daya Amazon EMR Tanpa Server saya

Anda dapat membuat peran yang dapat digunakan pengguna di akun lain atau orang-orang di luar organisasi Anda untuk mengakses sumber daya Anda. Anda dapat menentukan siapa saja yang dipercaya untuk mengambil peran tersebut. Untuk layanan yang mendukung kebijakan berbasis sumber daya atau daftar kontrol akses (ACL), Anda dapat menggunakan kebijakan tersebut untuk memberi orang akses ke sumber daya Anda.

Untuk mempelajari selengkapnya, periksa referensi berikut:

- Untuk mengetahui apakah Amazon EMR Serverless mendukung fitur-fitur ini, lihat [Identity and Access Management \(IAM\) di Amazon EMR Tanpa Server](#)

- 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](#).

Saya tidak dapat membuka server UI/Spark riwayat Langsung dari EMR Studio untuk men-debug pekerjaan saya atau kesalahan API terjadi ketika saya mencoba mendapatkan log menggunakan **get-dashboard-for-job-run**

Jika Anda menggunakan penyimpanan terkelola EMR Tanpa Server untuk pencatatan dan aplikasi EMR Tanpa Server Anda berada di subnet pribadi dengan titik akhir VPC untuk Amazon S3 dan Anda melampirkan kebijakan titik akhir untuk mengontrol akses, tambahkan izin yang disebutkan dalam [Logging for EMR Tanpa Server dengan penyimpanan terkelola dalam kebijakan VPC Anda ke titik akhir gateway S3 untuk EMR Tanpa Server untuk menyimpan dan melayani log](#) aplikasi.

Propagasi Identitas Tepercaya

Dengan Amazon EMR rilis 7.8.0 dan yang lebih tinggi, Anda dapat menyebarkan identitas pengguna dari AWS IAM Identity Center ke beban kerja interaktif dengan EMR Serverless melalui Apache Livy Endpoint. Beban kerja interaktif Apache Livy selanjutnya akan menyebarkan identitas yang disediakan ke layanan hilir seperti Amazon S3, Lake Formation, dan Amazon Redshift, memungkinkan akses data yang aman melalui identitas pengguna di hilir ini. Bagian berikut memberikan gambaran konseptual, prasyarat, dan langkah-langkah yang diperlukan untuk meluncurkan dan menyebarkan identitas ke beban kerja interaktif dengan EMR Tanpa Server melalui Apache Livy Endpoint.

Ikhtisar

[IAM Identity Center](#) adalah pendekatan yang direkomendasikan untuk otentikasi dan otorisasi tenaga kerja AWS untuk organisasi dari berbagai ukuran dan jenis. Dengan Identity Center, buat dan kelola

identitas pengguna di AWS, atau sambungkan sumber identitas yang ada, termasuk Microsoft Active Directory, Okta, Ping Identity, Google Workspace JumpCloud, dan Microsoft Entra ID (sebelumnya Azure AD).

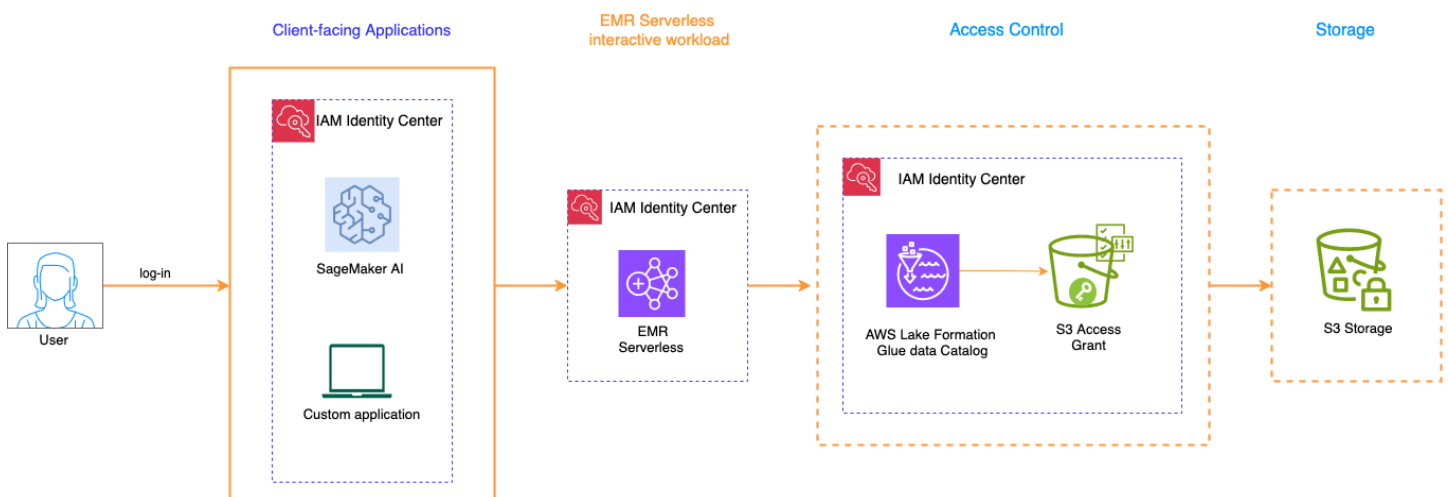
[Propagasi identitas terpercaya](#) adalah fitur Pusat AWS Identitas IAM yang dapat digunakan oleh administrator AWS layanan terhubung untuk memberikan dan mengaudit akses ke data layanan. Akses ke data ini didasarkan pada atribut pengguna seperti asosiasi grup. Menyiapkan propagasi identitas terpercaya memerlukan kolaborasi antara administrator AWS layanan yang terhubung dan administrator Pusat Identitas IAM. Untuk informasi lebih lanjut, lihat [Prasyarat dan pertimbangan](#) dalam Panduan Pengguna Pusat Identitas IAM.

Fitur dan manfaat

[Integrasi EMR Serverless Apache Livy Endpoint dengan IAM Identity Center Penyebaran identitas terpercaya memberikan manfaat berikut:](#)

- Kemampuan untuk menegakkan otorisasi tingkat tabel dengan identitas Pusat Identitas pada tabel katalog data AWS Glue yang dikelola AWS Lake Formation.
- Kemampuan untuk menegakkan otorisasi dengan identitas Pusat Identitas di kluster Amazon Redshift.
- Memungkinkan pelacakan ujung ke ujung tindakan pengguna untuk audit.
- Kemampuan untuk menerapkan otorisasi tingkat awalan Amazon S3 dengan identitas Pusat Identitas pada awalan S3 yang dikelola S3 S3 Access Grants.

Cara kerjanya



Contoh kasus penggunaan

Persiapan Data dan Rekayasa Fitur

Ilmuwan data dari beberapa tim peneliti berkolaborasi dalam proyek kompleks menggunakan platform data terpadu. Mereka masuk ke SageMaker AI menggunakan kredensi perusahaan mereka, segera mendapatkan akses ke danau data bersama yang luas yang mencakup beberapa akun. AWS Saat mereka memulai rekayasa fitur untuk model pembelajaran mesin baru, sesi Spark diluncurkan melalui EMR Tanpa Server menegakkan kolom Lake Formation dan kebijakan keamanan tingkat baris berdasarkan identitas mereka yang disebar. Para ilmuwan dapat secara efisien menyiapkan data dan fitur insinyur menggunakan alat yang sudah dikenal, sementara tim kepatuhan yakin bahwa setiap interaksi data dilacak dan diaudit secara otomatis. Lingkungan yang aman dan kolaboratif ini mempercepat jalur penelitian sambil mempertahankan standar perlindungan data yang ketat yang diperlukan dalam industri yang diatur.

Memulai dengan Propagasi Identitas Terpercaya

[Bagian ini membantu Anda mengonfigurasi aplikasi EMR-Serverless dengan Apache Livy Endpoint untuk berintegrasi dengan AWS IAM Identity Center dan mengaktifkan propagasi identitas Terpercaya.](#)

Prasyarat

- Instance Pusat Identitas di AWS Wilayah tempat Anda ingin membuat propagasi identitas Terpercaya mengaktifkan EMR Serverless Apache Livy Endpoint. Instance Pusat Identitas hanya dapat ada di satu Wilayah untuk sebuah AWS akun. lihat [Aktifkan Pusat Identitas IAM](#) dan [Berikan pengguna dan grup dari sumber identitas Anda ke Pusat Identitas IAM](#).
- Aktifkan propagasi identitas Terpercaya untuk layanan hilir seperti Lake Formation atau S3 Access Grants atau kluster Amazon Redshift yang berinteraksi dengan beban kerja interaktif untuk mengakses data.

Izin untuk membuat propagasi identitas terpercaya mengaktifkan EMR Aplikasi Tanpa Server

Selain [izin dasar yang diperlukan untuk mengakses EMR Tanpa Server, Anda harus mengonfigurasi izin tambahan untuk identitas atau peran IAM Anda yang digunakan](#) untuk membuat propagasi identitas terpercaya yang diaktifkan EMR Aplikasi Tanpa Server. Untuk propagasi identitas

terpercaya, EMR creates/bootstraps Serverless satu layanan mengelola Aplikasi Pusat Identitas di akun Anda yang dimanfaatkan layanan untuk validasi identitas dan propagasi identitas ke hilir.

```
"sso:DescribeInstance",  
"sso:CreateApplication",  
"sso>DeleteApplication",  
"sso:PutApplicationAuthenticationMethod",  
"sso:PutApplicationAssignmentConfiguration",  
"sso:PutApplicationGrant",  
"sso:PutApplicationAccessScope"
```

- `sso:DescribeInstance`— Memberikan izin untuk mendeskripsikan dan memvalidasi InstanCearn Pusat Identitas IAM yang Anda tentukan dalam parameter. `identity-center-configuration`
- `sso:CreateApplication`— Memberikan izin untuk membuat Aplikasi Pusat Identitas IAM yang dikelola EMR Tanpa Server yang digunakan untuk tindakan. `trusted-identity-propatgion`
- `sso>DeleteApplication`— memberikan izin untuk membersihkan Aplikasi Pusat Identitas IAM yang dikelola EMR Tanpa Server
- `sso:PutApplicationAuthenticationMethod`— Memberikan izin untuk menempatkan `AuthenticationMethod` pada Aplikasi Pusat Identitas IAM yang dikelola EMR Tanpa Server yang memungkinkan prinsipal layanan `emr-serverless` untuk berinteraksi dengan Aplikasi Pusat Identitas IAM.
- `sso:PutApplicationAssignmentConfiguration`— Memberikan izin untuk mengatur pengaturan “`User-assignment-not-required`” pada Aplikasi Pusat Identitas IAM.
- `sso:PutApplicationGrant`— Memberikan izin untuk menerapkan hibah `token-exchange`, `IntroSpectToken`, `RefreshToken`, dan `RevokeToken` pada Aplikasi Pusat Identitas IAM.
- `sso:PutApplicationAccessScope`— Memberikan izin untuk menerapkan propagasi identitas terpercaya yang diaktifkan cakupan hilir ke Aplikasi Pusat Identitas IAM. Kami menerapkan cakupan “`redshift:connect`”, “`lakeformation:query`” dan “`s3:read_write`” untuk mengaktifkan layanan ini. `trusted-identity-propagation`

Buat propagasi identitas terpercaya yang diaktifkan EMR Aplikasi Tanpa Server

Anda harus menentukan `-identity-center-configuration` bidang dengan `identityCenterInstanceArn` untuk mengaktifkan propagasi identitas terpercaya dalam aplikasi.

Gunakan perintah contoh berikut untuk membuat Aplikasi EMR Tanpa Server yang mengaktifkan propagasi identitas terpercaya.

Note

Anda juga harus menentukan `--interactive-configuration` `'{"livyEndpointEnabled":true}'` bahwa propagasi identitas terpercaya diaktifkan hanya untuk Apache Livy Endpoint.

```
aws emr-serverless create-application \
  --release-label emr-7.8.0 \
  --type "SPARK" \
  --identity-center-configuration '{"identityCenterInstanceArn" :
"arn:aws:sso:::instance/ssoins-123456789"}' \
  --interactive-configuration '{"livyEndpointEnabled":true}'
```

- `identity-center-configuration`— (opsional) Mengaktifkan propagasi identitas terpercaya Pusat Identitas jika ditentukan.
- `identityCenterInstanceArn`— (wajib) Pusat Identitas misalnya ARN.

Jika Anda tidak memiliki izin Pusat Identitas yang diperlukan (disebutkan sebelumnya), pertama-tama buat Aplikasi EMR Tanpa Server tanpa propagasi identitas terpercaya (misalnya, jangan `--identity-center-configuration` tentukan parameter) dan kemudian minta Admin Pusat Identitas Anda untuk mengaktifkan propagasi identitas terpercaya dengan menjalankan API `update-application`, lihat contoh di bawah ini:

```
aws emr-serverless update-application \
  --application-id applicationId \
  --identity-center-configuration '{"identityCenterInstanceArn" :
"arn:aws:sso:::instance/ssoins-123456789"}'
```

EMR Tanpa Server membuat Aplikasi Pusat Identitas terkelola layanan di akun Anda yang memanfaatkan layanan untuk validasi identitas dan propagasi identitas ke layanan hilir. EMR Aplikasi Pusat Identitas terkelola yang dibuat tanpa server dibagikan di semua aplikasi `trusted-identity-propagation` EMR Tanpa Server yang diaktifkan di akun Anda.

Note

Jangan mengubah pengaturan secara manual pada Aplikasi Pusat Identitas yang dikelola. Perubahan apa pun dapat memengaruhi semua aplikasi EMR Tanpa Server yang trusted-identity-propagation diaktifkan di akun Anda.

Izin Peran Eksekusi Job untuk menyebarkan identitas

Karena EMR-Serverless memanfaatkan job-execution-role kredensi yang ditingkatkan Identitas untuk menyebarkan identitas ke layanan hilir AWS , kebijakan kepercayaan Job Execution Role harus memiliki izin tambahan untuk meningkatkan kredensi peran eksekusi pekerjaan dengan identitas `sts:SetContext` untuk memungkinkan layanan hilir trusted-identity-propagation, seperti hibah akses S3, Lake Formation, atau Amazon Redshift. Untuk mempelajari lebih lanjut tentang cara membuat peran, lihat [Membuat peran runtime pekerjaan](#).

JSON

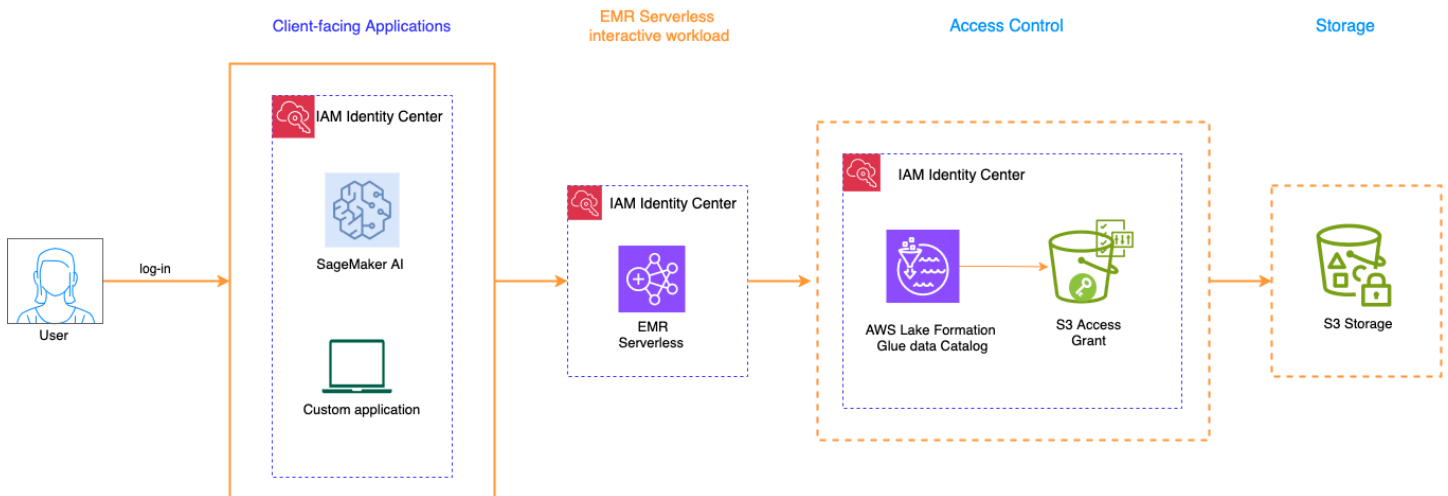
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "emr-serverless.amazonaws.com"
      },
      "Action": [ "sts:AssumeRole", "sts:SetContext" ]
    }
  ]
}
```

Selain itu, JobExecutionRole memerlukan izin untuk AWS layanan hilir yang akan dijalankan oleh pekerjaan untuk mengambil data menggunakan identitas pengguna. Silakan lihat tautan di bawah ini untuk mengonfigurasi S3 Access Grant, Lake Formation.

- [Menggunakan Lake Formation dengan EMR Serverless](#)
- [Menggunakan Hibah Akses Amazon S3 dengan EMR Tanpa Server](#)

Propagasi Identitas Tepercaya untuk beban kerja interaktif

Langkah-langkah untuk menyebarkan identitas ke beban kerja interaktif melalui titik akhir Apache Livy bergantung pada apakah pengguna Anda berinteraksi dengan lingkungan pengembangan AWS terkelola seperti Amazon SageMaker AI atau lingkungan Notebook yang dihosting sendiri sebagai aplikasi yang menghadap klien.



AWS lingkungan pengembangan terkelola

Aplikasi yang dihadapi klien AWS terkelola berikut ini mendukung propagasi identitas tepercaya dengan titik akhir Apache Livy tanpa server EMR:

- [Amazon SageMaker AI](#)

Lingkungan Notebook yang dihosting sendiri yang dikelola pelanggan

Untuk mengaktifkan propagasi identitas tepercaya bagi pengguna aplikasi yang dikembangkan khusus, lihat [Akses AWS layanan secara terprogram menggunakan propagasi identitas tepercaya](#) di Blog Keamanan.AWS

Sesi latar belakang pengguna

Sesi latar belakang pengguna memungkinkan analitik yang berjalan lama dan alur pembelajaran mesin untuk melanjutkan bahkan setelah pengguna keluar dari antarmuka notebook mereka. Kemampuan ini diimplementasikan melalui integrasi EMR Tanpa Server dengan fitur propagasi identitas tepercaya IAM Identity Center. Bagian ini menjelaskan opsi konfigurasi dan perilaku untuk sesi latar belakang pengguna.

Note

Sesi latar belakang pengguna berlaku untuk beban kerja Spark yang dimulai melalui antarmuka notebook seperti Amazon SageMaker Unified Studio. Mengaktifkan atau menonaktifkan fitur ini hanya memengaruhi sesi Livy baru; sesi Livy aktif yang ada tidak terpengaruh.

Konfigurasi sesi latar belakang pengguna

Sesi latar belakang pengguna harus diaktifkan pada dua tingkat untuk fungsionalitas yang tepat:

1. Tingkat instans Pusat Identitas IAM - biasanya dikonfigurasi oleh administrator IDC
2. Tingkat aplikasi EMR Tanpa Server - dikonfigurasi oleh administrator aplikasi EMR Tanpa Server

Aktifkan sesi latar belakang pengguna untuk aplikasi EMR Tanpa Server

Untuk mengaktifkan sesi latar belakang pengguna untuk aplikasi EMR Tanpa Server, Anda harus mengatur `userBackgroundSessionsEnabled` parameter ke `true` dalam `identityCenterConfiguration` saat membuat atau memperbarui aplikasi.

Prasyarat

- Peran IAM Anda yang digunakan untuk create/update aplikasi EMR Tanpa Server harus memiliki izin. `sso:PutApplicationSessionConfiguration` Izin ini memungkinkan EMR Tanpa Server untuk mengaktifkan sesi latar belakang Pengguna di tingkat aplikasi IDC terkelola EMR Tanpa Server.
- Aplikasi EMR Tanpa Server Anda harus menggunakan label rilis 7.8 atau yang lebih baru dan harus diaktifkan Trusted-Identity Propagation.

Untuk mengaktifkan sesi latar belakang pengguna menggunakan AWS CLI

```
aws emr-serverless create-application \  
  --name "my-analytics-app" \  
  --type "SPARK" \  
  --release-label "emr-7.8.0" \  
  --user-background-sessions-enabled true
```

```
--identity-center-configuration '{"identityCenterInstanceArn":
"arn:aws:sso:::instance/ssoins-1234567890abcdef", "userBackgroundSessionsEnabled":
true}'
```

Untuk memperbarui aplikasi yang ada:

```
aws emr-serverless update-application \
  --application-id applicationId \
  --identity-center-configuration '{"identityCenterInstanceArn":
"arn:aws:sso:::instance/ssoins-1234567890abcdef", "userBackgroundSessionsEnabled":
true}'
```

Matriks Konfigurasi

Konfigurasi sesi latar belakang pengguna yang efektif bergantung pada pengaturan aplikasi EMR Tanpa Server dan pengaturan tingkat instans Pusat Identitas IAM:

Matriks Konfigurasi Sesi Latar Belakang Pengguna

Pusat Identitas IAM Diaktifkan userBackgroundSession	EMR Tanpa Server Diaktifkan userBackgroundSessions	Perilaku
Ya	BETUL	Sesi latar belakang pengguna diaktifkan
Ya	SALAH	Sesi berakhir dengan logout pengguna
Tidak	BETUL	Aplikasi creation/update gagal dengan Pengecualian
Tidak	SALAH	Sesi berakhir dengan logout pengguna

Durasi sesi latar belakang pengguna default

Secara default, semua sesi latar belakang pengguna memiliki batas durasi 7 hari di IAM Identity Center. Administrator dapat mengubah durasi ini di konsol Pusat Identitas IAM. Pengaturan ini berlaku pada tingkat instans Pusat Identitas IAM, yang memengaruhi semua aplikasi Pusat Identitas IAM yang didukung dalam instance tersebut.

- Durasi dapat diatur ke nilai apa pun dari 15 menit hingga 90 hari.
- Pengaturan ini dikonfigurasi di konsol Pusat Identitas IAM di bawah Pengaturan → Otentikasi → Konfigurasi (bagian Pekerjaan Non-Interaktif)

Note

Sesi EMR Serverless Livy memiliki batas durasi maksimum terpisah 24 jam. Sesi akan berakhir ketika batas sesi Livy atau durasi sesi latar belakang pengguna tercapai, mana yang lebih dulu.

Dampak menonaktifkan sesi latar belakang pengguna

Saat sesi latar belakang pengguna dinonaktifkan di Pusat Identitas IAM:

Sesi Livy yang ada

Terus berjalan tanpa gangguan jika dimulai dengan sesi latar belakang pengguna diaktifkan. Sesi ini akan terus menggunakan token sesi latar belakang yang ada sampai mereka berakhir secara alami atau secara eksplisit dihentikan.

Sesi Livy baru

Akan menggunakan alur propagasi identitas tepercaya standar dan akan berakhir ketika pengguna log out atau sesi interaktif mereka kedaluwarsa (seperti saat menutup notebook Amazon SageMaker Unified Studio). JupyterLab

Mengubah durasi sesi latar belakang pengguna

Saat pengaturan durasi untuk sesi latar belakang pengguna diubah di Pusat Identitas IAM:

Sesi Livy yang ada

Lanjutkan untuk menjalankan dengan durasi sesi latar belakang yang sama dengan yang mereka mulai.

Sesi Livy baru

Akan menggunakan durasi sesi baru untuk sesi latar belakang.

Pertimbangan-pertimbangan

Ketentuan Pengakhiran Sesi

Saat menggunakan sesi latar belakang pengguna, sesi Livy akan terus berjalan hingga salah satu hal berikut terjadi:

- Sesi latar belakang pengguna kedaluwarsa (berdasarkan konfigurasi IDC, hingga 90 hari)
- Sesi latar belakang pengguna dicabut secara manual oleh administrator
- Sesi Livy mencapai batas waktu idle (default: 1 jam setelah pernyataan terakhir yang dieksekusi)
- Sesi Livy mencapai durasi maksimumnya (24 jam)
- Pengguna secara eksplisit menghentikan atau memulai ulang kernel notebook

Persistensi Data

Saat menggunakan sesi latar belakang pengguna:

- Pengguna tidak dapat menyambung kembali ke antarmuka notebook mereka untuk melihat hasil setelah mereka keluar
- Konfigurasi pernyataan Spark Anda untuk menulis hasil ke penyimpanan persisten (seperti Amazon S3) sebelum eksekusi selesai

Implikasi Biaya

- Pekerjaan akan terus berjalan hingga selesai bahkan setelah pengguna mengakhiri JupyterLab sesi Amazon SageMaker Unified Studio mereka dan akan dikenakan biaya untuk seluruh durasi proses yang diselesaikan.
- Pantau sesi latar belakang aktif Anda untuk menghindari biaya yang tidak perlu dari sesi yang terlupakan atau ditinggalkan.

Ketersediaan Fitur

Sesi latar belakang pengguna untuk EMR Tanpa Server tersedia untuk:

- Hanya mesin percikan (Mesin sarang tidak didukung)
- Sesi interaktif Livy saja (pekerjaan batch dan pekerjaan streaming tidak didukung)
- Label rilis EMR Tanpa Server 7.8 dan yang lebih baru

Pertimbangan untuk integrasi EMR Tanpa Server Trusted-Identity-Propagation

Pertimbangkan hal berikut ketika Anda menggunakan IAM Identity Center Trusted-Identity-Propagation dengan Aplikasi EMR Serverless:

- Propagasi Identitas Tepercaya melalui Identity Center didukung di Amazon EMR 7.8.0 dan lebih tinggi, dan hanya dengan Apache Spark.
- Propagasi Identitas Tepercaya hanya dapat digunakan untuk [beban kerja interaktif dengan EMR Tanpa Server](#) melalui titik akhir Apache Livy. Beban kerja interaktif melalui EMR Studio tidak mendukung Propagasi Identitas Tepercaya
- Pekerjaan batch dan beban kerja streaming tidak mendukung propagasi identitas Tepercaya
- Kontrol akses berbutir halus menggunakan AWS Lake Formation yang menggunakan Trusted Identity Propagation tersedia untuk [beban kerja interaktif dengan EMR](#) Tanpa Server melalui titik akhir Apache Livy.
- Propagasi Identitas Tepercaya dengan Amazon EMR didukung di Wilayah berikut AWS :
 - af-south-1 — Afrika (Cape Town)
 - ap-east-1 - Asia Pasifik (Hong Kong)
 - ap-northeast-1 – Asia Pacific (Tokyo)
 - ap-northeast-2 - Asia Pasifik (Seoul)
 - ap-northeast-3 - Asia Pasifik (Osaka)
 - ap-south-1 — Asia Pasifik (Mumbai)
 - ap-southeast-1 – Asia Pacific (Singapore)
 - ap-southeast-2 – Asia Pacific (Sydney)
 - ap-southeast-3 - Asia Pasifik (Jakarta)
 - ca-central-1 — Kanada (Tengah)
 - ca-west-1 — Kanada (Calgary)
 - eu-central-1 – Europe (Frankfurt)
 - eu-north-1 - Eropa (Stockholm)
 - eu-south-1 — Eropa (Milan)
 - eu-south-2 — Eropa (Spanyol)
 - eu-west-1 – Europe (Ireland)

- eu-west-2 - Eropa (London)
- eu-west-3 - Eropa (Paris)
- me-central-1 - Timur Tengah (UEA)
- me-south-1 - Timur Tengah (Bahrain)
- sa-east-1 — Amerika Selatan (Sao Paulo)
- us-east-1 – US East (N. Virginia)
- us-east-2 – US East (Ohio)
- us-west-1 - AS Barat (California Utara)
- us-west-2 – US West (Oregon)

Menggunakan Lake Formation dengan EMR Serverless

Anda dapat mengonfigurasi aplikasi EMR Tanpa Server untuk menggunakan Lake Formation dengan akses tabel penuh atau kontrol akses berbutir halus. Untuk detail tentang fitur yang didukung di setiap mode akses, tinjau tabel berikut.

Ketersediaan fitur

Fitur	Tersedia dari
Baca operasi (PILIH, DESKRIPSIKAN) untuk tabel Hive, Iceberg	EMR 7,2+
Tampilan multi-dialek	EMR 7,6+
Baca operasi (PILIH, DESKRIPSIKAN) untuk tabel Delta Lake dan Hudi	EMR 7,6+
Akses meja penuh untuk Hive, Iceberg	EMR 7,9+
Akses meja penuh untuk Danau Delta	EMR 7.11+
Menulis operasi (DDL, DHTML) untuk tabel Hive, Iceberg dan Delta Lake	EMR 7,12+
Akses meja penuh untuk Hudi	EMR 7,12+

Akses meja lengkap Lake Formation untuk EMR Tanpa Server

Dengan Amazon EMR merilis 7.8.0 dan yang lebih tinggi, Anda dapat memanfaatkan Lake AWS Formation with Glue Data Catalog di mana peran runtime pekerjaan memiliki izin tabel lengkap tanpa batasan kontrol akses berbutir halus. Kemampuan ini memungkinkan Anda membaca dan menulis ke tabel yang dilindungi oleh Lake Formation dari batch EMR Serverless Spark dan pekerjaan interaktif Anda. Lihat bagian berikut untuk mempelajari lebih lanjut tentang Lake Formation dan cara menggunakannya dengan EMR Tanpa Server.

Menggunakan Lake Formation dengan akses meja penuh

Anda dapat mengakses tabel katalog Glue Data yang dilindungi AWS Lake Formation dari pekerjaan EMR Serverless Spark atau sesi interaktif di mana peran runtime pekerjaan memiliki akses tabel penuh. Anda tidak perlu mengaktifkan AWS Lake Formation pada aplikasi EMR Serverless. Ketika pekerjaan Spark dikonfigurasi untuk Akses Tabel Penuh (FTA), kredensial AWS Lake Formation digunakan untuk data read/write S3 untuk tabel terdaftar AWS Lake Formation, sedangkan kredensial peran runtime pekerjaan akan digunakan untuk tabel yang tidak terdaftar di Lake Formation. read/write AWS

Important

Jangan aktifkan AWS Lake Formation untuk kontrol akses berbutir halus. Pekerjaan tidak dapat secara bersamaan menjalankan Full Table Access (FTA) dan Fine-Grained Access Control (FGAC) pada cluster EMR atau aplikasi yang sama.

Langkah 1: Aktifkan Akses Tabel Penuh di Lake Formation

Untuk menggunakan mode Akses Tabel Penuh (FTA), Anda harus mengizinkan mesin kueri pihak ketiga mengakses data tanpa validasi tag sesi IAM di Lake Formation AWS . Untuk mengaktifkan, ikuti langkah-langkah dalam [Integrasi aplikasi untuk akses tabel penuh](#).

Note

Saat mengakses tabel lintas akun, akses tabel penuh harus diaktifkan di akun produsen dan konsumen. Dengan cara yang sama, saat mengakses tabel lintas wilayah, pengaturan ini harus diaktifkan di wilayah produsen dan konsumen.

Langkah 2: Siapkan izin IAM untuk peran runtime pekerjaan

Untuk akses baca atau tulis ke data dasar, selain izin Lake Formation, peran runtime pekerjaan memerlukan izin `lakeformation:GetDataAccess` IAM. Dengan izin ini, Lake Formation memberikan permintaan kredensi sementara untuk mengakses data.

Berikut ini adalah contoh kebijakan tentang cara memberikan izin IAM untuk mengakses skrip di Amazon S3, mengunggah log ke S3, izin AWS Glue API, dan izin untuk mengakses Lake Formation.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ScriptAccess",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::*.amzn-s3-demo-bucket/scripts"
      ]
    },
    {
      "Sid": "LoggingAccess",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket/logs/*"
      ]
    },
    {
      "Sid": "GlueCatalogAccess",
      "Effect": "Allow",
      "Action": [
        "glue:Get*",
        "glue:Create*",
        "glue:Update*"
      ],
    }
  ]
}
```

```
    "Resource": [
      "*"
    ]
  },
  {
    "Sid": "LakeFormationAccess",
    "Effect": "Allow",
    "Action": [
      "lakeformation:GetDataAccess"
    ],
    "Resource": [
      "*"
    ]
  }
]
```

Langkah 2.1 Konfigurasi izin Lake Formation

- Pekerjaan percikan yang membaca data dari S3 memerlukan izin Lake Formation SELECT.
- Memicu pekerjaan yang write/delete data di S3 memerlukan izin Lake Formation ALL (SUPER).
- Pekerjaan Spark yang berinteraksi dengan katalog Glue Data memerlukan izin DESCRIPTION, ALTER, DROP yang sesuai.

Untuk informasi selengkapnya, lihat [Pemberian izin pada sumber daya Katalog Data](#).

Langkah 3: Inisialisasi sesi Spark untuk Akses Tabel Penuh menggunakan Lake Formation

Prasyarat

AWS Glue Data Catalog harus dikonfigurasi sebagai metastore untuk mengakses tabel Lake Formation.

Atur pengaturan berikut untuk mengkonfigurasi katalog Glue sebagai metastore:

```
--conf spark.sql.catalogImplementation=hive
--conf
  spark.hive.metastore.client.factory.class=com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalog
```

Untuk informasi selengkapnya tentang mengaktifkan Katalog Data untuk EMR Tanpa Server, lihat konfigurasi [Metastore](#) untuk EMR Tanpa Server.

Untuk mengakses tabel yang terdaftar dengan AWS Lake Formation, konfigurasi berikut perlu disetel selama inisialisasi Spark untuk mengonfigurasi Spark agar menggunakan kredensial Lake Formation AWS .

Hive

```
--conf spark.hadoop.fs.s3.credentialsResolverClass=com.amazonaws.glue.accesscontrol.AWSLakeFormationCredentialsProvider
--conf spark.hadoop.fs.s3.useDirectoryHeaderAsFolderObject=true
--conf spark.hadoop.fs.s3.folderObject.autoAction.disabled=true
--conf spark.sql.catalog.skipLocationValidationOnCreateTable.enabled=true
--conf spark.sql.catalog.createDirectoryAfterTable.enabled=true
--conf spark.sql.catalog.dropDirectoryBeforeTable.enabled=true
```

Iceberg

```
--conf spark.sql.catalog.spark_catalog=org.apache.iceberg.spark.SparkSessionCatalog
--conf spark.sql.catalog.spark_catalog.warehouse=S3_DATA_LOCATION
--conf spark.sql.catalog.spark_catalog.client.region=REGION
--conf spark.sql.catalog.spark_catalog.type=glue
--conf spark.sql.catalog.spark_catalog.glue.account-id=ACCOUNT_ID
--conf spark.sql.catalog.spark_catalog.glue.lakeformation-enabled=true
--conf spark.sql.catalog.dropDirectoryBeforeTable.enabled=true
```

Delta Lake

```
--conf spark.hadoop.fs.s3.credentialsResolverClass=com.amazonaws.glue.accesscontrol.AWSLakeFormationCredentialsProvider
--conf spark.hadoop.fs.s3.useDirectoryHeaderAsFolderObject=true
--conf spark.hadoop.fs.s3.folderObject.autoAction.disabled=true
--conf spark.sql.catalog.skipLocationValidationOnCreateTable.enabled=true
--conf spark.sql.catalog.createDirectoryAfterTable.enabled=true
--conf spark.sql.catalog.dropDirectoryBeforeTable.enabled=true
```

Hudi

```
--conf spark.hadoop.fs.s3.credentialsResolverClass=com.amazonaws.glue.accesscontrol.AWSLakeFormationCredentialsProvider
```

```

--conf spark.hadoop.fs.s3.useDirectoryHeaderAsFolderObject=true
--conf spark.hadoop.fs.s3.folderObject.autoAction.disabled=true
--conf spark.sql.catalog.skipLocationValidationOnCreateTable.enabled=true
--conf spark.sql.catalog.createDirectoryAfterTable.enabled=true
--conf spark.sql.catalog.dropDirectoryBeforeTable.enabled=true
--conf spark.jars=/usr/lib/hudi/hudi-spark-bundle.jar
--conf spark.sql.extensions=org.apache.spark.sql.hudi.HoodieSparkSessionExtension
--conf
  spark.sql.catalog.spark_catalog=org.apache.spark.sql.hudi.catalog.HoodieCatalog
--conf spark.serializer=org.apache.spark.serializer.KryoSerializer

```

- `spark.hadoop.fs.s3.credentialsResolverClass=com.amazonaws.glue.accesscontrol.AWSCredentialsProvider`
Konfigurasi Sistem File EMR (EMRFS) atau EMR S3A untuk menggunakan kredensial Lake Formation S3 untuk tabel terdaftar Lake AWS Formation. Jika tabel tidak terdaftar, gunakan kredensial peran runtime pekerjaan.
- `spark.hadoop.fs.s3.useDirectoryHeaderAsFolderObject=true` dan `spark.hadoop.fs.s3.folderObject.autoAction.disabled=true`
Konfigurasi EMRFS untuk menggunakan aplikasi header tipe konten/x-directory alih-alih akhiran `$folder$` saat membuat folder S3. Ini diperlukan saat membaca tabel Lake Formation, karena kredensial Lake Formation tidak mengizinkan membaca folder tabel dengan akhiran `$folder$`.
- `spark.sql.catalog.skipLocationValidationOnCreateTable.enabled=true`:
Konfigurasi Spark untuk melewati validasi kekosongan lokasi tabel sebelum pembuatan. Ini diperlukan untuk tabel terdaftar Lake Formation, karena kredensial Lake Formation untuk memverifikasi lokasi kosong hanya tersedia setelah pembuatan tabel Katalog Data Glue. Tanpa konfigurasi ini, kredensial peran runtime job akan memvalidasi lokasi tabel kosong.
- `spark.sql.catalog.createDirectoryAfterTable.enabled=true`:
Konfigurasi Spark untuk membuat folder Amazon S3 setelah pembuatan tabel di metastore Hive. Ini diperlukan untuk tabel terdaftar Lake Formation, karena kredensial Lake Formation untuk membuat folder S3 hanya tersedia setelah pembuatan tabel Glue Data Catalog.
- `spark.sql.catalog.dropDirectoryBeforeTable.enabled=true`:
Konfigurasi Spark untuk menjatuhkan folder S3 sebelum penghapusan tabel di metastore Hive. Ini diperlukan untuk tabel terdaftar Lake Formation, karena kredensial Lake Formation untuk menjatuhkan folder S3 tidak tersedia setelah penghapusan tabel dari Katalog Data Glue.
- `spark.sql.catalog.<catalog>.glue.lakeformation-enabled=true`:
Konfigurasi katalog Gunung Es untuk menggunakan kredensial AWS Lake Formation S3 untuk tabel terdaftar Lake Formation. Jika tabel tidak terdaftar, gunakan kredensial lingkungan default.

Konfigurasi mode akses tabel penuh di SageMaker Unified Studio

Untuk mengakses tabel terdaftar Lake Formation dari sesi Spark interaktif di JupyterLab notebook, gunakan mode izin kompatibilitas. Gunakan perintah ajaib `%%configure` untuk mengatur konfigurasi Spark Anda. Pilih konfigurasi berdasarkan jenis tabel Anda:

For Hive tables

```
%%configure -f
{
  "conf": {
    "spark.hadoop.fs.s3.credentialsResolverClass":
    "com.amazonaws.glue.accesscontrol.AWSLakeFormationCredentialResolver",
    "spark.hadoop.fs.s3.useDirectoryHeaderAsFolderObject": true,
    "spark.hadoop.fs.s3.folderObject.autoAction.disabled": true,
    "spark.sql.catalog.skipLocationValidationOnCreateTable.enabled": true,
    "spark.sql.catalog.createDirectoryAfterTable.enabled": true,
    "spark.sql.catalog.dropDirectoryBeforeTable.enabled": true
  }
}
```

For Iceberg tables

```
%%configure -f
{
  "conf": {
    "spark.sql.catalog.spark_catalog":
    "org.apache.iceberg.spark.SparkSessionCatalog",
    "spark.sql.catalog.spark_catalog.warehouse": "S3_DATA_LOCATION",
    "spark.sql.catalog.spark_catalog.client.region": "REGION",
    "spark.sql.catalog.spark_catalog.type": "glue",
    "spark.sql.catalog.spark_catalog.glue.account-id": "ACCOUNT_ID",
    "spark.sql.catalog.spark_catalog.glue.lakeformation-enabled": "true",
    "spark.sql.catalog.dropDirectoryBeforeTable.enabled": "true"
  }
}
```

For Delta Lake tables

```
%%configure -f
{
  "conf": {
```

```

    "spark.hadoop.fs.s3.credentialsResolverClass":
    "com.amazonaws.glue.accesscontrol.AWSLakeFormationCredentialResolver",
    "spark.hadoop.fs.s3.useDirectoryHeaderAsFolderObject": true,
    "spark.hadoop.fs.s3.folderObject.autoAction.disabled": true,
    "spark.sql.catalog.skipLocationValidationOnCreateTable.enabled": true,
    "spark.sql.catalog.createDirectoryAfterTable.enabled": true,
    "spark.sql.catalog.dropDirectoryBeforeTable.enabled": true
  }
}

```

For Hudi tables

```

%%configure -f
{
  "conf": {
    "spark.hadoop.fs.s3.credentialsResolverClass":
    "com.amazonaws.glue.accesscontrol.AWSLakeFormationCredentialResolver",
    "spark.hadoop.fs.s3.useDirectoryHeaderAsFolderObject": true,
    "spark.hadoop.fs.s3.folderObject.autoAction.disabled": true,
    "spark.sql.catalog.skipLocationValidationOnCreateTable.enabled": true,
    "spark.sql.catalog.createDirectoryAfterTable.enabled": true,
    "spark.sql.catalog.dropDirectoryBeforeTable.enabled": true,
    "spark.jars": "/usr/lib/hudi/hudi-spark-bundle.jar",
    "spark.sql.extensions":
    "org.apache.spark.sql.hudi.HoodieSparkSessionExtension",
    "spark.sql.catalog.spark_catalog":
    "org.apache.spark.sql.hudi.catalog.HoodieCatalog",
    "spark.serializer": "org.apache.spark.serializer.KryoSerializer"
  }
}

```

Ganti placeholder:

- S3_DATA_LOCATION: Jalur ember S3 Anda
- REGION: AWS wilayah (misalnya, us-east-1)
- ACCOUNT_ID: ID AWS akun Anda

Note

Anda harus mengatur konfigurasi ini sebelum menjalankan operasi Spark apa pun di buku catatan Anda.

Operasi yang Didukung

Operasi ini akan menggunakan kredensial AWS Lake Formation untuk mengakses data tabel.

- CREATE TABLE
- ALTER TABLE
- MASUKKAN KE
- SISIPKAN TIMPA
- UPDATE
- BERGABUNG MENJADI
- DELETE FROM
- MENGANALISIS TABEL
- MEJA PERBAIKAN
- MEJA DROP
- Percikan kueri sumber data
- Spark datasource menulis

Note

Operasi yang tidak tercantum di atas akan terus menggunakan izin IAM untuk mengakses data tabel.

Pertimbangan-pertimbangan

- Jika tabel Hive dibuat menggunakan pekerjaan yang tidak mengaktifkan akses tabel penuh, dan tidak ada catatan yang disisipkan, pembacaan atau penulisan berikutnya dari pekerjaan dengan akses tabel penuh akan gagal. Ini karena EMR Spark tanpa akses tabel penuh menambahkan `$folder$` akhiran ke nama folder tabel. Untuk mengatasi ini, Anda dapat:

- Masukkan setidaknya satu baris ke dalam tabel dari pekerjaan yang tidak mengaktifkan FTA.
- Konfigurasi pekerjaan yang tidak mengaktifkan FTA untuk tidak menggunakan `$folder` \$ akhiran dalam nama folder di S3. Ini dapat dicapai dengan mengatur konfigurasi `spark.hadoop.fs.s3.useDirectoryHeaderAsFolderObject=true` Spark.
- Buat folder S3 di lokasi tabel `s3://path/to/table/table_name` menggunakan konsol S3 atau AWS AWS S3 CLI.
- Akses Tabel Lengkap didukung dengan EMR Filesystem (EMRFS) yang dimulai di Amazon EMR rilis 7.8.0, dan dengan sistem file S3A dimulai pada rilis Amazon EMR 7.10.0.
- Akses Tabel Penuh didukung untuk tabel Hive, Iceberg, Delta, dan Hudi.
- Pertimbangan Hudi FTA Write Support:
 - Hudi FTA menulis require using `HoodieCredentialedHadoopStorage` for credential vending selama eksekusi pekerjaan. Atur konfigurasi berikut saat menjalankan pekerjaan Hudi:
`hoodie.storage.class=org.apache.spark.sql.hudi.storage.HoodieCredentialedHadoopStorage`
 - Dukungan tulis Akses Tabel Lengkap (FTA) untuk Hudi tersedia mulai dari Amazon EMR rilis 7.12.
 - Dukungan penulisan Hudi FTA saat ini hanya berfungsi dengan konfigurasi Hudi default. Pengaturan Hudi khusus atau non-default mungkin tidak sepenuhnya didukung dan dapat mengakibatkan perilaku yang tidak terduga.
 - Pengelompokan untuk tabel Hudi Merge-On-Read (MOR) tidak didukung pada saat ini di bawah mode tulis FTA.
- Pekerjaan yang mereferensikan tabel dengan aturan Lake Formation Fine-Grained Access Control (FGAC) atau Tampilan Katalog Data Glue akan gagal. Untuk menanyakan tabel dengan aturan FGAC atau Tampilan Katalog Data Glue, Anda harus menggunakan mode FGAC. Anda dapat mengaktifkan mode FGAC dengan mengikuti langkah-langkah yang diuraikan dalam AWS dokumentasi: Menggunakan [EMR Tanpa Server dengan Lake AWS Formation untuk kontrol akses butir](#) halus.
- Akses tabel penuh tidak mendukung Spark Streaming.
- Saat menulis Spark DataFrame ke tabel Lake Formation, hanya mode APPEND yang didukung untuk tabel Hive dan Iceberg: `df.write.mode("append").saveAsTable(table_name)`
- Membuat tabel eksternal memerlukan izin IAM.
- Karena Lake Formation menyimpan sementara kredensial dalam pekerjaan Spark, pekerjaan batch Spark atau sesi interaktif yang sedang berjalan mungkin tidak mencerminkan perubahan izin.

- Anda harus menggunakan peran yang ditentukan pengguna dan bukan peran terkait layanan: [Persyaratan Lake Formation untuk](#) peran.

Hudi FTA Write Support - Operasi yang Didukung

Tabel berikut menunjukkan operasi penulisan yang didukung untuk tabel Hudi Copy-On-Write (COW) dan Merge-On-Read (MOR) di bawah mode Akses Tabel Penuh:

Hudi FTA Mendukung Operasi Tulis

Tipe Tabel	Operasi	Perintah Tulis SQL	Status
LEMBU	INSERT	MASUKKAN KE DALAM TABEL	Didukung
LEMBU	INSERT	MASUKKAN KE DALAM TABEL - PARTISI (Statis, Dinamis)	Didukung
LEMBU	INSERT	SISIPKAN TIMPA	Didukung
LEMBU	INSERT	INSERT OVERWRITE - PARTISI (Statis, Dinamis)	Didukung
UPDATE	UPDATE	PERBARUI TABEL	Didukung
LEMBU	UPDATE	PERBARUI TABEL - Ubah Partisi	Tidak Didukung
DELETE	DELETE	HAPUS DARI TABEL	Didukung

Tipe Tabel	Operasi	Perintah Tulis SQL	Status
MENGUBAH	MENGUBAH	UBAH TABEL - GANTI NAMA MENJADI	Tidak Didukung
LEMBU	MENGUBAH	MENGUBAH TABEL - MENGATUR TBLPROPER TIES	Didukung
LEMBU	MENGUBAH	MENGUBAH TABEL - UNSET TBLPROPER TIES	Didukung
LEMBU	MENGUBAH	MENGUBAH TABEL - MENGUBAH KOLOM	Didukung
LEMBU	MENGUBAH	MENGUBAH TABEL - TAMBAHKAN KOLOM	Didukung
LEMBU	MENGUBAH	MENGUBAH TABEL - TAMBAHKAN PARTISI	Didukung
LEMBU	MENGUBAH	UBAH TABEL - JATUHKAN PARTISI	Didukung

Tipe Tabel	Operasi	Perintah Tulis SQL	Status
LEMBU	MENGUBAH	MENGUBAH TABEL - MEMULIHKAN PARTISI	Didukung
LEMBU	MENGUBAH	MEMPERBAIKI PARTISI SINKRONISASI TABEL	Didukung
MENJATUHKAN	MENJATUHKAN	MEJA DROP	Didukung
LEMBU	MENJATUHKAN	DROP TABLE - MEMBERSIHKAN	Didukung
CREATE	CREATE	BUAT TABEL - Dikelola	Didukung
LEMBU	CREATE	BUAT TABEL - PARTISI OLEH	Didukung
LEMBU	CREATE	BUAT TABEL JIKA TIDAK ADA	Didukung
LEMBU	CREATE	BUAT TABEL SEPERTI	Didukung
LEMBU	CREATE	BUAT TABEL SEBAGAI PILIH	Didukung
CREATE	CREATE	BUAT TABEL dengan LOKASI - Tabel Eksternal	Tidak Didukung

Tipe Tabel	Operasi	Perintah Tulis SQL	Status
KERANGKA DATA (SISIPKAN)	KERANGKA DATA (SISIPKAN)	saveAsTable.Menimpa	Didukung
LEMBU	KERANGKA DATA (SISIPKAN)	saveAsTable.Menambahkan	Tidak Didukung
LEMBU	KERANGKA DATA (SISIPKAN)	saveAsTable.Abaikan	Didukung
LEMBU	KERANGKA DATA (SISIPKAN)	saveAsTable.ErrorIfExists	Didukung
LEMBU	KERANGKA DATA (SISIPKAN)	saveAsTable - Tabel eksternal (Path)	Tidak Didukung
LEMBU	KERANGKA DATA (SISIPKAN)	simpan (jalur) - DF v1	Tidak Didukung
MOR	INSERT	MASUKKAN KE DALAM TABEL	Didukung
MOR	INSERT	MASUKKAN KE DALAM TABEL - PARTISI (Statis, Dinamis)	Didukung
MOR	INSERT	SISIPKAN TIMPA	Didukung
MOR	INSERT	INSERT OVERWRITE - PARTISI (Statis, Dinamis)	Didukung

Tipe Tabel	Operasi	Perintah Tulis SQL	Status
UPDATE	UPDATE	PERBARUI TABEL	Didukung
MOR	UPDATE	PERBARUI TABEL - Ubah Partisi	Tidak Didukung
DELETE	DELETE	HAPUS DARI TABEL	Didukung
MENGUBAH	MENGUBAH	UBAH TABEL - GANTI NAMA MENJADI	Tidak Didukung
MOR	MENGUBAH	MENGUBAH TABEL - MENGATUR TBLPROPERTIES	Didukung
MOR	MENGUBAH	MENGUBAH TABEL - UNSET TBLPROPERTIES	Didukung
MOR	MENGUBAH	MENGUBAH TABEL - MENGUBAH KOLOM	Didukung
MOR	MENGUBAH	MENGUBAH TABEL - TAMBAHKAN KOLOM	Didukung

Tipe Tabel	Operasi	Perintah Tulis SQL	Status
MOR	MENGUBAH	MENGUBAH TABEL - TAMBAHKAN PARTISI	Didukung
MOR	MENGUBAH	UBAH TABEL - JATUHKAN PARTISI	Didukung
MOR	MENGUBAH	MENGUBAH TABEL - MEMULIHKAN PARTISI	Didukung
MOR	MENGUBAH	MEMPERBAIKI PARTISI SINKRONISASI TABEL	Didukung
MENJATUHKAN	MENJATUHKAN	MEJA DROP	Didukung
MOR	MENJATUHKAN	DROP TABLE - MEMBERSIHKAN	Didukung
CREATE	CREATE	BUAT TABEL - Dikelola	Didukung
MOR	CREATE	BUAT TABEL - PARTISI OLEH	Didukung
MOR	CREATE	BUAT TABEL JIKA TIDAK ADA	Didukung
MOR	CREATE	BUAT TABEL SEPERTI	Didukung

Tipe Tabel	Operasi	Perintah Tulis SQL	Status
MOR	CREATE	BUAT TABEL SEBAGAI PILIH	Didukung
CREATE	CREATE	BUAT TABEL dengan LOKASI - Tabel Eksternal	Tidak Didukung
KERANGKA DATA (UPSERT)	KERANGKA DATA (UPSERT)	saveAsTable.Menimpa	Didukung
MOR	KERANGKA DATA (UPSERT)	saveAsTable.Menambahkan	Tidak Didukung
MOR	KERANGKA DATA (UPSERT)	saveAsTable.Abaikan	Didukung
MOR	KERANGKA DATA (UPSERT)	saveAsTable.ErrorIfExists	Didukung
MOR	KERANGKA DATA (UPSERT)	saveAsTable - Tabel eksternal (Path)	Tidak Didukung
MOR	KERANGKA DATA (UPSERT)	simpan (jalur) - DF v1	Tidak Didukung
KERANGKA DATA (HAPUS)	KERANGKA DATA (HAPUS)	saveAsTable.Menambahkan	Tidak Didukung
MOR	KERANGKA DATA (HAPUS)	saveAsTable - Tabel eksternal (Path)	Tidak Didukung
MOR	KERANGKA DATA (HAPUS)	simpan (jalur) - DF v1	Tidak Didukung

Tipe Tabel	Operasi	Perintah Tulis SQL	Status
KERANGKA DATA (BULK_INSERT)	KERANGKA DATA (BULK_INSERT)	saveAsTable.Menimpa	Didukung
MOR	KERANGKA DATA (BULK_INSERT)	saveAsTable.Menambahkan	Tidak Didukung
MOR	KERANGKA DATA (BULK_INSERT)	saveAsTable.Abaikan	Didukung
MOR	KERANGKA DATA (BULK_INSERT)	saveAsTable.ErrorIfExists	Didukung
MOR	KERANGKA DATA (BULK_INSERT)	saveAsTable - Tabel eksternal (Path)	Tidak Didukung
MOR	KERANGKA DATA (BULK_INSERT)	simpan (jalur) - DF v1	Tidak Didukung

Menggunakan EMR Tanpa Server dengan AWS Lake Formation kontrol akses berbutir halus

Ikhtisar

Dengan Amazon EMR merilis 7.2.0 dan yang lebih tinggi, manfaatkan AWS Lake Formation untuk menerapkan kontrol akses berbutir halus pada tabel Katalog Data yang didukung oleh S3. Kemampuan ini memungkinkan Anda mengonfigurasi kontrol akses tingkat tabel, baris, kolom, dan sel untuk read kueri dalam pekerjaan Spark Tanpa Server EMR Amazon Anda. Untuk mengonfigurasi kontrol akses berbutir halus untuk pekerjaan batch Apache Spark dan sesi interaktif, gunakan EMR Studio. Lihat bagian berikut untuk mempelajari lebih lanjut tentang Lake Formation dan cara menggunakannya dengan EMR Tanpa Server.

Menggunakan Amazon EMR Tanpa Server dengan AWS Lake Formation dikenakan biaya tambahan. Untuk informasi lebih lanjut, lihat [harga Amazon EMR](#).

Bagaimana EMR Serverless bekerja dengan AWS Lake Formation

Menggunakan EMR Tanpa Server dengan Lake Formation memungkinkan Anda menerapkan lapisan izin pada setiap pekerjaan Spark untuk menerapkan kontrol izin Lake Formation saat EMR Tanpa Server mengeksekusi pekerjaan. EMR Tanpa Server menggunakan [profil sumber daya Spark untuk membuat dua profil](#) untuk melaksanakan pekerjaan secara efektif. Profil pengguna mengeksekusi kode yang disediakan pengguna, sementara profil sistem memberlakukan kebijakan Lake Formation. Untuk informasi lebih lanjut, lihat [Apa itu AWS Lake Formation](#) dan [Pertimbangan dan batasan](#).

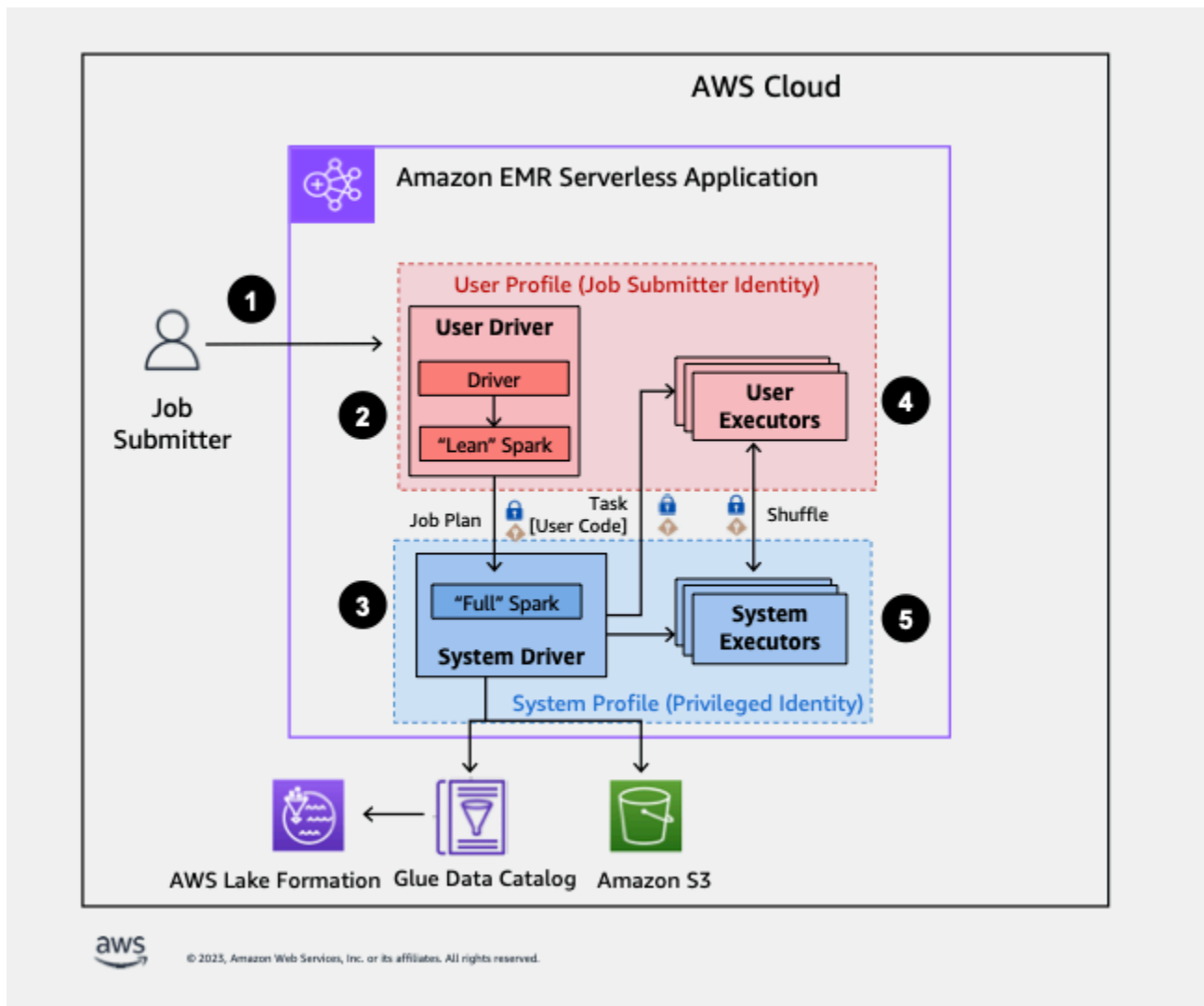
Saat Anda menggunakan kapasitas pra-inisialisasi dengan Lake Formation, kami sarankan Anda memiliki minimal dua driver Spark. Setiap pekerjaan yang mendukung Lake Formation menggunakan dua driver Spark, satu untuk profil pengguna dan satu untuk profil sistem. Untuk kinerja terbaik, gunakan dua kali lipat jumlah driver untuk pekerjaan yang mendukung Lake Formation dibandingkan jika Anda tidak menggunakan Lake Formation.

Saat Anda menjalankan pekerjaan Spark di EMR Tanpa Server, pertimbangkan juga dampak alokasi dinamis pada manajemen sumber daya dan kinerja kluster. Konfigurasi `spark.dynamicAllocation.maxExecutors` jumlah maksimum pelaksana per profil sumber daya berlaku untuk pengguna dan pelaksana sistem. Jika Anda mengonfigurasi angka tersebut agar sama dengan jumlah maksimum pelaksana yang diizinkan, pekerjaan Anda mungkin macet karena satu jenis pelaksana yang menggunakan semua sumber daya yang tersedia, yang mencegah pelaksana lain saat Anda menjalankan pekerjaan.

Jadi Anda tidak kehabisan sumber daya, EMR Serverless menetapkan jumlah maksimum default pelaksana per profil sumber daya menjadi 90% dari nilai `spark.dynamicAllocation.maxExecutors` Anda dapat mengganti konfigurasi ini ketika Anda menentukan `spark.dynamicAllocation.maxExecutorsRatio` dengan nilai antara 0 dan 1. Selain itu, konfigurasi juga properti berikut untuk mengoptimalkan alokasi sumber daya dan kinerja keseluruhan:

- `spark.dynamicAllocation.cachedExecutorIdleTimeout`
- `spark.dynamicAllocation.shuffleTracking.timeout`
- `spark.cleaner.periodicGC.interval`

Berikut ini adalah ikhtisar tingkat tinggi tentang bagaimana EMR Tanpa Server mendapatkan akses ke data yang dilindungi oleh kebijakan keamanan Lake Formation.



1. Seorang pengguna mengirimkan pekerjaan Spark ke aplikasi EMR Tanpa AWS Lake Formation Server yang diaktifkan.
2. EMR Tanpa Server mengirimkan pekerjaan ke driver pengguna dan menjalankan pekerjaan di profil pengguna. Driver pengguna menjalankan versi lean Spark yang tidak memiliki kemampuan untuk meluncurkan tugas, meminta pelaksana, mengakses S3 atau Glue Catalog. Ini membangun rencana kerja.
3. EMR Serverless menyiapkan driver kedua yang disebut driver sistem dan menjalankannya di profil sistem (dengan identitas istimewa). EMR Tanpa Server menyiapkan saluran TLS terenkripsi antara dua driver untuk komunikasi. Driver 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 S3, dan Katalog

Data untuk akses data. Ini meminta pelaksana dan mengkompilasi Job Plan ke dalam urutan tahapan eksekusi.

4. EMR Serverless kemudian menjalankan tahapan pada pelaksana dengan driver pengguna atau driver sistem. 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 AWS Lake Formation atau yang menerapkan filter keamanan didelegasikan ke pelaksana sistem.

Mengaktifkan Lake Formation di Amazon EMR

Untuk mengaktifkan Lake Formation, atur `spark.emr-serverless.lakeformation.enabled` ke `true` bawah `spark-defaults` klasifikasi untuk parameter konfigurasi runtime saat [membuat aplikasi EMR Tanpa Server](#).

```
aws emr-serverless create-application \  
  --release-label emr-7.13.0 \  
  --runtime-configuration '{  
    "classification": "spark-defaults",  
    "properties": {  
      "spark.emr-serverless.lakeformation.enabled": "true"  
    }  
  }' \  
  --type "SPARK"
```

Anda juga dapat mengaktifkan Lake Formation ketika Anda membuat aplikasi baru di EMR Studio. Pilih Gunakan Lake Formation untuk kontrol akses berbutir halus, tersedia di bawah Konfigurasi tambahan.

[Enkripsi antar pekerja](#) diaktifkan secara default saat Anda menggunakan Lake Formation dengan EMR Tanpa Server, jadi Anda tidak perlu mengaktifkan enkripsi antar-pekerja secara eksplisit lagi.

Mengaktifkan Lake Formation untuk pekerjaan Spark

Untuk mengaktifkan Lake Formation untuk pekerjaan Spark individual, setel `spark.emr-serverless.lakeformation.enabled` ke `true` saat menggunakan `spark-submit`.

```
--conf spark.emr-serverless.lakeformation.enabled=true
```

Izin IAM peran runtime pekerjaan

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.

Berikut ini adalah contoh kebijakan tentang cara memberikan izin IAM untuk mengakses skrip di S3, mengunggah log ke S3, izin AWS Glue API, dan izin untuk mengakses Lake Formation.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ScriptAccess",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3::*.amzn-s3-demo-bucket/scripts",
        "arn:aws:s3::*.amzn-s3-demo-bucket/*"
      ]
    },
    {
      "Sid": "LoggingAccess",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3::amzn-s3-demo-bucket/logs/*"
      ]
    },
    {
      "Sid": "GlueCatalogAccess",
      "Effect": "Allow",
      "Action": [
```

```
        "glue:Get*",
        "glue:Create*",
        "glue:Update*"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Sid": "LakeFormationAccess",
    "Effect": "Allow",
    "Action": [
        "lakeformation:GetDataAccess"
    ],
    "Resource": [
        "*"
    ]
}
]
```

Menyiapkan izin Lake Formation untuk peran runtime pekerjaan

Pertama, daftarkan lokasi meja Hive Anda dengan Lake Formation. Kemudian buat izin untuk peran runtime pekerjaan Anda di tabel yang Anda inginkan. Untuk detail lebih lanjut tentang Lake Formation, lihat [Apa itu AWS Lake Formation?](#) di Panduan AWS Lake Formation Pengembang.

Setelah Anda mengatur izin Lake Formation, kirimkan pekerjaan Spark di Amazon EMR Tanpa Server. Untuk informasi lebih lanjut tentang pekerjaan Spark, lihat contoh [Spark](#).

Mengirimkan pekerjaan

Setelah Anda selesai menyiapkan hibah Lake Formation, Anda dapat [mengirimkan pekerjaan Spark di EMR](#) Tanpa Server. Bagian berikut menunjukkan contoh cara mengkonfigurasi dan mengirimkan properti job run.

Persyaratan izin

Tabel tidak terdaftar di AWS Lake Formation

Untuk tabel yang tidak terdaftar AWS Lake Formation, peran runtime pekerjaan mengakses Katalog Data AWS Glue dan data tabel yang mendasarinya di Amazon S3. Ini mengharuskan peran runtime pekerjaan memiliki izin IAM yang sesuai untuk operasi AWS Glue dan Amazon S3.

Tabel terdaftar di AWS Lake Formation

Untuk tabel yang terdaftar AWS Lake Formation, peran runtime pekerjaan mengakses metadata Katalog Data AWS Glue, sementara kredensial sementara yang dijual oleh Lake Formation mengakses data tabel yang mendasarinya di Amazon S3. Izin Lake Formation yang diperlukan untuk menjalankan operasi bergantung pada panggilan AWS Glue Data Catalog dan Amazon S3 API yang memulai tugas Spark dan dapat diringkas sebagai berikut:

- Izin DESCRIBE memungkinkan peran runtime untuk membaca tabel atau metadata database dalam Katalog Data
- Izin ALTER memungkinkan peran runtime untuk memodifikasi tabel atau metadata database dalam Katalog Data
- Izin DROP memungkinkan peran runtime untuk menghapus tabel atau metadata database dari Katalog Data
- Izin SELECT memungkinkan peran runtime membaca data tabel dari Amazon S3
- Izin INSERT memungkinkan peran runtime untuk menulis data tabel ke Amazon S3
- Izin DELETE memungkinkan peran runtime untuk menghapus data tabel dari Amazon S3

Note

Lake Formation mengevaluasi izin dengan malas saat pekerjaan Spark memanggil AWS Glue untuk mengambil metadata tabel dan Amazon S3 untuk mengambil data tabel. Pekerjaan yang menggunakan peran runtime dengan izin yang tidak mencukupi tidak akan gagal sampai Spark melakukan panggilan AWS Glue atau Amazon S3 yang memerlukan izin yang hilang.

Note

Dalam matriks tabel yang didukung berikut:

- Operasi yang ditandai sebagai Didukung secara eksklusif menggunakan kredensial Lake Formation untuk mengakses data tabel untuk tabel yang terdaftar dengan Lake Formation. Jika izin Lake Formation tidak mencukupi, operasi tidak akan kembali ke kredensial peran runtime. Untuk tabel yang tidak terdaftar di Lake Formation, kredensial peran runtime pekerjaan mengakses data tabel.
- Operasi yang ditandai sebagai Didukung dengan izin IAM di lokasi Amazon S3 tidak menggunakan kredensial Lake Formation untuk mengakses data tabel yang mendasarinya di Amazon S3. Untuk menjalankan operasi ini, peran runtime pekerjaan harus memiliki izin Amazon S3 IAM yang diperlukan untuk mengakses data tabel, terlepas dari apakah tabel terdaftar di Lake Formation.

Hive

Operasi	AWS Lake Formation izin	Status Support
SELECT	SELECT	Didukung
CREATE TABLE	CREATE_TABLE	Didukung
BUAT TABEL SEPERTI	CREATE_TABLE	Didukung dengan izin IAM di lokasi Amazon S3
BUAT TABEL SEBAGAI PILIH	CREATE_TABLE	Didukung dengan izin IAM di lokasi Amazon S3
MENGGAMBARAKAN TABEL	MENJELASKAN	Didukung
TBLPROPERTIES	MENJELASKAN	Didukung
TAMPILKAN KOLOM	MENJELASKAN	Didukung
TAMPILKAN PARTISI	MENJELASKAN	Didukung

Operasi	AWS Lake Formation izin	Status Support
TAMPILKAN TABEL BUAT	MENJELASKAN	Didukung
UBAH TABEL tablename	PILIH dan UBAH	Didukung
UBAH LOKASI tablename SET TABEL	-	Tidak didukung
UBAH TABEL tablename TAMBAHKAN PARTISI	PILIH, INSERT dan ALTER	Didukung
MEJA PERBAIKAN	PILIH dan UBAH	Didukung
MEMUAT DATA		Tidak didukung
INSERT	INSERT dan ALTER	Didukung
SISIPKAN TIMPA	PILIH, MASUKKAN, HAPUS dan UBAH	Didukung
MEJA DROP	SELECT, DROP, DELETE dan ALTER	Didukung
MEMOTONG TABEL	PILIH, MASUKKAN, HAPUS dan UBAH	Didukung
Penulis Rangka Data V1	Sama seperti operasi SQL yang sesuai	Didukung saat menambahkan data ke tabel yang ada. Lihat pertimbangan dan batasan untuk informasi lebih lanjut

Operasi	AWS Lake Formation izin	Status Support
Penulis Rangka Data V2	Sama seperti operasi SQL yang sesuai	Didukung saat menambahkan data ke tabel yang ada. Lihat pertimbangan dan batasan untuk informasi lebih lanjut

Iceberg

Operasi	AWS Lake Formation izin	Status Support
SELECT	SELECT	Didukung
CREATE TABLE	CREATE_TABLE	Didukung
BUAT TABEL SEPERTI	CREATE_TABLE	Didukung dengan izin IAM di lokasi Amazon S3
BUAT TABEL SEBAGAI PILIH	CREATE_TABLE	Didukung dengan izin IAM di lokasi Amazon S3
GANTI TABEL SEBAGAI PILIH	PILIH, INSERT dan ALTER	Didukung
MENGGAMBARAKAN TABEL	MENJELASKAN	Didukung dengan izin IAM di lokasi Amazon S3
TBLPROPERTIES	MENJELASKAN	Didukung dengan izin IAM di lokasi Amazon S3
TAMPILKAN TABEL BUAT	MENJELASKAN	Didukung dengan izin IAM di lokasi Amazon S3
ALTER TABLE	PILIH, INSERT dan ALTER	Didukung
MENGUBAH LOKASI SET TABEL	PILIH, INSERT dan ALTER	Didukung dengan izin IAM di lokasi Amazon S3

Operasi	AWS Lake Formation izin	Status Support
MENGUBAH TABEL TULIS DIURUTKAN OLEH	PILIH, INSERT dan ALTER	Didukung dengan izin IAM di lokasi Amazon S3
MENGUBAH TABEL TULIS DIDISTRIB USIKAN OLEH	PILIH, INSERT, dan ALTER	Didukung dengan izin IAM di lokasi Amazon S3
UBAH TABEL GANTI NAMA TABEL	CREATE_TABLE, dan DROP	Didukung
MASUKKAN KE	PILIH, INSERT dan ALTER	Didukung
SISIPKAN TIMPA	PILIH, INSERT dan ALTER	Didukung
DELETE	PILIH, INSERT dan ALTER	Didukung
UPDATE	PILIH, INSERT dan ALTER	Didukung
BERGABUNG MENJADI	PILIH, INSERT dan ALTER	Didukung
MEJA DROP	PILIH, HAPUS, dan JATUHKAN	Didukung
DataFrame Penulis V1	-	Tidak didukung
DataFrame Penulis V2	Sama seperti operasi SQL yang sesuai	Didukung saat menambahkan data ke tabel yang ada. Lihat pertimbangan dan batasan untuk informasi lebih lanjut.

Operasi	AWS Lake Formation izin	Status Support
Tabel metadata	SELECT	Didukung. Tabel tertentu disembunyikan. Lihat pertimbangan dan batasan untuk informasi lebih lanjut.
Prosedur tersimpan	-	<p>Didukung untuk tabel yang memenuhi ketentuan berikut:</p> <ul style="list-style-type: none"> Tabel tidak terdaftar di AWS Lake Formation Tabel yang tidak menggunakan <code>register_table</code> dan <code>migrate</code> <p>Lihat pertimbangan dan batasan untuk informasi lebih lanjut.</p>

Konfigurasi Spark untuk Iceberg: Contoh berikut menunjukkan cara mengkonfigurasi Spark dengan Iceberg. Untuk menjalankan pekerjaan Iceberg, sediakan properti berikut `spark-submit`.

```
--conf spark.sql.catalog.spark_catalog=org.apache.iceberg.spark.SparkSessionCatalog
--conf spark.sql.catalog.spark_catalog.warehouse=<S3_DATA_LOCATION>
--conf spark.sql.catalog.spark_catalog.glue.account-id=<ACCOUNT_ID>
--conf spark.sql.catalog.spark_catalog.client.region=<REGION>
--conf spark.sql.catalog.spark_catalog.glue.endpoint=https://
glue.<REGION>.amazonaws.com
```

Hudi

Operasi	AWS Lake Formation izin	Status Support
SELECT	SELECT	Didukung
CREATE TABLE	CREATE_TABLE	Didukung dengan izin IAM di lokasi Amazon S3
BUAT TABEL SEPERTI	CREATE_TABLE	Didukung dengan izin IAM di lokasi Amazon S3

Operasi	AWS Lake Formation izin	Status Support
BUAT TABEL SEBAGAI PILIH	-	Tidak didukung
MENGGAMBARAKAN TABEL	MENJELASKAN	Didukung dengan izin IAM di lokasi Amazon S3
TBLPROPERTIES	MENJELASKAN	Didukung dengan izin IAM di lokasi Amazon S3
TAMPILKAN KOLOM	MENJELASKAN	Didukung dengan izin IAM di lokasi Amazon S3
TAMPILKAN TABEL BUAT	MENJELASKAN	Didukung dengan izin IAM di lokasi Amazon S3
ALTER TABLE	SELECT	Didukung dengan izin IAM di lokasi Amazon S3
MASUKKAN KE	PILIH dan UBAH	Didukung dengan izin IAM di lokasi Amazon S3
SISIPKAN TIMPA	PILIH dan UBAH	Didukung dengan izin IAM di lokasi Amazon S3
DELETE	-	Tidak didukung
UPDATE	-	Tidak didukung
BERGABUNG MENJADI	-	Tidak didukung
MEJA DROP	PILIH dan DROP	Didukung dengan izin IAM di lokasi Amazon S3
DataFrame Penulis V1	-	Tidak didukung
DataFrame Penulis V2	Sama seperti operasi SQL yang sesuai	Didukung dengan izin IAM di lokasi Amazon S3

Operasi	AWS Lake Formation izin	Status Support
Tabel metadata	-	Tidak didukung
Pemeliharaan meja dan fitur utilitas	-	Tidak didukung

Sampel berikut mengkonfigurasi Spark dengan Hudi, menentukan lokasi file dan properti lain yang diperlukan untuk digunakan.

Konfigurasi percikan untuk Hudi: Cuplikan ini saat digunakan di notebook menentukan jalur ke file JAR bundel Hudi Spark, yang memungkinkan fungsionalitas Hudi di Spark. Ini juga mengkonfigurasi Spark untuk menggunakan AWS Glue Data Catalog sebagai metastore.

```
%%configure -f
{
  "conf": {
    "spark.jars": "/usr/lib/hudi/hudi-spark-bundle.jar",
    "spark.hadoop.hive.metastore.client.factory.class":
"com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveClientFactory",
    "spark.serializer": "org.apache.spark.serializer.JavaSerializer",
    "spark.sql.catalog.spark_catalog":
"org.apache.spark.sql.hudi.catalog.HoodieCatalog",
    "spark.sql.extensions":
"org.apache.spark.sql.hudi.HoodieSparkSessionExtension"
  }
}
```

Konfigurasi percikan untuk Hudi dengan AWS Glue: Cuplikan ini saat digunakan di notebook memungkinkan Hudi sebagai format data-lake yang didukung dan memastikan bahwa pustaka dan dependensi Hudi tersedia.

```
%%configure
{
  "--conf": "spark.serializer=org.apache.spark.serializer.JavaSerializer --conf
spark.sql.catalog.spark_catalog=org.apache.spark.sql.hudi.catalog.HoodieCatalog --
conf
spark.sql.extensions=org.apache.spark.sql.hudi.HoodieSparkSessionExtension",
  "--datalake-formats": "hudi",
  "--enable-glue-datacatalog": True,
```

```

    "--enable-lakeformation-fine-grained-access": "true"
  }

```

Delta Lake

Operasi	AWS Lake Formation izin	Status Support
SELECT	SELECT	Didukung
CREATE TABLE	CREATE_TABLE	Didukung
BUAT TABEL SEPERTI	-	Tidak didukung
BUAT TABEL SEBAGAI PILIH	CREATE_TABLE	Didukung
GANTI TABEL SEBAGAI PILIH	PILIH, INSERT dan ALTER	Didukung
MENGGAMBARAKAN TABEL	MENJELASKAN	Didukung dengan izin IAM di lokasi Amazon S3
TBLPROPERTIES	MENJELASKAN	Didukung dengan izin IAM di lokasi Amazon S3
TAMPILKAN KOLOM	MENJELASKAN	Didukung dengan izin IAM di lokasi Amazon S3
TAMPILKAN TABEL BUAT	MENJELASKAN	Didukung dengan izin IAM di lokasi Amazon S3
ALTER TABLE	PILIH dan INSERT	Didukung
MENGUBAH LOKASI SET TABEL	PILIH dan INSERT	Didukung dengan izin IAM di lokasi Amazon S3
UBAH tablename KLASTER TABEL DENGAN	PILIH dan INSERT	Didukung dengan izin IAM di lokasi Amazon S3

Lowongan kerja debugging

Note

Dengan fitur ini, akses stdout dan stderr log untuk pekerja profil sistem yang mungkin berisi informasi sensitif tanpa filter. Izin berikut harus digunakan hanya untuk mengakses data non-produksi. Untuk aplikasi yang dibuat untuk digunakan dengan pekerjaan produksi, kami sangat menyarankan agar Anda menambahkan izin ini hanya untuk administrator atau pengguna dengan akses data yang lebih tinggi.

Dengan EMR-7.3.0 dan yang lebih baru, EMR Serverless memungkinkan kemampuan self-debugging untuk pekerjaan batch yang mendukung Lake Formation. Untuk melakukannya, gunakan parameter baru `accessSystemProfileLog` di [GetDashboardForJobRunAPI](#). Jika `accessSystemProfileLog` disetel ke `true`, Anda dapat mengakses log stdout dan stderr untuk pekerja profil sistem, yang dapat digunakan untuk men-debug pekerjaan batch EMR Tanpa Server yang diaktifkan Lake Formation.

```
aws emr-serverless get-dashboard-for-job-run \
  --application-id application-id
  --job-run-id job-run-id
  --access-system-profile-logs
```

Izin yang diperlukan

Kepala sekolah yang ingin men-debug pekerjaan batch yang diaktifkan Lake Formation `GetDashboardForJobRun` harus memiliki izin tambahan berikut:

```
{
  "Sid": "AccessSystemProfileLogs",
  "Effect": "Allow",
  "Action": [
    "emr-serverless:GetDashboardForJobRun",
    "emr-serverless:AccessSystemProfileLogs",
    "glue:GetDatabases",
    "glue:SearchTables"
  ],
  "Resource": [
    "arn:aws:emr-serverless:region:account-id:/applications/applicationId/
    jobruns/jobid",
```

```
    "arn:aws:glue:region:account-id:catalog",  
    "arn:aws:glue:region:account-id:database/*",  
    "arn:aws:glue:region:account-id:table/*/*"  
  ]  
}
```

Pertimbangan-pertimbangan

Log profil sistem untuk debugging terlihat untuk pekerjaan yang mengakses database atau tabel di Lake Formation dalam akun yang sama dengan pekerjaan. Mereka tidak terlihat dalam skenario berikut:

- Jika katalog data yang dikelola menggunakan izin Lake Formation memiliki database dan tabel lintas akun
- Jika katalog data yang dikelola menggunakan izin Lake Formation memiliki tautan sumber daya

Bekerja dengan tampilan Glue Data Catalog

Anda dapat membuat dan mengelola tampilan di Katalog Data AWS Glue untuk digunakan dengan EMR Tanpa Server. Ini umumnya dikenal sebagai tampilan AWS Glue Data Catalog. Tampilan ini berguna karena mendukung beberapa mesin kueri SQL, sehingga Anda dapat mengakses tampilan yang sama di berbagai AWS layanan, seperti EMR Tanpa Server, Amazon Athena dan Amazon Redshift.

Dengan membuat tampilan di Katalog Data, gunakan hibah sumber daya dan kontrol akses berbasis tag AWS Lake Formation untuk memberikan akses ke sana. Dengan menggunakan metode kontrol akses ini, Anda tidak perlu mengonfigurasi akses tambahan ke tabel yang Anda referensikan saat membuat tampilan. Metode pemberian izin ini disebut semantik definer, dan pandangan ini disebut tampilan definer. Untuk informasi selengkapnya tentang kontrol akses di Lake Formation, lihat [Memberikan dan mencabut izin pada sumber daya Katalog Data di Panduan Pengembang AWS Lake Formation](#).

Tampilan Katalog Data berguna untuk kasus penggunaan berikut:

- Kontrol akses granular — Anda dapat membuat tampilan yang membatasi akses data berdasarkan izin yang dibutuhkan pengguna. Misalnya, Anda dapat menggunakan tampilan di Katalog Data untuk mencegah karyawan yang tidak bekerja di departemen SDM melihat informasi identitas pribadi (PII).

- Definisi tampilan lengkap — Dengan menerapkan filter pada tampilan Anda di Katalog Data, Anda memastikan bahwa catatan data yang tersedia dalam tampilan di Katalog Data selalu lengkap.
- Keamanan yang ditingkatkan — Definisi kueri yang digunakan untuk membuat tampilan harus lengkap. Manfaat ini berarti bahwa tampilan dalam Katalog Data kurang rentan terhadap perintah SQL dari aktor jahat.
- Berbagi data sederhana — Bagikan data dengan AWS akun lain tanpa memindahkan data. Untuk informasi lebih lanjut, lihat [berbagi data lintas akun di Lake Formation](#).

Membuat tampilan Katalog Data

Ada berbagai cara untuk membuat tampilan Katalog Data. Ini termasuk menggunakan AWS CLI atau Spark SQL. Beberapa contoh mengikuti.

Using SQL

Berikut ini menunjukkan sintaks untuk membuat tampilan Data Catalog. Perhatikan jenis MULTI DIALECT tampilan. Ini membedakan tampilan Katalog Data dari tampilan lain. SECURITYPredikat ditentukan sebagaiDEFINER. Ini menunjukkan tampilan Katalog Data dengan DEFINER semantik.

```
CREATE [ OR REPLACE ] PROTECTED MULTI DIALECT VIEW [IF NOT EXISTS] view_name
[(column_name [COMMENT column_comment], ...)]
[ COMMENT view_comment ]
[TBLPROPERTIES (property_name = property_value, ... )]
SECURITY DEFINER
AS query;
```

Berikut ini adalah contoh CREATE pernyataan, mengikuti sintaks:

```
CREATE PROTECTED MULTI DIALECT VIEW catalog_view
SECURITY DEFINER
AS
SELECT order_date, sum(totalprice) AS price
FROM source_table
GROUP BY order_date
```

Anda juga dapat membuat tampilan dalam mode dry-run, menggunakan SQL, untuk menguji pembuatan tampilan, tanpa benar-benar membuat sumber daya. Menggunakan opsi ini menghasilkan “dry run” yang memvalidasi input dan, jika validasi berhasil, mengembalikan JSON

dari objek tabel AWS Glue yang akan mewakili tampilan. Dalam hal ini, Tampilan sebenarnya tidak dibuat.

```
CREATE [ OR REPLACE ] PROTECTED MULTI DIALECT VIEW view_name
SECURITY DEFINER
[ SHOW VIEW JSON ]
AS view-sql
```

Using the AWS CLI

Note

Saat Anda menggunakan perintah CLI, SQL yang digunakan untuk membuat tampilan tidak diuraikan. Hal ini dapat mengakibatkan kasus di mana tampilan dibuat, tetapi kueri tidak berhasil. Pastikan untuk menguji sintaks SQL Anda sebelum membuat tampilan.

Anda menggunakan perintah CLI berikut untuk membuat tampilan:

```
aws glue create-table --cli-input-json '{
  "DatabaseName": "database",
  "TableInput": {
    "Name": "view",
    "StorageDescriptor": {
      "Columns": [
        {
          "Name": "col1",
          "Type": "data-type"
        },
        ...
        {
          "Name": "col_n",
          "Type": "data-type"
        }
      ],
      "SerdeInfo": {}
    },
    "ViewDefinition": {
      "SubObjects": [
        "arn:aws:glue:aws-region:aws-account-id:table/database/referenced-table1",
        ...
        "arn:aws:glue:aws-region:aws-account-id:table/database/referenced-tableN",

```

```

    ],
    "IsProtected": true,
    "Representations": [
      {
        "Dialect": "SPARK",
        "DialectVersion": "1.0",
        "ViewOriginalText": "Spark-SQL",
        "ViewExpandedText": "Spark-SQL"
      }
    ]
  }
}
}'

```

Operasi tampilan yang didukung

Fragmen perintah berikut menunjukkan kepada Anda berbagai cara untuk bekerja dengan tampilan Katalog Data:

- CREATE VIEW

Membuat tampilan data-katalog. Berikut ini adalah contoh yang menunjukkan pembuatan tampilan dari tabel yang ada:

```

CREATE PROTECTED MULTI DIALECT VIEW catalog_view
SECURITY DEFINER AS SELECT * FROM my_catalog.my_database.source_table

```

- UBAH TAMPILAN

Sintaks yang tersedia:

- ALTER VIEW *view_name* [FORCE] ADD DIALECT AS *query*
- ALTER VIEW *view_name* [FORCE] UPDATE DIALECT AS *query*
- ALTER VIEW *view_name* DROP DIALECT

Anda dapat menggunakan FORCE ADD DIALECT opsi untuk memaksa memperbarui skema dan sub objek sesuai dialek mesin baru. Perhatikan bahwa melakukan hal ini dapat mengakibatkan kesalahan kueri jika Anda juga tidak menggunakannya FORCE untuk memperbarui dialek mesin lainnya. Berikut ini menunjukkan sampel:

```

ALTER VIEW catalog_view FORCE ADD DIALECT

```

```
AS
SELECT order_date, sum(totalprice) AS price
FROM source_table
GROUP BY orderdate;
```

Berikut ini menunjukkan cara mengubah tampilan untuk memperbarui dialek:

```
ALTER VIEW catalog_view UPDATE DIALECT AS
SELECT count(*) FROM my_catalog.my_database.source_table;
```

• DESKRIPSIKAN TAMPILAN

Sintaks yang tersedia untuk menggambarkan tampilan:

- `SHOW COLUMNS {FROM|IN} view_name [{FROM|IN} database_name]`— Jika pengguna memiliki izin AWS Glue dan Lake Formation yang diperlukan untuk menggambarkan tampilan, mereka dapat membuat daftar kolom. Berikut ini menunjukkan beberapa contoh perintah untuk menampilkan kolom:

```
SHOW COLUMNS FROM my_database.source_table;
SHOW COLUMNS IN my_database.source_table;
```

- `DESCRIBE view_name`— Jika pengguna memiliki izin AWS Glue dan Lake Formation yang diperlukan untuk mendeskripsikan tampilan, mereka dapat mencantumkan kolom dalam tampilan bersama dengan metadatanya.

• TAMPILAN DROP

Sintaks yang tersedia:

- `DROP VIEW [IF EXISTS] view_name`

Contoh berikut menunjukkan DROP pernyataan yang menguji apakah tampilan ada sebelum menjatuhkannya:

```
DROP VIEW IF EXISTS catalog_view;
```

• TAMPILKAN TAMPILAN BUAT

- `SHOW CREATE VIEW view_name`- Menunjukkan pernyataan SQL yang menciptakan tampilan yang ditentukan. Berikut ini adalah contoh yang menunjukkan pembuatan tampilan data-katalog:

```
SHOW CREATE TABLE my_database.catalog_view;
```

```
CREATE PROTECTED MULTI DIALECT VIEW my_catalog.my_database.catalog_view (
  net_profit,
  customer_id,
  item_id,
  sold_date)
TBLPROPERTIES (
  'transient_lastDdlTime' = '1736267222')
SECURITY DEFINER AS SELECT * FROM
my_database.store_sales_partitioned_lf WHERE customer_id IN (SELECT customer_id
from source_table limit 10)
```

- TAMPILKAN TAMPILAN

Daftar semua tampilan dalam katalog seperti tampilan biasa, tampilan multi-dialek (MDV), dan MDV tanpa dialek Spark. Sintaks yang tersedia adalah sebagai berikut:

- `SHOW VIEWS [{ FROM | IN } database_name] [LIKE regex_pattern]:`

Berikut ini menunjukkan perintah sampel untuk menampilkan tampilan:

```
SHOW VIEWS IN marketing_analytics LIKE 'catalog_view*';
```

Untuk informasi selengkapnya tentang membuat dan mengonfigurasi tampilan data-katalog, lihat tampilan [Katalog Data AWS Glue Bangunan di Panduan Pengembang AWS Lake Formation](#) .

Menanyakan tampilan Katalog Data

Setelah membuat tampilan Katalog Data, Anda dapat melakukan kueri menggunakan pekerjaan Amazon EMR Serverless Spark yang mengaktifkan kontrol akses berbutir halus. AWS Lake Formation Peran runtime pekerjaan harus memiliki SELECT izin Lake Formation pada tampilan Katalog Data. Anda tidak perlu memberikan akses ke tabel dasar yang direferensikan dalam tampilan.

Setelah semuanya disiapkan, Anda dapat menanyakan tampilan Anda. Misalnya, setelah membuat aplikasi EMR Tanpa Server di EMR Studio, jalankan kueri berikut untuk mengakses tampilan.

```
SELECT * from my_database.catalog_view LIMIT 10;
```

Fungsi yang bermanfaat adalah `invoker_principal`. Ia mengembalikan pengenal unik dari peran runtime pekerjaan EMRS. Ini dapat digunakan untuk mengontrol output tampilan, berdasarkan prinsip pemanggilan. Anda dapat menggunakan ini untuk menambahkan kondisi dalam tampilan

yang menyempurnakan hasil kueri, berdasarkan peran pemanggilan. Peran runtime pekerjaan harus memiliki izin untuk tindakan `LakeFormation:GetDataLakePrincipal` IAM untuk menggunakan fungsi ini.

```
select invoker_principal();
```

Anda dapat menambahkan fungsi ini ke WHERE klausa, misalnya, untuk memperbaiki hasil kueri.

Pertimbangan dan batasan

Saat Anda membuat tampilan Katalog Data, berikut ini berlaku:

- Anda hanya dapat membuat tampilan Katalog Data dengan Amazon EMR 7.6 ke atas.
- Penentu tampilan Katalog Data harus memiliki SELECT akses ke tabel dasar dasar yang diakses oleh tampilan. Membuat tampilan Katalog Data gagal jika tabel dasar tertentu memiliki filter Lake Formation yang dikenakan pada peran definer.
- Tabel dasar tidak boleh memiliki izin `IAMAllowedPrincipals` data lake di Lake Formation. Jika ada, kesalahan tampilan Multi Dialek mungkin hanya referensi tabel tanpa izin `IAMAllowedPrincipals` terjadi.
- Lokasi Amazon S3 tabel harus terdaftar sebagai lokasi danau data Lake Formation. Jika tabel tidak terdaftar, kesalahan tampilan Multi Dialek mungkin hanya referensi tabel terkelola Lake Formation terjadi. Untuk informasi tentang cara mendaftarkan lokasi Amazon S3 di Lake Formation, lihat [Mendaftarkan lokasi Amazon S3](#) di AWS Lake Formation Panduan Pengembang.
- Anda hanya dapat membuat tampilan Katalog PROTECTED Data. UNPROTECTED tampilan tidak didukung.
- Anda tidak dapat mereferensikan tabel di AWS akun lain dalam definisi tampilan Katalog Data. Anda juga tidak dapat mereferensikan tabel di akun yang sama yang berada di wilayah terpisah.
- Untuk berbagi data di seluruh akun atau wilayah, seluruh tampilan harus dibagikan lintas akun dan lintas wilayah, menggunakan tautan sumber daya Lake Formation.
- Fungsi yang ditentukan pengguna (UDFs) tidak didukung.
- Anda dapat menggunakan tampilan berdasarkan tabel Iceberg. Format meja terbuka Apache Hudi dan Delta Lake juga didukung.
- Anda tidak dapat mereferensikan tampilan lain dalam tampilan Katalog Data.
- Skema tampilan AWS Glue Data Catalog selalu disimpan menggunakan huruf kecil. Misalnya, jika Anda menggunakan pernyataan DDL untuk membuat tampilan Katalog Data Glue dengan kolom

bernamaCastle, kolom yang dibuat dalam Katalog Data Glue akan dibuat huruf kecil, untuk castle. Jika Anda kemudian menentukan nama kolom dalam kueri DHTML sebagai Castle atauCASTLE, EMR Spark akan membuat nama huruf kecil bagi Anda untuk menjalankan kueri. Tetapi judul kolom ditampilkan menggunakan casing yang Anda tentukan dalam kueri.

Jika Anda ingin kueri gagal dalam kasus di mana nama kolom yang ditentukan dalam kueri DMLtidak cocok dengan nama kolom di Katalog Data Glue, aturspark.sql.caseSensitive=true.

Dukungan format meja terbuka

EMR Tanpa Server mendukung kueri SELECT di Apache Hive, Apache Iceberg, Delta Lake (7.6.0+), dan Apache Hudi (7.6.0+). Dimulai dengan EMR 7.12, operasi DHTML dan DDL yang memodifikasi data tabel didukung untuk tabel Apache Hive, Apache Iceberg, dan Delta Lake menggunakan kredensi penjual Lake Formation.

Pertimbangan dan batasan

Umum

Tinjau batasan berikut saat menggunakan Lake Formation dengan EMR Serverless.

Note

Saat Anda mengaktifkan Lake Formation untuk pekerjaan Spark di EMR Tanpa Server, pekerjaan tersebut meluncurkan driver sistem dan driver pengguna. Jika Anda menentukan kapasitas pra-inisialisasi saat peluncuran, ketentuan driver dari kapasitas pra-inisialisasi, dan jumlah driver sistem sama dengan jumlah driver pengguna yang Anda tentukan. Jika Anda memilih kapasitas On Demand, EMR Serverless meluncurkan driver sistem selain driver pengguna. Untuk memperkirakan biaya yang terkait dengan pekerjaan EMR Tanpa Server dengan Lake Formation Anda, gunakan [AWS Kalkulator Harga](#)

- [Amazon EMR Tanpa Server dengan Lake Formation tersedia di semua Wilayah Tanpa Server EMR yang didukung.](#)
- Aplikasi yang mendukung Lake Formation tidak mendukung penggunaan gambar [EMR](#) Tanpa Server yang disesuaikan.

- Anda tidak dapat mematikan `DynamicResourceAllocation` untuk pekerjaan Lake Formation.
- Anda hanya dapat menggunakan Lake Formation dengan pekerjaan Spark.
- EMR Tanpa Server dengan Lake Formation hanya mendukung satu sesi Spark selama pekerjaan.
- EMR Tanpa Server 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
 - Kontrol akses untuk kolom bersarang
- EMR Tanpa Server memblokir fungsionalitas yang mungkin merusak isolasi lengkap driver sistem, termasuk yang berikut ini:
 - UDTs, HiveUDFs, dan fungsi apa pun yang ditentukan pengguna yang melibatkan kelas khusus
 - Sumber data kustom
 - Pasokan stoples tambahan untuk ekstensi Spark, konektor, atau metastore
 - Perintah `ANALYZE TABLE`
- [Jika aplikasi EMR Tanpa Server Anda berada dalam subnet pribadi dengan titik akhir VPC untuk Amazon S3 dan Anda melampirkan kebijakan titik akhir untuk mengontrol akses, sebelum pekerjaan Anda dapat mengirim data log ke Amazon S3 Terkelola, sertakan izin yang dirinci dalam Penyimpanan terkelola AWS dalam kebijakan VPC Anda ke titik akhir gateway S3.](#) Untuk permintaan pemecahan masalah, hubungi AWS dukungan.
- Dimulai dengan Amazon EMR 7.9.0, Spark FGAC mendukung AFile Sistem S3 saat digunakan dengan skema `s3a://`.
- Amazon EMR 7.11 mendukung pembuatan tabel terkelola menggunakan CTAS.
- Amazon EMR 7.12 mendukung pembuatan tabel terkelola dan eksternal menggunakan CTAS.

Izin

- Untuk menegakkan kontrol akses, `EXPLORE PLAN` dan operasi DDL seperti `DESCRIBE TABLE` tidak mengekspos informasi terbatas.
- Saat Anda mendaftarkan lokasi tabel dengan Lake Formation, akses data menggunakan kredensial tersimpan Lake Formation, bukan izin IAM peran runtime pekerjaan EMR Tanpa Server. Pekerjaan akan gagal jika peran terdaftar untuk lokasi tabel salah dikonfigurasi, bahkan ketika peran runtime memiliki izin IAM S3 untuk lokasi tersebut.

- Dimulai dengan Amazon EMR 7.12, Anda dapat menulis ke tabel Hive dan Iceberg yang ada menggunakan DataFrameWriter (V2) dengan kredensial Lake Formation dalam mode append. Untuk operasi menimpa atau saat membuat tabel baru, EMR menggunakan kredensial peran runtime untuk memodifikasi data tabel.
- Batasan berikut berlaku saat menggunakan tampilan atau tabel cache sebagai data sumber (batasan ini tidak berlaku untuk tampilan AWS Glue Data Catalog):
 - Untuk operasi MERGE, DELETE, dan UPDATE
 - Didukung: Menggunakan tampilan dan tabel cache sebagai tabel sumber.
 - Tidak didukung: Menggunakan tampilan dan tabel cache dalam klausa penetapan dan kondisi.
 - Untuk CREATE OR REPLACE dan REPLACE TABLE AS SELECT operasi:
 - Tidak didukung: Menggunakan tampilan dan tabel cache sebagai tabel sumber.
- Tabel Delta Lake dengan UDFs data sumber mendukung operasi MERGE, DELETE, dan UPDATE hanya ketika vektor penghapusan diaktifkan.

Log dan debugging

- EMR Tanpa Server membatasi akses ke driver sistem Log Spark pada aplikasi yang mendukung Lake Formation. Karena driver sistem berjalan dengan izin tinggi, peristiwa dan log yang dihasilkan driver sistem dapat mencakup informasi sensitif. Untuk mencegah pengguna atau kode yang tidak sah mengakses data sensitif ini, EMR Serverless menonaktifkan akses ke log driver sistem.
- Log profil sistem selalu disimpan dalam penyimpanan terkelola - ini adalah pengaturan wajib yang tidak dapat dinonaktifkan. Log ini disimpan dengan aman dan dienkripsi menggunakan kunci KMS yang Dikelola Pelanggan atau kunci KMS Terkelola AWS .

Gunung es

Tinjau pertimbangan berikut 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 `metadatabasehistory`, `metadatabase_log_entries`, `snapshots`, `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 terdaftar 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.

Pemecahan masalah

Lihat bagian berikut untuk solusi pemecahan masalah.

Pencatatan log

EMR Tanpa Server menggunakan profil sumber daya Spark untuk membagi eksekusi pekerjaan. EMR Tanpa Server menggunakan profil pengguna untuk menjalankan kode yang Anda berikan, sementara profil sistem memberlakukan kebijakan Lake Formation. Anda dapat mengakses log untuk tugas yang dijalankan sebagai profil pengguna.

[Untuk informasi selengkapnya tentang men-debug pekerjaan yang mendukung Lake Formation, lihat Lowongan Debugging.](#)

UI Langsung dan Server Sejarah Spark

Live UI dan Spark History Server memiliki semua peristiwa Spark yang dihasilkan dari profil pengguna dan peristiwa yang disunting yang dihasilkan dari driver sistem.

Anda dapat melihat semua tugas dari pengguna dan driver sistem di tab Executors. Namun, tautan log hanya tersedia untuk profil pengguna. Selain itu, beberapa informasi disunting dari Live UI, seperti jumlah catatan keluaran.

Job gagal dengan izin Lake Formation yang tidak mencukupi

Pastikan bahwa peran runtime pekerjaan Anda memiliki izin untuk menjalankan SELECT dan DESCRIBE pada tabel yang Anda akses.

Job dengan eksekusi RDD gagal

EMR Tanpa Server saat ini tidak mendukung operasi kumpulan data terdistribusi (RDD) yang tangguh 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 Tanpa Server 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](#) .

Spark API terdaftar yang diizinkan kontrol akses berbutir halus asli PySpark

Untuk menjaga keamanan dan kontrol akses data, Spark fine-grained access control (FGAC) membatasi fungsi tertentu. PySpark Pembatasan ini diberlakukan melalui:

- Pemblokiran eksplisit yang mencegah eksekusi fungsi
- Ketidakcocokan arsitektur yang membuat fungsi tidak berfungsi
- Fungsi yang dapat menimbulkan kesalahan, mengembalikan akses pesan yang ditolak, atau tidak melakukan apa pun saat dipanggil

PySpark Fitur berikut tidak didukung di Spark FGAC:

- Operasi RDD (diblokir dengan Spark RDDUnsupported Exception)
- Spark Connect (tidak didukung)
- Spark Streaming (tidak didukung)

Meskipun kami telah menguji fungsi yang terdaftar di lingkungan Native Spark FGAC dan mengonfirmasi bahwa fungsi tersebut berfungsi seperti yang diharapkan, pengujian kami biasanya hanya mencakup penggunaan dasar setiap API. Fungsi dengan beberapa jenis input atau jalur logika kompleks mungkin memiliki skenario yang belum diuji.

Untuk fungsi apa pun yang tidak tercantum di sini dan tidak jelas bagian dari kategori yang tidak didukung di atas, kami merekomendasikan:

- Mengujinya terlebih dahulu di lingkungan gamma atau penerapan skala kecil
- Memverifikasi perilaku mereka sebelum menggunakannya dalam produksi

Note

Jika Anda melihat metode kelas terdaftar tetapi bukan kelas dasarnya, metode tersebut harus tetap berfungsi—itu hanya berarti kita belum secara eksplisit memverifikasi konstruktor kelas dasar.

PySpark API diatur ke dalam modul. Dukungan umum untuk metode dalam setiap modul dirinci dalam tabel di bawah ini.

Nama modul	Status	Catatan
pyspark_core	Didukung	Modul ini berisi kelas RDD utama, dan fungsi-fungsi ini sebagian besar tidak didukung.
pyspark_sql	Didukung	
pyspark_testing	Didukung	
pyspark_resource	Didukung	
pyspark_streaming	Diblokir	Penggunaan streaming diblokir di Spark FGAC.
pyspark_mllib	Eksperimental	Modul ini berisi operasi MLberbasis RDD, dan fungsi-fungsi ini sebagian besar tidak didukung. Modul ini tidak diuji secara menyeluruh.
pyspark_ml	Eksperimental	Modul ini berisi operasi DataFrame berbasis ML, dan fungsi-fungsi ini sebagian besar didukung.

Nama modul	Status	Catatan
		Modul ini tidak diuji secara menyeluruh.
pyspark_panda	Didukung	
pyspark_pandas_slow	Didukung	
pyspark_connect	Diblokir	Penggunaan Spark Connect diblokir di Spark FGAC.
pyspark_pandas_connect	Diblokir	Penggunaan Spark Connect diblokir di Spark FGAC.
pyspark_pandas_slow_connect	Diblokir	Penggunaan Spark Connect diblokir di Spark FGAC.
pyspark_errors	Eksperimental	Modul ini tidak diuji secara menyeluruh. Kelas kesalahan khusus tidak dapat digunakan.

Daftar Izin API

Untuk daftar yang dapat diunduh dan lebih mudah dicari, file dengan modul dan kelas tersedia di [fungsi Python yang diizinkan di Native FGAC](#).

Enkripsi antar pekerja

Dengan Amazon EMR versi 6.15.0 dan yang lebih tinggi, aktifkan komunikasi terenkripsi TLS timbal balik antara pekerja dalam pekerjaan Spark Anda. Saat diaktifkan, EMR Tanpa Server secara otomatis menghasilkan dan mendistribusikan sertifikat unik untuk setiap pekerja yang disediakan di bawah pekerjaan Anda. Ketika para pekerja ini berkomunikasi untuk bertukar pesan kontrol atau mentransfer data shuffle, mereka membuat koneksi TLS timbal balik dan menggunakan sertifikat yang dikonfigurasi untuk memverifikasi identitas satu sama lain. Jika pekerja tidak dapat

memverifikasi sertifikat lain, jabat tangan TLS gagal, dan EMR Serverless membatalkan koneksi di antara mereka.

Jika Anda menggunakan Lake Formation dengan EMR Serverless, enkripsi Mutual-TLS diaktifkan secara default.

Mengaktifkan enkripsi TLS timbal balik pada EMR Tanpa Server

Untuk mengaktifkan enkripsi TLS timbal balik pada aplikasi spark Anda, atur `spark.ssl.internode.enabled` ke `true` saat membuat aplikasi [EMR Tanpa Server](#). Jika Anda menggunakan AWS konsol untuk membuat aplikasi EMR Tanpa Server, pilih Gunakan pengaturan khusus, lalu perluas konfigurasi Aplikasi, dan masukkan aplikasi Anda. `runtimeConfiguration`

```
aws emr-serverless create-application \  
--release-label emr-6.15.0 \  
--runtime-configuration '{  
  "classification": "spark-defaults",  
  "properties": {"spark.ssl.internode.enabled": "true"}  
}' \  
--type "SPARK"
```

Jika Anda ingin mengaktifkan enkripsi TLS timbal balik untuk menjalankan tugas percikan individual, setel `spark.ssl.internode.enabled` ke `true` saat menggunakan `spark-submit`

```
--conf spark.ssl.internode.enabled=true
```

Enkripsi Disk dengan KMS CMK

EMR Tanpa Server mengenkripsi semua disk yang dilampirkan ke pekerja secara default menggunakan kunci enkripsi milik layanan. Anda dapat memilih untuk mengenkripsi disk ini menggunakan kunci terkelola AWS KMS pelanggan Anda sendiri (). CMKs Ini memberi Anda kontrol lebih besar atas kunci enkripsi Anda, termasuk kemampuan untuk membuat dan memelihara kebijakan utama, dan mengaudit penggunaan kunci.

Anda dapat mengonfigurasi enkripsi disk baik saat membuat aplikasi atau saat mengirimkan pekerjaan individual. Ketika diaktifkan di tingkat aplikasi, semua pekerjaan pada aplikasi tersebut mewarisi pengaturan enkripsi. Anda juga dapat mengganti default aplikasi dengan menentukan konfigurasi enkripsi disk saat mengirimkan pekerjaan.

Note

EMR Enkripsi disk tanpa server hanya mendukung kunci KMS simetris. Tombol KMS asimetris tidak didukung. Anda harus menggunakan kunci KMS enkripsi simetris yang dibuat di. AWS KMS Untuk informasi lebih lanjut tentang AWS KMS, lihat [Apa itu AWS KMS?](#)

Menggunakan Konteks Enkripsi

Secara opsional, EMR Serverless menggunakan konteks enkripsi untuk menyediakan data otentikasi tambahan untuk operasi enkripsi. Konteks enkripsi adalah sekumpulan pasangan kunci-nilai yang dapat berisi data otentikasi tambahan non-rahasia. Konteks enkripsi terikat secara kriptografis ke data terenkripsi, sehingga konteks enkripsi yang sama diperlukan untuk mendekripsi data.

Di EMR Tanpa Server, Anda dapat menentukan konteks enkripsi khusus saat mengonfigurasi enkripsi disk. Konteks enkripsi ini disertakan dalam AWS CloudTrail log untuk membantu Anda mengidentifikasi dan memahami operasi KMS Anda.

Note

Jangan menyimpan informasi sensitif dalam konteks enkripsi seperti yang muncul di plaintext di AWS CloudTrail log.

Mengonfigurasi Enkripsi Disk dengan Kunci yang Dikelola Pelanggan

CreateApplication

Untuk mengenkripsi disk dengan kunci KMS Anda sendiri, sertakan `diskEncryptionConfiguration` parameter saat membuat aplikasi EMR Tanpa Server.

```
aws emr-serverless create-application \  
  --type TYPE \  
  --name APPLICATION_ID \  
  --release-label RELEASE_LABEL \  
  --region AWS_REGION \  
  --disk-encryption-configuration '{  
    "encryptionKeyArn": "key-arn",
```

```

    "encryptionContext": {
      "key": "value"
    }
  }'

```

UpdateApplication

Untuk memperbarui konteks enkripsi and/or ARN kunci KMS, tentukan `diskEncryptionConfiguration` parameter dengan nilai baru saat memperbarui aplikasi.

```

aws emr-serverless update-application \
  --name APPLICATION_ID \
  --region AWS_REGION \
  --disk-encryption-configuration '{
    "encryptionKeyArn": "key-arn",
    "encryptionContext": {
      "key": "value"
    }
  }'

```

Note

Untuk menghapus enkripsi disk yang dikonfigurasi pada aplikasi, berikan kosong `diskEncryptionConfiguration` selama pembaruan aplikasi.

StartJobRun

Untuk mengenkripsi disk dengan kunci KMS Anda sendiri, gunakan `diskEncryptionConfiguration` konfigurasi saat Anda mengirimkan pekerjaan.

```

--configuration-overrides '{
  "diskEncryptionConfiguration": {
    "encryptionKeyArn": "key-arn",
    "encryptionContext": {
      "key": "value"
    }
  }
}'

```

Titik akhir Public Livy

Untuk mengenkripsi disk dengan kunci KMS Anda sendiri saat membuat sesi Spark melalui titik akhir Livy publik, tentukan konfigurasi enkripsi di objek sesi. `conf`

```
data = {
  "kind": "pyspark",
  "heartbeatTimeoutInSeconds": 60,
  "conf": {
    "emr-serverless.session.executionRoleArn": "role_arn",
    "spark.emr-serverless.disk.encryptionKeyArn": "key-arn",
    "spark.emr-serverless.disk.encryptionContext": "key1:value1,key2:value2" #
Optional
  }
}

# Send request to create a session with the Livy API endpoint
request = AWSRequest(method='POST', url=endpoint + "/sessions", data=json.dumps(data),
headers=headers)
```

Izin yang diperlukan untuk enkripsi disk

Izin kunci enkripsi untuk EMR Tanpa Server

Ketika Anda mengenkripsi disk dengan kunci enkripsi Anda sendiri, Anda harus mengkonfigurasi izin kunci KMS berikut untuk prinsipal: `emr-serverless.amazonaws.com`

- `kms:GenerateDataKey`: Untuk menghasilkan kunci data untuk mengenkripsi volume disk
- `kms:Decrypt`: Untuk mendekripsi kunci data saat mengakses konten disk terenkripsi

```
{
  "Effect": "Allow",
  "Principal": {
    "Service": "emr-serverless.amazonaws.com"
  },
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey"
  ],
  "Resource": "*",
}
```

```

"Condition": {
  "StringLike": {
    "aws:SourceArn": "arn:aws:emr-serverless:region:aws-account-id:/
applications/application-id"
  },
  "StringEquals": {
    "kms:EncryptionContext:applicationId": "application-id",
    "aws:SourceAccount": "aws-account-id"
  }
}
}

```

Sebagai praktik keamanan terbaik, kami menyarankan Anda menambahkan kunci `aws:SourceArn` kondisi ke kebijakan kunci KMS. Kunci kondisi global IAM `aws:SourceArn` membantu memastikan bahwa EMR Tanpa Server menggunakan kunci KMS hanya untuk ARN aplikasi. Selain itu, termasuk kunci `aws:SourceAccount` kondisi menyediakan lapisan keamanan lain dengan membatasi penggunaan kunci KMS Anda untuk permintaan yang berasal dari ID AWS akun yang ditentukan dalam kondisi.

Peran runtime pekerjaan harus memiliki izin berikut dalam kebijakan IAM-nya:

```

{
  "Sid": "Enable GDK and Decrypt",
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:GenerateDataKey",
      "kms:Decrypt"
    ],
    "Resource": "key-arn"
  }
}

```

Izin pengguna yang diperlukan

Pengguna yang mengirimkan pekerjaan harus memiliki izin untuk menggunakan kunci. Anda dapat menentukan izin dalam kebijakan kunci KMS atau kebijakan IAM untuk pengguna, grup, atau peran. Jika pengguna yang mengirimkan pekerjaan tidak memiliki izin kunci KMS, EMR Tanpa Server menolak pengiriman job run.

Contoh kebijakan kunci

Kebijakan utama berikut memberikan izin untuk `kms:DescribeKey`, `kms:GenerateDataKey` dan `kms:Decrypt`:

- `kms:DescribeKey`: Untuk memverifikasi bahwa kunci KMS yang dikelola pelanggan diaktifkan dan SIMETRIS sebelum menggunakannya.

```
{
  "Sid": "Enable DescribeKey",
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:user/user-name"
  },
  "Action": [
    "kms:DescribeKey"
  ],
  "Resource": "*"
},
{
  "Sid": "Enable GDK and Decrypt",
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:user/user-name"
  },
  "Action": [
    "kms:GenerateDataKey",
    "kms:Decrypt"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:ViaService": "emr-serverless.region.amazonaws.com",
      "kms:EncryptionContext:key": "value"
    }
  }
}
```

Sebagai praktik keamanan terbaik, kami menyarankan Anda menambahkan kunci `kms:viaService` kondisi ke kebijakan kunci KMS. Ini membatasi penggunaan kunci KMS untuk permintaan validasi hanya dari `emr-serverless`.

Contoh kebijakan IAM

Kebijakan IAM berikut memberikan izin untuk `kms:DescribeKey`, `kms:GenerateDataKey` dan `kms:Decrypt`

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:DescribeKey",
      "kms:GenerateDataKey",
      "kms:Decrypt"
    ],
    "Resource": "key-arn"
  }
}
```

Pemantauan Penggunaan Kunci

Anda dapat memantau penggunaan kunci terkelola pelanggan Anda di EMR Tanpa Server melalui AWS CloudTrail. AWS CloudTrail menangkap semua panggilan API ke AWS KMS sebagai peristiwa, termasuk panggilan dari konsol EMR Tanpa Server, EMR API Tanpa Server, CLI, atau SDK. AWS CloudTrail

Informasi yang diambil mencakup konteks enkripsi yang Anda tentukan, yang dapat membantu Anda mengidentifikasi dan mengaudit sumber daya EMR Tanpa Server tertentu yang menggunakan kunci KMS Anda. Misalnya, Anda mungkin melihat peristiwa yang mirip dengan yang berikut ini di AWS CloudTrail. Untuk informasi selengkapnya tentang penggunaan AWS CloudTrail, lihat [Panduan AWS CloudTrail Pengguna](#).

GenerateDataKey

Contoh peristiwa untuk `GenerateDataKey` operasi saat EMR Serverless membuat volume disk terenkripsi

```
{
  "eventVersion": "1.11",
  "userIdentity": {
    "type": "AWSService",
    "principalId": "user",
```

```

    "invokedBy": "AWS Internal"
  },
  "eventTime": "2025-07-28T21:43:51Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "GenerateDataKey",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "ipAddress",
  "userAgent": "userAgent",
  "requestParameters": {
    "encryptionContext": {
      "applicationId": "test"
    },
    "keyId": "arn:aws:kms:region:accountId:key/ffffffff-fffff-aaaaa-eeee-sample",
    "keySpec": "AES_256"
  },
  "responseElements": null,
  "additionalEventData": {
    "keyMaterialId":
"145c963debe558dfb01848d2a4539da940f3478852f86cfe2f52d5df796a5a02"
  },
  "requestID": "cc9d1c5e-97c4-4a4f-ae7a-e576sample",
  "eventID": "0b0fef09-f28d-4da8-a5a1-17b74sample",
  "readOnly": true,
  "resources": [
    {
      "accountId": "account",
      "type": "AWS::KMS::Key",
      "ARN": "arn:aws:kms:region:accountId:key/ffffffff-fffff-aaaaa-eeee-sample"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "recipientAccountId": "accountId",
  "eventCategory": "Management"
}

```

Dekripsi

Contoh peristiwa untuk operasi Dekripsi saat EMR Tanpa Server mengakses data terenkripsi.

```

{
  "eventVersion": "1.11",
  "userIdentity": {

```

```

    "type": "AWSService",
    "principalId": "user",
    "invokedBy": "AWS Internal"
  },
  "eventTime": "2025-07-28T21:43:51Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "Decrypt",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "ipAddress",
  "userAgent": "userAgent",
  "requestParameters": {
    "encryptionContext": {
      "applicationId": "test"
    },
    "keyId": "arn:aws:kms:region:accountId:key/ffffffff-fffff-aaaaa-eeee-sample",
    "keySpec": "AES_256"
  },
  "responseElements": null,
  "additionalEventData": {
    "keyMaterialId":
"145c963debe558dfb01848d2a4539da940f3478852f86cfe2f52d5df796a5a02"
  },
  "requestID": "cc9d1c5e-97c4-4a4f-ae7a-e576sample",
  "eventID": "0b0fef09-f28d-4da8-a5a1-17b74sample",
  "readOnly": true,
  "resources": [
    {
      "accountId": "account",
      "type": "AWS::KMS::Key",
      "ARN": "arn:aws:kms:region:accountId:key/ffffffff-fffff-aaaaa-eeee-sample"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "recipientAccountId": "accountId",
  "eventCategory": "Management"
}

```

Pelajari Lebih Lanjut

Sumber daya berikut memberikan informasi lebih lanjut tentang enkripsi data saat istirahat.

- Untuk informasi selengkapnya tentang konsep AWS KMS dasar, lihat [Panduan AWS KMS Pengembang](#).
- Untuk informasi selengkapnya tentang praktik terbaik Keamanan AWS KMS, lihat [Panduan AWS KMS Pengembang](#).

Secrets Manager untuk perlindungan data dengan EMR Serverless

AWS Secrets Manager adalah layanan penyimpanan rahasia untuk melindungi kredensi database, kunci API, dan informasi rahasia lainnya. Kemudian dalam kode Anda, ganti kredensi hardcoded dengan panggilan API ke Secrets Manager. Ini membantu memastikan bahwa rahasia tidak dapat dikompromikan oleh seseorang yang memeriksa kode Anda, karena rahasianya tidak ada. Untuk ikhtisar, lihat [Panduan AWS Secrets Manager Pengguna](#).

Secrets Manager mengenkripsi rahasia menggunakan AWS Key Management Service kunci. Untuk informasi selengkapnya, lihat [Enkripsi rahasia dan dekripsi](#) di AWS Secrets Manager Panduan Pengguna.

Anda dapat mengonfigurasi Secrets Manager untuk secara otomatis memutar rahasia untuk Anda sesuai dengan jadwal yang Anda tentukan. Ini memungkinkan Anda mengganti rahasia jangka panjang dengan rahasia jangka pendek, yang membantu mengurangi risiko kompromi secara signifikan. Untuk informasi selengkapnya, lihat [Putar AWS Secrets Manager rahasia](#) di Panduan AWS Secrets Manager Pengguna.

Amazon EMR Serverless terintegrasi dengan AWS Secrets Manager sehingga Anda dapat menyimpan data Anda di Secrets Manager dan menggunakan ID rahasia dalam konfigurasi Anda.

Bagaimana EMR Serverless menggunakan rahasia

Ketika Anda menyimpan data Anda di Secrets Manager dan menggunakan ID rahasia dalam konfigurasi untuk EMR Serverless, Anda tidak meneruskan data konfigurasi sensitif ke EMR Serverless dalam teks biasa dan mengeksposnya ke eksternal. APIs Jika Anda menunjukkan bahwa pasangan kunci-nilai berisi ID rahasia untuk rahasia yang Anda simpan di Secrets Manager, EMR Serverless mengambil rahasia ketika mengirimkan data konfigurasi ke pekerja untuk menjalankan pekerjaan.

Untuk menunjukkan bahwa pasangan kunci-nilai untuk konfigurasi berisi referensi ke rahasia yang disimpan di Secrets Manager, tambahkan `EMR.secret@` anotasi ke nilai konfigurasi. Untuk properti

konfigurasi apa pun dengan anotasi Id rahasia, EMR Serverless memanggil Secrets Manager dan menyelesaikan rahasia pada saat eksekusi pekerjaan.

Cara membuat rahasia

Untuk membuat rahasia, ikuti langkah-langkah di [Buat AWS Secrets Manager rahasia](#) di Panduan AWS Secrets Manager Pengguna. Pada Langkah 3, pilih bidang Plaintext untuk memasukkan nilai sensitif Anda.

Berikan rahasia dalam klasifikasi konfigurasi

Contoh berikut menunjukkan bagaimana memberikan rahasia dalam klasifikasi konfigurasi di `startJobRun`. Jika Anda ingin mengonfigurasi klasifikasi untuk Secrets Manager di tingkat aplikasi, lihat [Konfigurasi aplikasi default untuk EMR Serverless](#)

Dalam contoh, ganti *SecretName* dengan nama rahasia untuk diambil. Untuk informasi lebih lanjut, lihat [Cara membuat rahasia](#).

Dalam bagian ini

- [Tentukan referensi rahasia - Spark](#)
- [Tentukan referensi rahasia - Sarang](#)

Tentukan referensi rahasia - Spark

Example— Tentukan referensi rahasia dalam konfigurasi metastore Hive eksternal untuk Spark

```
aws emr-serverless start-job-run \  
  --application-id "application-id" \  
  --execution-role-arn "job-role-arn" \  
  --job-driver '{  
    "sparkSubmit": {  
      "entryPoint": "s3://amzn-s3-demo-bucket/scripts/spark-jdbc.py",  
      "sparkSubmitParameters": "--jars s3://amzn-s3-demo-bucket/mariadb-  
connector-java.jar  
      --conf  
spark.hadoop.javax.jdo.option.ConnectionDriverName=org.mariadb.jdbc.Driver  
      --conf spark.hadoop.javax.jdo.option.ConnectionUserName=connection-user-  
name  
      --conf  
spark.hadoop.javax.jdo.option.ConnectionPassword=EMR.secret@SecretName
```

```

        --conf spark.hadoop.javax.jdo.option.ConnectionURL=jdbc:mysql://db-host:db-
port/db-name
        --conf spark.driver.cores=2
        --conf spark.executor.memory=10G
        --conf spark.driver.memory=6G
        --conf spark.executor.cores=4"
    }
}' \
--configuration-overrides '{
    "monitoringConfiguration": {
        "s3MonitoringConfiguration": {
            "logUri": "s3://amzn-s3-demo-bucket/spark/logs/"
        }
    }
}'
}'

```

Example— Tentukan referensi rahasia untuk konfigurasi metastore Hive eksternal dalam klasifikasi spark-defaults

```

{
    "classification": "spark-defaults",
    "properties": {

        "spark.hadoop.javax.jdo.option.ConnectionDriverName": "org.mariadb.jdbc.Driver"
        "spark.hadoop.javax.jdo.option.ConnectionURL": "jdbc:mysql://db-host:db-
port/db-name"
        "spark.hadoop.javax.jdo.option.ConnectionUserName": "connection-user-name"
        "spark.hadoop.javax.jdo.option.ConnectionPassword":
"EMR.secret@SecretName",
    }
}

```

Tentukan referensi rahasia - Sarang

Example— Tentukan referensi rahasia dalam konfigurasi metastore Hive eksternal untuk Hive

```

aws emr-serverless start-job-run \
  --application-id "application-id" \
  --execution-role-arn "job-role-arn" \
  --job-driver '{
    "hive": {
        "query": "s3://amzn-s3-demo-bucket/emr-serverless-hive/query/hive-query.q1",

```

```

    "parameters": "--hiveconf hive.exec.scratchdir=s3://amzn-s3-demo-bucket/emr-
serverless-hive/hive/scratch
                --hiveconf hive.metastore.warehouse.dir=s3://amzn-s3-demo-bucket/
emr-serverless-hive/hive/warehouse
                --hiveconf javax.jdo.option.ConnectionUserName=username
                --hiveconf
javax.jdo.option.ConnectionPassword=EMR.secret@SecretName
                --hiveconf
hive.metastore.client.factory.class=org.apache.hadoop.hive.q1.metadata.SessionHiveMetaStoreCli
                --hiveconf
javax.jdo.option.ConnectionDriverName=org.mariadb.jdbc.Driver
                --hiveconf javax.jdo.option.ConnectionURL=jdbc:mysql://db-host:db-
port/db-name"
    }
}' \
--configuration-overrides '{
  "monitoringConfiguration": {
    "s3MonitoringConfiguration": {
      "logUri": "s3://amzn-s3-demo-bucket"
    }
  }
}'

```

Example— Tentukan referensi rahasia untuk konfigurasi metastore Hive eksternal dalam klasifikasi hive-site

```

{
  "classification": "hive-site",
  "properties": {
    "hive.metastore.client.factory.class":
"org.apache.hadoop.hive.q1.metadata.SessionHiveMetaStoreClientFactory",
    "javax.jdo.option.ConnectionDriverName": "org.mariadb.jdbc.Driver",
    "javax.jdo.option.ConnectionURL": "jdbc:mysql://db-host:db-port/db-name",
    "javax.jdo.option.ConnectionUserName": "username",
    "javax.jdo.option.ConnectionPassword": "EMR.secret@SecretName"
  }
}

```

Berikan akses kepada EMR Serverless untuk mengambil rahasianya

Untuk memungkinkan EMR Tanpa Server mengambil nilai rahasia dari Secrets Manager, tambahkan pernyataan kebijakan berikut ke rahasia Anda saat Anda membuatnya. Anda harus membuat rahasia

Anda dengan kunci KMS yang dikelola pelanggan untuk EMR Tanpa Server untuk membaca nilai rahasia. Untuk informasi selengkapnya, lihat [Izin untuk kunci KMS](#) di AWS Secrets Manager Panduan Pengguna.

Dalam kebijakan berikut, ganti *applicationId* dengan ID untuk aplikasi Anda.

Kebijakan sumber daya untuk rahasia

Anda harus menyertakan izin berikut dalam kebijakan sumber daya untuk rahasia AWS Secrets Manager agar EMR Tanpa Server dapat mengambil nilai rahasia. Untuk memastikan bahwa hanya aplikasi tertentu yang dapat mengambil rahasia ini, Anda dapat secara opsional menentukan ID aplikasi EMR Tanpa Server sebagai syarat dalam kebijakan.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowSECRETSMANAGERGetsecretvalue",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "emr-serverless.amazonaws.com"
        ]
      },
      "Action": [
        "secretsmanager:GetSecretValue",
        "secretsmanager:DescribeSecret"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:SourceArn": "arn:aws:emr-serverless:*:123456789012:/applications/"
        }
      }
    }
  ]
}
```

Buat rahasia Anda dengan kebijakan berikut untuk kunci yang dikelola pelanggan AWS Key Management Service (AWS KMS):

Kebijakan untuk kunci yang dikelola pelanggan AWS KMS

```
{
  "Sid": "Allow EMR Serverless to use the key for decrypting secrets",
  "Effect": "Allow",
  "Principal": {
    "Service": [
      "emr-serverless.amazonaws.com"
    ]
  },
  "Action": [
    "kms:Decrypt",
    "kms:DescribeKey"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:ViaService": "secretsmanager.Wilayah AWS.amazonaws.com"
    }
  }
}
```

Memutar rahasianya

Rotasi adalah saat Anda memperbarui rahasia secara berkala. Anda dapat mengonfigurasi AWS Secrets Manager untuk secara otomatis memutar rahasia untuk Anda pada jadwal yang Anda tentukan. Dengan cara ini, Anda dapat mengganti rahasia jangka panjang dengan rahasia jangka pendek. Ini membantu mengurangi risiko kompromi. EMR Tanpa Server mengambil nilai rahasia dari konfigurasi berannotasi saat pekerjaan bertransisi ke status berjalan. Jika Anda atau proses memperbarui nilai rahasia di Secrets Manager, Anda harus mengirimkan pekerjaan baru sehingga pekerjaan dapat mengambil nilai yang diperbarui.

Note

Pekerjaan yang sudah dalam status berjalan tidak dapat mengambil nilai rahasia yang diperbarui. Hal ini dapat mengakibatkan kegagalan pekerjaan.

Menggunakan Hibah Akses Amazon S3 dengan EMR Tanpa Server

Ikhtisar Hibah Akses S3 untuk EMR Tanpa Server

Dengan Amazon EMR rilis 6.15.0 dan yang lebih tinggi, Amazon S3 Access Grants menyediakan solusi kontrol akses yang dapat diskalakan untuk menambah akses ke data Amazon S3 Anda dari EMR Tanpa Server. Jika Anda memiliki konfigurasi izin yang kompleks atau besar untuk data S3 Anda, gunakan Access Grants untuk menskalakan izin data S3 untuk pengguna, peran, dan aplikasi.

Gunakan Hibah Akses S3 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 aplikasi EMR Tanpa Server Anda.

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.

Bagian ini menjelaskan cara meluncurkan aplikasi EMR Tanpa Server yang menggunakan S3 Access Grants untuk menyediakan akses ke data di Amazon S3. Untuk langkah-langkah menggunakan Hibah Akses S3 dengan penerapan EMR Amazon lainnya, lihat dokumentasi berikut:

- [Menggunakan Hibah Akses S3 dengan Amazon EMR](#)
- [Menggunakan Hibah Akses S3 dengan Amazon EMR di EKS](#)

Luncurkan aplikasi EMR Tanpa Server dengan S3 Access Grants untuk manajemen data

Anda dapat mengaktifkan S3 Access Grants di EMR Serverless dan meluncurkan aplikasi 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 aplikasi EMR Tanpa Server Anda. Sertakan izin IAM yang diperlukan bahwa Anda harus menjalankan pekerjaan Spark dan menggunakan S3 Access Grants, 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 untuk eksekusi pekerjaan yang memiliki izin tambahan untuk mengakses S3 secara langsung, maka pengguna dapat mengakses data yang diizinkan oleh peran tersebut meskipun mereka tidak memiliki izin dari S3 Access Grants.

2. Luncurkan aplikasi EMR Tanpa Server Anda dengan label rilis EMR Amazon 6.15.0 atau lebih tinggi dan klasifikasi, seperti yang ditunjukkan contoh berikut `spark-defaults`. Ganti nilai *red text* dengan nilai yang sesuai untuk skenario penggunaan Anda.

```

aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "s3://us-east-1.elasticmapreduce/emr-containers/samples/
wordcount/scripts/wordcount.py",
      "entryPointArguments": ["s3://amzn-s3-demo-destination-bucket1/
wordcount_output"],
      "sparkSubmitParameters": "--conf spark.executor.cores=1 --conf
spark.executor.memory=4g --conf spark.driver.cores=1 --conf spark.driver.memory=4g
--conf spark.executor.instances=1"
    }
  }' \
  --configuration-overrides '{
    "applicationConfiguration": [{
      "classification": "spark-defaults",
      "properties": {
        "spark.hadoop.fs.s3.s3AccessGrants.enabled": "true",
        "spark.hadoop.fs.s3.s3AccessGrants.fallbackToIAM": "false"
      }
    }
  ]
}

```

```
}'
```

Akses S3 Memberikan pertimbangan dengan EMR Tanpa Server

Untuk informasi dukungan, kompatibilitas, dan perilaku penting saat Anda menggunakan Hibah Akses Amazon S3 dengan EMR Tanpa Server, lihat pertimbangan [Hibah Akses S3 dengan Amazon EMR di Panduan Manajemen EMR Amazon](#).

Mencatat panggilan API Tanpa Server Amazon EMR menggunakan AWS CloudTrail

Amazon EMR Serverless terintegrasi dengan AWS CloudTrail, layanan yang menyediakan catatan tindakan yang diambil oleh pengguna, peran, atau layanan AWS di EMR Tanpa Server. CloudTrail menangkap semua panggilan API untuk EMR Tanpa Server sebagai peristiwa. Panggilan yang diambil termasuk panggilan dari konsol EMR Tanpa Server dan panggilan kode ke operasi API EMR Tanpa Server. Jika Anda membuat jejak, aktifkan pengiriman CloudTrail acara secara terus menerus ke bucket Amazon S3, termasuk peristiwa untuk EMR Tanpa Server. Jika Anda tidak mengonfigurasi jejak, Anda masih dapat mengakses peristiwa terbaru di CloudTrail konsol dalam Riwayat acara. Dengan menggunakan informasi yang dikumpulkan oleh CloudTrail, Anda dapat menentukan permintaan yang dibuat untuk EMR Tanpa Server, alamat IP dari mana permintaan dibuat, siapa yang membuat permintaan, kapan dibuat, dan detail tambahan.

Untuk mempelajari lebih lanjut, lihat [Panduan AWS CloudTrail Pengguna](#).

EMR Informasi tanpa server di CloudTrail

CloudTrail diaktifkan pada Akun AWS saat Anda membuat akun. Ketika aktivitas terjadi di EMR Tanpa Server, aktivitas tersebut dicatat dalam suatu CloudTrail peristiwa bersama dengan peristiwa AWS layanan lainnya dalam riwayat Acara. Anda dapat mengakses, mencari, dan mengunduh acara terbaru di situs Anda Akun AWS. Untuk informasi selengkapnya, lihat [Melihat acara dengan riwayat CloudTrail Acara](#).

Untuk catatan acara yang sedang berlangsung di Anda Akun AWS, termasuk acara untuk EMR Tanpa Server, 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 Wilayah AWS. Jejak mencatat peristiwa dari semua Wilayah di AWS partisi dan mengirimkan file log ke bucket Amazon S3 yang Anda tentukan. Selain itu, konfigurasi layanan lain untuk

menganalisis lebih lanjut dan menindaklanjuti data peristiwa yang dikumpulkan dalam CloudTrail log. Untuk informasi lebih lanjut, lihat yang berikut ini:

- [Ikhtisar untuk membuat jejak](#)
- [CloudTrail layanan dan integrasi yang didukung](#)
- [Mengonfigurasi notifikasi Amazon SNS untuk CloudTrail](#)
- [Menerima file CloudTrail log dari beberapa wilayah](#) dan [Menerima file CloudTrail log dari beberapa akun](#)

[Semua tindakan EMR Tanpa Server dicatat oleh CloudTrail dan didokumentasikan dalam Referensi API EMR Tanpa Server](#). Misalnya, panggilan ke `CreateApplication`, `StartJobRun` dan `CancelJobRun` 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 lebih lanjut, lihat elemen [CloudTrail UserIdentity](#).

Memahami entri file log EMR Tanpa Server

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 `CreateApplication` tindakan.

```
{
  "eventVersion": "1.08",
  "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": {
        "creationDate": "2022-06-01T23:46:52Z",
        "mfaAuthenticated": "false"
      }
    }
  },
  "eventTime": "2022-06-01T23:49:28Z",
  "eventSource": "emr-serverless.amazonaws.com",
  "eventName": "CreateApplication",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "203.0.113.0",
  "userAgent": "PostmanRuntime/7.26.10",
  "requestParameters": {
    "name": "my-serverless-application",
    "releaseLabel": "emr-6.6",
    "type": "SPARK",
    "clientToken": "0a1b234c-de56-7890-1234-567890123456"
  },
  "responseElements": {
    "name": "my-serverless-application",
    "applicationId": "1234567890abcdef0",
    "arn": "arn:aws:emr-serverless:us-west-2:555555555555:/
applications/1234567890abcdef0"
  },
  "requestID": "890b8639-e51f-11e7-b038-EXAMPLE",
  "eventID": "874f89fa-70fc-4798-bc00-EXAMPLE",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "recipientAccountId": "012345678910",
```

```
"eventCategory": "Management"  
}
```

Validasi kepatuhan untuk Amazon EMR Tanpa Server

Keamanan dan kepatuhan EMR Tanpa Server dinilai oleh auditor pihak ketiga sebagai bagian dari beberapa program AWS kepatuhan, termasuk yang berikut:

- Kontrol Sistem dan Organisasi (SOC)
- Standar Keamanan Data Industri Kartu Pembayaran (PCI DSS)
- Program Manajemen Risiko dan Otorisasi Federal (FedRAMP) Moderat
- Undang-Undang Akuntabilitas dan Portabilitas Asuransi Kesehatan (HIPAA)

AWS menyediakan daftar AWS layanan yang sering diperbarui dalam lingkup program kepatuhan khusus di [AWS Layanan dalam Lingkup oleh Program Kepatuhan](#).

Laporan audit pihak ketiga tersedia untuk Anda unduh AWS Artifact. Untuk informasi selengkapnya, lihat [Mengunduh Laporan di AWS Artifact](#).

Untuk informasi selengkapnya tentang program AWS kepatuhan, lihat [Program AWS Kepatuhan](#).

Tanggung jawab kepatuhan Anda saat menggunakan EMR Tanpa Server ditentukan oleh sensitivitas data Anda, tujuan kepatuhan organisasi Anda, serta undang-undang dan peraturan yang berlaku. Jika penggunaan EMR Tanpa Server Anda tunduk pada kepatuhan dengan standar seperti HIPAA, PCI, atau FedRAMP Moderate, sediakan sumber daya untuk membantu: AWS

- [Panduan Memulai Cepat Keamanan dan Kepatuhan](#) yang membahas pertimbangan arsitektur dan langkah-langkah untuk menerapkan lingkungan dasar yang berfokus pada keamanan dan kepatuhan. AWS
- [AWS Panduan Kepatuhan Pelanggan](#) dapat membantu Anda memahami model tanggung jawab bersama melalui lensa kepatuhan. Panduan ini merangkum praktik terbaik untuk mengamankan Layanan AWS dan memetakan panduan kontrol keamanan di banyak kerangka kerja (termasuk National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), dan International Organization for Standardization (ISO)).
- [AWS Config](#) dapat digunakan untuk menilai seberapa baik konfigurasi sumber daya Anda telah mematuhi praktik internal, pedoman industri, dan peraturan yang berlaku.

- [AWS Sumber Daya Kepatuhan](#) adalah kumpulan buku kerja dan panduan yang mungkin berlaku untuk industri dan lokasi Anda.
- [AWS Security Hub](#) memberi Anda pandangan komprehensif tentang status keamanan Anda di dalamnya AWS dan membantu Anda memeriksa kepatuhan Anda terhadap standar industri keamanan dan praktik terbaik.
- [AWS Audit Manager](#) Ini Layanan AWS membantu Anda terus mengaudit AWS penggunaan Anda untuk menyederhanakan cara Anda mengelola risiko dan kepatuhan terhadap peraturan dan standar industri.

Ketahanan di Amazon EMR Tanpa Server

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 Zones, merancang dan mengoperasikan aplikasi dan database yang secara otomatis gagal di antara zona tanpa gangguan. Zona Ketersediaan memiliki ketersediaan dan toleransi kesalahan yang lebih baik, dan dapat diskalakan dibandingkan infrastruktur biasa yang terdiri dari satu atau beberapa pusat data.

Untuk informasi selengkapnya tentang AWS Wilayah dan Availability Zone, lihat [Infrastruktur AWS Global](#).

Selain infrastruktur AWS global, Amazon EMR Serverless menawarkan integrasi dengan Amazon S3 melalui EMRFS untuk membantu mendukung ketahanan data dan kebutuhan pencadangan Anda.

Keamanan infrastruktur di Amazon EMR Tanpa Server

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.

Analisis konfigurasi dan kerentanan di Amazon EMR Tanpa Server

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 Tanpa Server](#)
- [Model Tanggung Jawab Bersama](#)
- [Amazon Web Services: Gambaran Umum Proses Keamanan](#)

Titik akhir dan kuota untuk EMR Serverless

Titik akhir layanan

Untuk terhubung secara terprogram ke sebuah Layanan AWS, Anda menggunakan endpoint. Endpoint adalah URL titik masuk untuk layanan AWS web. Selain titik akhir standar, beberapa Layanan AWS menawarkan AWS titik akhir FIPS di Wilayah tertentu. Tabel berikut mencantumkan endpoint layanan untuk EMR Tanpa Server. Untuk informasi lebih lanjut, lihat [Layanan AWS titik akhir](#).

EMR Titik akhir layanan tanpa server

Nama wilayah	Wilayah	Titik Akhir	Protokol
AS Timur (Ohio)	us-east-2 (terbatas pada Availability Zone berikut: use2-az1, use2-az2, dan use2-az3)	emr-serverless.us-east-2.amazonaws.com	HTTPS
AS Timur (Virginia Utara)	us-east-1 (terbatas pada Availability Zone berikut: use1-az1, use1-az2, use1-az4, use1-az5, dan use1-az6)	emr-serverless.us-east-1.amazonaws.com emr-serverless-fips.us-east-1.amazonaws.com	HTTPS
AS Barat (California Utara)	us-west-1	emr-serverless.us-west-1.amazonaws.com	HTTPS
AS Barat (Oregon)	us-west-2	emr-serverless.us-	HTTPS

Nama wilayah	Wilayah	Titik Akhir	Protokol
		west-2.amazonaws.com emr-serverless-fips.us-west-2.amazonaws.com	
Africa (Cape Town)	af-south-1	emr-serverless.af-south-1.amazonaws.com	HTTPS
Asia Pasifik (Hong Kong)	ap-east-1	emr-serverless.ap-east-1.amazonaws.com	HTTPS
Asia Pasifik (Jakarta)	ap-southeast-3	emr-serverless.ap-southeast-3.amazonaws.com	HTTPS
Asia Pacific (Melbourne)	ap-southeast-4	emr-serverless.ap-southeast-4.amazonaws.com	HTTPS
Asia Pasifik (Malaysia)	ap-southeast-5	emr-serverless.ap-southeast-5.amazonaws.com	HTTPS

Nama wilayah	Wilayah	Titik Akhir	Protokol
Asia Pasifik (Mumbai)	ap-south-1	emr-serverless.ap-south-1.amazonaws.com	HTTPS
Asia Pasifik (Osaka)	ap-northeast-3	emr-serverless.ap-northeast-3.amazonaws.com	HTTPS
Asia Pasifik (Seoul)	ap-northeast-2	emr-serverless.ap-northeast-2.amazonaws.com	HTTPS
Asia Pasifik (Singapura)	ap-southeast-1	emr-serverless.ap-southeast-1.amazonaws.com	HTTPS
Asia Pasifik (Sydney)	ap-southeast-2	emr-serverless.ap-southeast-2.amazonaws.com	HTTPS
Asia Pasifik (Tokyo)	ap-northeast-1	emr-serverless.ap-northeast-1.amazonaws.com	HTTPS

Nama wilayah	Wilayah	Titik Akhir	Protokol
Kanada (Pusat)	ca-central-1-1 (terbatas pada Availability Zone berikut: cac1-az1 dan cac1-az2)	emr-serverless.ca-central-1.amazonaws.com	HTTPS
Kanada Barat (Calgary)	ca-west-1	emr-serverless.ca-west-1.amazonaws.com	HTTPS
Eropa (Frankfurt)	eu-central-1	emr-serverless.eu-central-1.amazonaws.com	HTTPS
Europe (Zurich)	eu-central-2	emr-serverless.eu-central-2.amazonaws.com	HTTPS
Eropa (Irlandia)	eu-west-1	emr-serverless.eu-west-1.amazonaws.com	HTTPS
Eropa (London)	eu-west-2	emr-serverless.eu-west-2.amazonaws.com	HTTPS
Europe (Milan)	eu-south-1	emr-serverless.eu-south-1.amazonaws.com	HTTPS

Nama wilayah	Wilayah	Titik Akhir	Protokol
Eropa (Paris)	eu-west-3	emr-serve rless.eu- west-3.am azonaws.com	HTTPS
Eropa (Spanyol)	eu-south-2	emr-serve rless.eu- south-2.a mazonaws.com	HTTPS
Eropa (Stockholm)	eu-north-1	emr-serve rless.eu- north-1.a mazonaws.com	HTTPS
Israel (Tel Aviv)	il-central-1	emr-serve rless.il- central-1 .amazonaws.com	HTTPS
Timur Tengah (Bahrain)	me-south-1	emr-serve rless.me- south-1.a mazonaws.com	HTTPS
Timur Tengah (UAE)	me-central-1	emr-serve rless.me- central-1 .amazonaws.com	HTTPS
Amerika Selatan (Sao Paulo)	sa-east-1	emr-serve rless.sa- east-1.am azonaws.com	HTTPS

Nama wilayah	Wilayah	Titik Akhir	Protokol
Tiongkok (Beijing)	cn-north-1 (terbatas pada Availability Zone berikut:cn1-az1,cnn1-az2)	emr-serve rless.cn- north-1.a mazonaws. com.cn	HTTPS
AWS GovCloud (AS-Timur)	us-gov-east-1	emr-serve rless.us-gov- east-1.amazona ws.com	HTTPS
AWS GovCloud (AS-Barat)	us-gov-west-1	emr-serve rless.us-gov- west-1.amazona ws.com	HTTPS

Kuota layanan

Kuota layanan, juga dikenal sebagai batas, adalah jumlah maksimum sumber daya layanan atau operasi yang Akun AWS dapat Anda gunakan. EMR Tanpa Server mengumpulkan metrik penggunaan kuota layanan setiap menit dan menerbitkannya di namespace. `AWS/Usage`

Note

AWS Akun baru memiliki kuota awal yang lebih rendah yang dapat meningkat seiring waktu. Amazon EMR Tanpa Server memantau penggunaan akun di masing-masing akun Wilayah AWS, dan kemudian secara otomatis meningkatkan kuota berdasarkan penggunaan Anda.

Tabel berikut mencantumkan kuota layanan untuk EMR Tanpa Server. Untuk informasi lebih lanjut, lihat [Layanan AWS kuota](#).

Nama	Batas default	Dapat disesuaikan?	Deskripsi
Maks bersamaan v CPUs per akun	16	Ya	Jumlah maksimum v CPUs yang dapat berjalan secara bersamaan untuk akun saat ini Wilayah AWS.

Batas API

Berikut ini menjelaskan batas API per Wilayah untuk Anda Akun AWS.

Sumber daya	Kuota bawaan
ListApplications	10 transaksi per detik. Burst 50 transaksi per detik.
CreateApplication	1 transaksi per detik. Burst 25 transaksi per detik.
DeleteApplication	1 transaksi per detik. Burst 25 transaksi per detik.
GetApplication	10 transaksi per detik. Burst 50 transaksi per detik.
UpdateApplication	1 transaksi per detik. Burst 25 transaksi per detik.
ListJobRuns	1 transaksi per detik. Burst 25 transaksi per detik.
StartJobRun	1 transaksi per detik. Burst 25 transaksi per detik.

Sumber daya	Kuota bawaan
GetDashboardForJobRun	1 transaksi per detik. Burst 2 transaksi per detik.
CancelJobRun	1 transaksi per detik. Burst 25 transaksi per detik.
GetJobRun	10 transaksi per detik. Burst 50 transaksi per detik.
StartApplication	1 transaksi per detik. Burst 25 transaksi per detik.
StopApplication	1 transaksi per detik. Burst 25 transaksi per detik.

Pertimbangan lainnya

Daftar berikut berisi pertimbangan lain dengan EMR Serverless.

- EMR Tanpa Server tersedia dalam hal berikut: Wilayah AWS
 - AS Timur (Ohio)
 - AS Timur (Virginia Utara)
 - AS Barat (California Utara)
 - AS Barat (Oregon)
 - Afrika (Cape Town)
 - Asia Pasifik (Hong Kong)
 - Asia Pasifik (Taipei)
 - Asia Pasifik (Jakarta)
 - Asia Pasifik (Mumbai)
 - Asia Pasifik (Hyderabad)
 - Asia Pasifik (Osaka)
 - Asia Pasifik (Seoul)
 - Asia Pasifik (Singapura)
 - Asia Pasifik (Sydney)
 - Asia Pasifik (Malaysia)
 - Asia Pasifik (Selandia Baru)
 - Asia Pasifik (Thailand)
 - Asia Pasifik (Tokyo)
 - Kanada (Pusat)
 - Eropa (Frankfurt)
 - Eropa (Irlandia)
 - Eropa (London)
 - Eropa (Milan)
 - Eropa (Paris)
 - Eropa (Spanyol)
 - Eropa (Stockholm)

- Timur Tengah (Bahrain)
- Middle East (UAE)
- Meksiko (Tengah)
- Amerika Selatan (Sao Paulo)
- AWS GovCloud (US-East)
- AWS GovCloud (US-West)

Untuk daftar titik akhir yang terkait dengan Wilayah ini, lihat. [Titik akhir layanan](#)

- Batas waktu default untuk menjalankan pekerjaan adalah 12 jam. Anda dapat mengubah setelan ini dengan `executionTimeoutMinutes` properti di `startJobRun` API atau AWS SDK. Anda dapat mengatur `executionTimeoutMinutes` ke 0 jika Anda ingin pekerjaan Anda tidak pernah habis. Misalnya, jika Anda memiliki aplikasi streaming, atur `executionTimeoutMinutes` ke 0 untuk memungkinkan pekerjaan streaming berjalan terus menerus.
- `billedResourceUtilization` Properti di `getJobRun` API menunjukkan vCPU agregat, memori, dan penyimpanan AWS yang telah ditagih untuk menjalankan pekerjaan. Sumber daya yang ditagih mencakup penggunaan minimum 1 menit untuk pekerja, ditambah penyimpanan tambahan lebih dari 20 GB per pekerja. Sumber daya ini tidak termasuk penggunaan untuk pekerja pra-inisialisasi yang menganggur.
- Tanpa konektivitas VPC, pekerjaan dapat mengakses beberapa Layanan AWS titik akhir dalam hal yang sama. Wilayah AWS Layanan ini termasuk Amazon S3, AWS Glue, AWS Lake Formation, Amazon CloudWatch Logs,, AWS KMS, AWS Security Token Service Amazon DynamoDB, dan. AWS Secrets Manager Anda dapat mengaktifkan konektivitas VPC untuk mengakses lain Layanan AWS melalui [AWS PrivateLink](#), tetapi Anda tidak diharuskan untuk melakukan ini. Untuk mengakses layanan eksternal, buat aplikasi Anda dengan VPC.
- EMR Serverless tidak mendukung HDFS. Disk lokal pada pekerja adalah penyimpanan sementara yang digunakan EMR Serverless untuk mengacak dan memproses data selama pekerjaan berjalan.

Amazon EMR Versi rilis tanpa server

Rilis EMR Amazon adalah seperangkat aplikasi open source dari ekosistem big data. Setiap rilis menyertakan aplikasi data besar, komponen, dan fitur yang Anda pilih agar Amazon EMR Serverless disebarkan dan dikonfigurasi saat menjalankan pekerjaan.

Dengan Amazon EMR 6.6.0 dan yang lebih tinggi, terapkan EMR Tanpa Server. Opsi penerapan ini tidak tersedia dengan versi rilis Amazon EMR sebelumnya. Saat Anda mengirimkan pekerjaan, tentukan salah satu rilis yang didukung berikut ini.

Topik

- [Waktu proses AWS untuk Apache Spark \(emr-spark-8.0.0\)](#)
- [AWS runtime untuk Apache Spark \(emr-spark-8.0-preview\)](#)
- [EMR Tanpa Server 7.13.0](#)
- [EMR Tanpa Server 7.12.0](#)
- [EMR Tanpa Server 7.11.0](#)
- [EMR Tanpa Server 7.10.0](#)
- [EMR Tanpa Server 7.9.0](#)
- [EMR Tanpa Server 7.8.0](#)
- [EMR Tanpa Server 7.7.0](#)
- [EMR Tanpa Server 7.6.0](#)
- [EMR Tanpa Server 7.5.0](#)
- [EMR Tanpa Server 7.4.0](#)
- [EMR Tanpa Server 7.3.0](#)
- [EMR Tanpa Server 7.2.0](#)
- [EMR Tanpa Server 7.1.0](#)
- [EMR Tanpa Server 7.0.0](#)
- [EMR Tanpa Server 6.15.0](#)
- [EMR Tanpa Server 6.14.0](#)
- [EMR Tanpa Server 6.13.0](#)
- [EMR Tanpa Server 6.12.0](#)

- [EMR Tanpa Server 6.11.0](#)
- [EMR Tanpa Server 6.10.0](#)
- [EMR Tanpa Server 6.9.0](#)
- [EMR Tanpa Server 6.8.0](#)
- [EMR Tanpa Server 6.7.0](#)
- [EMR Tanpa Server 6.6.0](#)

Waktu proses AWS untuk Apache Spark (emr-spark-8.0.0)

Tabel berikut mencantumkan versi aplikasi yang tersedia dengan AWS runtime for Apache Spark (emr-spark-8.0.0).

Informasi versi aplikasi

Aplikasi	Versi
Spark	4.0.2-amzn-0
Gunung es	1.10.1-amzn-0
kuala	4.0.0-amzn-1-spark
Hudi	1.1.0-amzn-0

Catatan rilis AWS runtime untuk Apache Spark (emr-spark-8.0.0)

- Rilis GA - Ini adalah rilis ketersediaan umum AWS runtime for Apache Spark yang menampilkan Apache Spark 4.0.2. Rilis ini tersedia di EMR Tanpa Server, EMR di EC2, dan EMR di EKS.
- Ketersediaan Regional - Tersedia di semua AWS Wilayah di mana EMR Tanpa Server tersedia, kecuali wilayah Timur Tengah (Bahrain) dan Timur Tengah (UEA).
- Batasan yang diketahui - Titik akhir aman Spark Connect dengan dukungan FGAC Asli tidak tersedia dalam rilis ini.
- Dokumentasi Tambahan - Untuk dokumentasi Apache Spark tambahan, lihat Dokumentasi Rilis [Apache Spark 4.0.2](#).

Memulai

Untuk memulai dengan Apache Spark 4.0.2, buat aplikasi EMR Tanpa Server menggunakan CLI: AWS

```
aws emr-serverless create-application --type SPARK \  
  --release-label emr-spark-8.0.0 \  
  --name spark4-serverless \  
  --region us-east-1
```

Catatan

- Rilis ini menggantikan rilis pratinjau (emr-spark-8.0-preview). Pratinjau terbatas pada Spark 4.0.1 dan tidak memiliki FGAC, Hudi, konektor data, dan Persistent Spark History Server.
- `--type` parameter untuk `create-application` penggunaan SPARK (huruf besar).

AWS runtime untuk Apache Spark (emr-spark-8.0-preview)

Tabel berikut mencantumkan versi aplikasi yang tersedia dengan AWS runtime for Apache Spark (emr-spark-8.0-preview).

Informasi versi aplikasi

Aplikasi	Versi
Spark	4.0.1-amzn-0

Catatan rilis AWS runtime untuk Apache Spark (emr-spark-8.0-preview)

- Rilis pratinjau - Ini adalah rilis pratinjau yang AWS runtime for Apache Spark menampilkan Apache Spark 4.0.1. Pratinjau ini hanya tersedia di EMR Tanpa Server.
- Ketersediaan Regional - Rilis pratinjau ini tersedia di semua AWS Wilayah di mana EMR Tanpa Server tersedia, kecuali wilayah China dan AWS GovCloud (AS).
- Informasi versi aplikasi - Rilis ini dikirimkan dengan versi aplikasi berikut:
 - AWS SDK for Java 2.35.5, 1.12.792

- Python 3.9, 3.11, 3.12
- Scala 2.13.16
- AmazonCloudWatchAgent 1.300034.0-amzn-0
- Delta 4.0.0-amzn-0-spark
- Gunung es 1.10.0-amzn-spark-0
- Rilis ini dikirimkan dengan Amazon Corretto 17 (dibangun di atas OpenJDK) secara default untuk aplikasi yang mendukung Corretto 17 (JDK 17).
- Batasan pratinjau - Kemampuan berikut tidak tersedia dalam rilis pratinjau ini:
 - Fitur Interaktif dan Integrasi: SageMaker Unified Studio, integrasi EMR Studio, Spark Connect, Livy, JupyterEnterpriseGateway dan tidak didukung.
 - Format Tabel dan Kontrol Akses: Hudi, Delta Universal Format, dan fine-grained access control (FGAC) dengan pemfilteran tingkat baris atau tingkat kolom dan operator tidak didukung. DDL/DML
 - Konektor Data: konektor spark-sql-kinesis, emr-dynamodb, dan spark-redshift tidak tersedia.
 - Server Sejarah: Server Sejarah Persisten Spark tidak tersedia dalam rilis pratinjau ini. Pengguna masih dapat mengakses UI Spark langsung untuk memantau dan men-debug pekerjaan tanpa server aktif secara real-time.
 - Fitur Khusus: Tampilan Terwujud tidak tersedia.
- Kemampuan pratinjau - Anda dapat menguji kemampuan berikut dalam rilis pratinjau ini. Rilis pratinjau ini tidak disarankan untuk beban kerja produksi:
 - Fitur SQL: Mode ANSI SQL dengan penanganan tipe yang lebih ketat, sintaks SQL PIPE (|>) untuk operasi rantai, tipe data VARIANT untuk data JSON semi-terstruktur, skrip SQL dengan pernyataan aliran kontrol dan variabel sesi, dan fungsi yang ditentukan pengguna SQL.
 - Penyempurnaan Streaming: Arbitrary Stateful Processing API v2 dengan WithState operator transformasi, Pembaca Sumber Data Status untuk status streaming yang dapat dikueri (eksperimental), dan penyimpanan status yang disempurnakan dengan checkpointing log perubahan RocksDB yang ditingkatkan.
 - Dukungan Format Tabel: Apache Iceberg v3 dengan dukungan tipe data VARIANT, integrasi Tabel AWS S3, dan Akses Tabel Penuh (FTA) dengan tabel AWS Lake Formation untuk Iceberg, Delta Lake, dan Hive.
- Dokumentasi Tambahan - Untuk dokumentasi Apache Spark tambahan, lihat Dokumentasi Rilis [Apache Spark 4.0.1](#).

Memulai

Untuk memulai pratinjau Apache Spark 4.0.1, buat aplikasi EMR Tanpa Server menggunakan CLI: AWS

```
aws emr-serverless create-application --type spark \  
--release-label emr-spark-8.0-preview \  
--region us-east-1 --name spark4-preview
```

EMR Tanpa Server 7.13.0

Tabel berikut mencantumkan versi aplikasi yang tersedia dengan EMR Serverless 7.13.0.

Aplikasi	Versi
Apache Spark	3.5.6
Apache Hive	3.1.3
Apache Tez	0.10.2

Catatan rilis EMR Serverless 7.13.0

- Fitur baru
 - Spark Connect untuk PySpark sesi interaktif — EMR Serverless sekarang mendukung sesi PySpark interaktif melalui Apache Spark Connect. Dengan Amazon EMR rilis 7.13.0 dan yang lebih baru, Anda dapat terhubung ke Spark yang berjalan di EMR Tanpa Server dari klien PySpark lokal, termasuk IDE seperti VS Code,, dan notebook Jupyter. PyCharm Untuk informasi selengkapnya, lihat [Jalankan sesi interaktif dengan Amazon EMR Tanpa Server melalui Spark Connect](#).

EMR Tanpa Server 7.12.0

Tabel berikut mencantumkan versi aplikasi yang tersedia dengan EMR Serverless 7.12.0.

Aplikasi	Versi
Apache Spark	3.5.6
Apache Hive	3.1.3
Apache Tez	0.10.2

Catatan rilis EMR Serverless 7.12.0

- **Fitur baru**
 - **Penyimpanan tanpa server untuk EMR Tanpa Server** - Amazon EMR tanpa server memperkenalkan penyimpanan tanpa server, dengan rilis EMR 7.12 dan yang lebih baru, yang menghilangkan penyediaan disk lokal untuk beban kerja Apache Spark. EMR Serverless secara otomatis menangani operasi data perantara seperti shuffle tanpa biaya penyimpanan. Penyimpanan tanpa server memisahkan penyimpanan dari komputasi, memungkinkan Spark untuk segera melepaskan pekerja saat idle daripada membuat mereka tetap aktif untuk menyimpan data sementara. Untuk mempelajari lebih lanjut, lihat [Penyimpanan tanpa server](#).
 - **Iceberg Materialized Views** - Memulai Amazon EMR 7.12.0, Amazon EMR Spark mendukung pembuatan dan pengelolaan Iceberg Materialized Views (MV)
 - **Akses Tabel Penuh Hudi** - Memulai Amazon EMR 7.12.0, Amazon 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.
 - **Peningkatan versi Iceberg** - Amazon EMR 7.12.0 mendukung Apache Iceberg versi 1.10

EMR Tanpa Server 7.11.0

Tabel berikut mencantumkan versi aplikasi yang tersedia dengan EMR Serverless 7.11.0.

Aplikasi	Versi
Apache Spark	3.5.6

Aplikasi	Versi
Apache Hive	3.1.3
Apache Tez	0.10.2

Catatan rilis EMR Serverless 7.11.0

- Waktu eksekusi Job Maksimum — Nilai maksimum untuk `executionTimeoutMinutes` in `StartJobRun` action untuk pekerjaan BATCH adalah 7 hari sejak rilis ini dan seterusnya. `executionTimeoutMinutes` tidak dapat lagi disetel ke 0 yaitu tidak ada batas waktu, untuk pekerjaan batch berjalan.

EMR Tanpa Server 7.10.0

Tabel berikut mencantumkan versi aplikasi yang tersedia dengan EMR Serverless 7.10.0.

Aplikasi	Versi
Apache Spark	3.5.5
Apache Hive	3.1.3
Apache Tez	0.10.2

Catatan rilis EMR Serverless 7.10.0

- Metrik untuk EMR Tanpa Server — Metrik pemantauan direstrukturisasi untuk fokus pada dimensi dan dimensi. `ApplicationName` `JobName` Metrik lama tidak akan lagi diperbarui, ke depan. Untuk informasi lebih lanjut, lihat [Memantau aplikasi dan pekerjaan tanpa server EMR](#).

EMR Tanpa Server 7.9.0

Tabel berikut mencantumkan versi aplikasi yang tersedia dengan EMR Serverless 7.9.0.

Aplikasi	Versi
Apache Spark	3.5.5
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Tanpa Server 7.8.0

Tabel berikut mencantumkan versi aplikasi yang tersedia dengan EMR Serverless 7.8.0.

Aplikasi	Versi
Apache Spark	3.5.4
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Tanpa Server 7.7.0

Tabel berikut mencantumkan versi aplikasi yang tersedia dengan EMR Serverless 7.7.0.

Aplikasi	Versi
Apache Spark	3.5.3
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Tanpa Server 7.6.0

Tabel berikut mencantumkan versi aplikasi yang tersedia dengan EMR Serverless 7.6.0.

Aplikasi	Versi
Apache Spark	3.5.3
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Tanpa Server 7.5.0

Tabel berikut mencantumkan versi aplikasi yang tersedia dengan EMR Serverless 7.5.0.

Aplikasi	Versi
Apache Spark	3.5.2
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Tanpa Server 7.4.0

Tabel berikut mencantumkan versi aplikasi yang tersedia dengan EMR Serverless 7.4.0.

Aplikasi	Versi
Apache Spark	3.5.2
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Tanpa Server 7.3.0

Tabel berikut mencantumkan versi aplikasi yang tersedia dengan EMR Serverless 7.3.0.

Aplikasi	Versi
Apache Spark	3.5.1
Apache Hive	3.1.3
Apache Tez	0.10.2

Catatan rilis EMR Serverless 7.3.0

- **Konkurensi pekerjaan dan antrian dengan EMR Tanpa Server** — Konkurensi dan antrian Job diaktifkan secara default saat Anda membuat aplikasi EMR Tanpa Server baru di Amazon EMR rilis 7.3.0 atau lebih tinggi. Untuk informasi lebih lanjut, lihat [the section called “Konkurensi dan antrian pekerjaan”](#), yang merinci bagaimana memulai dengan konkurensi dan antrian dan juga berisi daftar pertimbangan fitur.

EMR Tanpa Server 7.2.0

Tabel berikut mencantumkan versi aplikasi yang tersedia dengan EMR Serverless 7.2.0.

Aplikasi	Versi
Apache Spark	3.5.1
Apache Hive	3.1.3
Apache Tez	0.10.2

Catatan rilis EMR Serverless 7.2.0

- **Lake Formation dengan EMR Tanpa Server** — Anda sekarang dapat menggunakan AWS Lake Formation untuk menerapkan kontrol akses berbutir halus pada tabel Katalog Data yang didukung oleh S3. Kemampuan ini memungkinkan Anda mengonfigurasi kontrol akses tingkat tabel, baris, kolom, dan sel untuk kueri baca dalam pekerjaan EMR Serverless Spark Anda. Untuk informasi lebih lanjut, lihat [the section called “Lake Formation untuk FGAC”](#) dan [the section called “Pertimbangan dan batasan”](#).

EMR Tanpa Server 7.1.0

Tabel berikut mencantumkan versi aplikasi yang tersedia dengan EMR Serverless 7.1.0.

Aplikasi	Versi
Apache Spark	3.5.0
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Tanpa Server 7.0.0

Tabel berikut mencantumkan versi aplikasi yang tersedia dengan EMR Serverless 7.0.0.

Aplikasi	Versi
Apache Spark	3.5.0
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Tanpa Server 6.15.0

Tabel berikut mencantumkan versi aplikasi yang tersedia dengan EMR Serverless 6.15.0.

Aplikasi	Versi
Apache Spark	3.4.1
Apache Hive	3.1.3
Apache Tez	0.10.2

Catatan rilis EMR Serverless 6.15.0

- Dukungan TLS - Dengan Amazon EMR Serverless rilis 6.15.0 dan yang lebih tinggi, aktifkan komunikasi terenkripsi TLS timbal-TLS antara pekerja dalam pekerjaan Spark Anda berjalan. Saat diaktifkan, EMR Serverless secara otomatis menghasilkan sertifikat unik untuk setiap pekerja yang disediakan di bawah pekerjaan yang digunakan pekerja selama jabat tangan TLS untuk mengautentikasi satu sama lain dan membuat saluran terenkripsi untuk memproses data dengan aman. [Untuk informasi lebih lanjut tentang enkripsi TLS timbal-balik, lihat enkripsi. Inter-worker](#)

EMR Tanpa Server 6.14.0

Tabel berikut mencantumkan versi aplikasi yang tersedia dengan EMR Serverless 6.14.0.

Aplikasi	Versi
Apache Spark	3.4.1
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Tanpa Server 6.13.0

Tabel berikut mencantumkan versi aplikasi yang tersedia dengan EMR Serverless 6.13.0.

Aplikasi	Versi
Apache Spark	3.4.1
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Tanpa Server 6.12.0

Tabel berikut mencantumkan versi aplikasi yang tersedia dengan EMR Serverless 6.12.0.

Aplikasi	Versi
Apache Spark	3.4.0
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Tanpa Server 6.11.0

Tabel berikut mencantumkan versi aplikasi yang tersedia dengan EMR Serverless 6.11.0.

Aplikasi	Versi
Apache Spark	3.3.2
Apache Hive	3.1.3
Apache Tez	0.10.2

Catatan rilis EMR Serverless 6.11.0

- [Akses sumber daya S3 di akun lain](#) - Dengan rilis 6.11.0 dan yang lebih tinggi, Anda dapat mengonfigurasi beberapa peran IAM untuk diasumsikan saat mengakses bucket Amazon S3 di akun berbeda dari EMR Tanpa Server. AWS

EMR Tanpa Server 6.10.0

Tabel berikut mencantumkan versi aplikasi yang tersedia dengan EMR Serverless 6.10.0.

Aplikasi	Versi
Apache Spark	3.3.1
Apache Hive	3.1.3
Apache Tez	0.10.2

Catatan rilis EMR Serverless 6.10.0

- Untuk aplikasi EMR Tanpa Server dengan rilis 6.10.0 atau lebih tinggi, nilai default untuk properti adalah `spark.dynamicAllocation.maxExecutors infinity` Sebelumnya rilis default ke100. Untuk informasi lebih lanjut, lihat [Properti pekerjaan percikan](#).

EMR Tanpa Server 6.9.0

Tabel berikut mencantumkan versi aplikasi yang tersedia dengan EMR Serverless 6.9.0.

Aplikasi	Versi
Apache Spark	3.3.0
Apache Hive	3.1.3
Apache Tez	0.10.2

Catatan rilis EMR Serverless 6.9.0

- 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 Tanpa Server](#).
- Rilis EMR Tanpa Server 6.9.0 menambahkan dukungan untuk AWS arsitektur Graviton2 (arm64). Anda dapat menggunakan `architecture` parameter untuk `update-application` API `create-application` dan untuk memilih arsitektur arm64. Untuk informasi lebih lanjut, lihat [Amazon EMR Opsi arsitektur tanpa server](#).
- Anda sekarang dapat mengeksport, mengimpor, menanyakan, dan bergabung dengan tabel Amazon DynamoDB langsung dari aplikasi EMR Serverless Spark dan Hive Anda. Untuk informasi lebih lanjut, lihat [Menghubungkan ke DynamoDB dengan Amazon EMR Tanpa Server](#).

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 Tanpa Server 6.8.0

Tabel berikut mencantumkan versi aplikasi yang tersedia dengan EMR Serverless 6.8.0.

Aplikasi	Versi
Apache Spark	3.3.0
Apache Hive	3.1.3
Apache Tez	0.9.2

EMR Tanpa Server 6.7.0

Tabel berikut mencantumkan versi aplikasi yang tersedia dengan EMR Serverless 6.7.0.

Aplikasi	Versi
Apache Spark	3.2.1
Apache Hive	3.1.3
Apache Tez	0.9.2

Engine-specific perubahan, penyempurnaan, dan masalah yang diselesaikan

Tabel berikut mencantumkan fitur khusus mesin baru.

Ubah	Deskripsi
Fitur	Penjadwal Tez sekarang mendukung preemption tugas Tez alih-alih preemption wadah

EMR Tanpa Server 6.6.0

Tabel berikut mencantumkan versi aplikasi yang tersedia dengan EMR Serverless 6.6.0.

Aplikasi	Versi
Apache Spark	3.2.0
Apache Hive	3.1.2
Apache Tez	0.9.2

Catatan rilis awal EMR Tanpa Server

- EMR Serverless mendukung klasifikasi konfigurasi Spark. `spark-defaults` Klasifikasi ini mengubah nilai dalam file `spark-defaults.conf` XML's Spark. Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Untuk informasi selengkapnya, lihat [Konfigurasi aplikasi](#).
- EMR Tanpa Server mendukung klasifikasi `hive-site` konfigurasi Hive,,, dan `tez-site` `emrfs-site` `core-site` Klasifikasi ini dapat mengubah nilai dalam file Hive, `hive-site.xml` file Tez, pengaturan EMRFS Amazon EMR, atau `tez-site.xml` file Hadoop, masing-masing. `core-site.xml` Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Untuk informasi selengkapnya, lihat [Konfigurasi aplikasi](#).

Engine-specific perubahan, penyempurnaan, dan masalah yang diselesaikan

- Tabel berikut mencantumkan Hive dan Tez backports.

Hive dan Tez berubah

Ubah	Deskripsi
Backport	TEZ-4430 : Memperbaiki masalah dengan <code>tez.task.launch.cmd-opts</code> properti
Backport	HIVE-25971 : Memperbaiki penundaan shutdown tugas Tez karena membuka kumpulan utas yang di-cache

Riwayat dokumen

Tabel berikut menjelaskan perubahan penting pada dokumentasi sejak rilis terakhir EMR Tanpa Server. Untuk informasi lebih lanjut tentang pembaruan dokumentasi ini, Anda dapat berlangganan umpan RSS.

Perubahan	Deskripsi	Tanggal
Rilis GA baru	Waktu proses AWS untuk Apache Spark (emr-spark-8.0.0)	21 Mei 2026
Fitur baru	Jalankan sesi interaktif dengan Spark Connect	April 23, 2026
Rilis pratinjau baru	AWS runtime untuk Apache Spark (emr-spark-8.0-preview)	November 21, 2025
Rilis baru	EMR Tanpa Server 7.2.0	Juli 25, 2024
Rilis baru	EMR Tanpa Server 7.1.0	April 17, 2024
Perbarui ke kebijakan yang ada.	Menambahkan yang baru Sid CloudWatchPolicyStatement dan EC2PolicyStatement ke AmazonEMRServerlessServiceRolePolicy kebijakan.	Januari 25, 2024
Rilis baru	EMR Tanpa Server 7.0.0	Desember 29, 2023
Rilis baru	EMR Tanpa Server 6.15.0	17 November 2023
Fitur baru	Konfigurasi beberapa peran IAM untuk diasumsikan saat Anda mengakses bucket Amazon S3 di akun berbeda	18 Oktober 2023

	dari EMR Tanpa Server (6.11 dan lebih tinggi)	
Rilis baru	EMR Tanpa Server 6.14.0	17 Oktober 2023
Fitur baru	Konfigurasi aplikasi default untuk EMR Serverless	25 September 2023
Perbarui ke properti Hive default	Memperbarui nilai default untuk properti pekerjaan hive.driver.disk hive.tez.disk.size , hive.tez.auto.reducer.parallelism ,, dan tez.grouping.min-size Hive.	12 September 2023
Rilis baru	EMR Tanpa Server 6.13.0	11 September 2023
Rilis baru	EMR Tanpa Server 6.12.0	21 Juli 2023
Rilis baru	EMR Tanpa Server 6.11.0	8 Juni 2023
Pembaruan ke kebijakan peran terkait layanan	Memperbarui peran AmazonEMRServerlessServiceRolePolicy _SLR untuk mempublikasikan penggunaan tingkat akun di namespace. "AWS/Usage"	20 April 2023
EMR Serverless ketersediaan umum (GA)	Ini adalah rilis publik pertama EMR Tanpa Server.	1 Juni 2022

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.