aws

Panduan EKS Pengembangan Amazon EMR

Amazon EMR

Amazon EMR: Panduan EKS Pengembangan Amazon EMR

Table of Contents

Apa itu Amazon EMR di EKS?

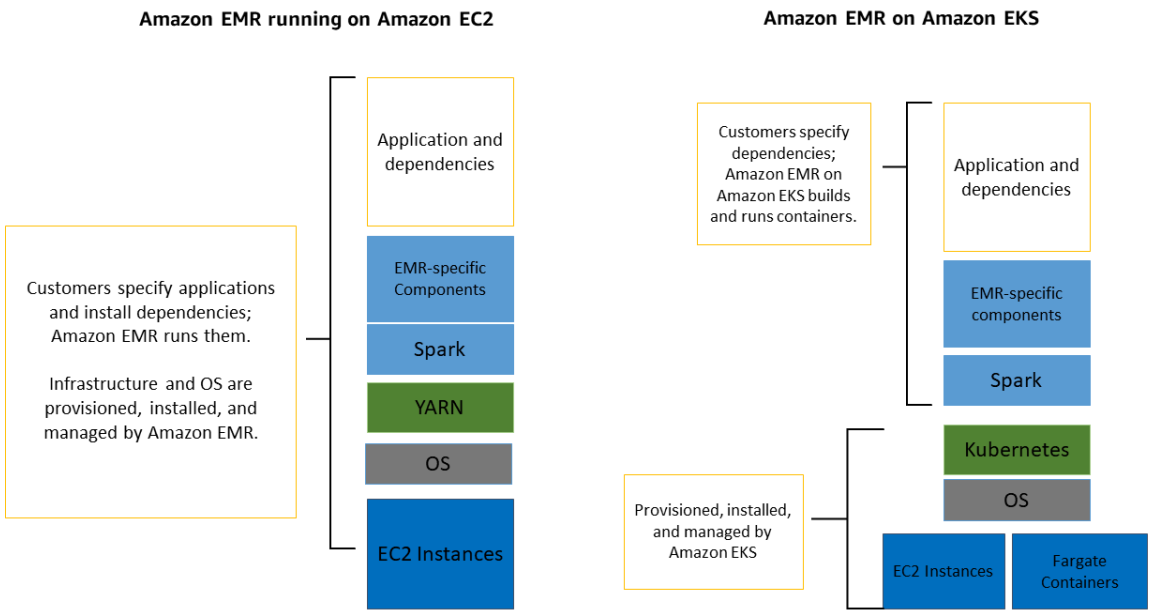
Amazon EMR di EKS menyediakan opsi penyebaran untuk Amazon EMR yang memungkinkan Anda untuk menjalankan kerangka kerja big data sumber terbuka di Amazon Elastic Kubernetes Service (Amazon EKS). Dengan opsi penyebaran ini, Anda dapat fokus pada menjalankan beban kerja analitik sementara Amazon EMR di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

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

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

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

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



Topik

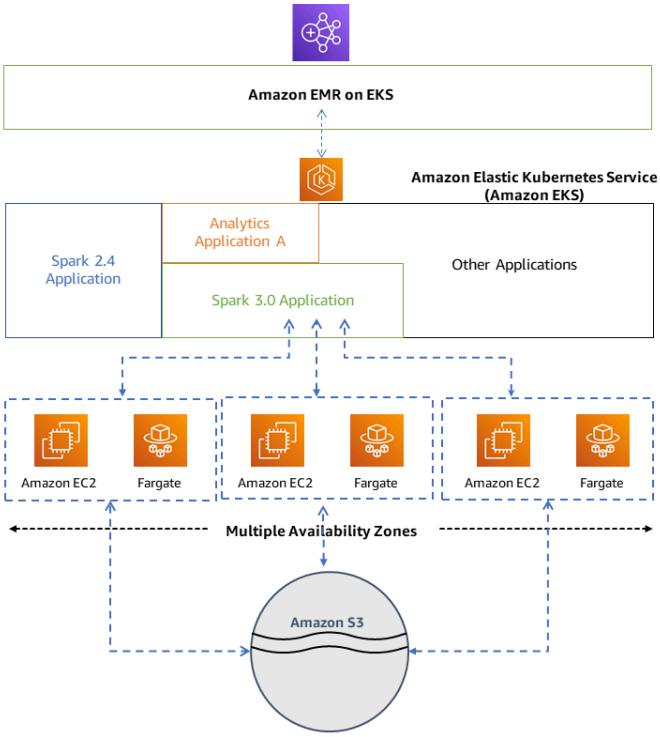
- Arsitektur
- Konsep
- Bagaimana komponen bekerja sama

Arsitektur

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

Dengan penggabungan longgar layanan, Anda dapat menjalankan beberapa tugas yang terisolasi dengan aman secara bersamaan. Anda juga dapat mengukur tugas yang sama dengan backend komputasi yang berbeda atau menyebarkan tugas Anda di beberapa Availability Zones yang sama untuk meningkatkan ketersediaan.

Diagram berikut menggambarkan cara Amazon EMR di EKS bekerja dengan layanan AWS lainnya.



Konsep

Namespace Kubernetes

Amazon EKS menggunakan namespace Kubernetes untuk membagi sumber daya klaster antara beberapa pengguna dan aplikasi. Namespace ini adalah dasar untuk lingkungan multi-penyewa. Sebuah namespace Kubernetes dapat memiliki baik Amazon EC2 atau AWS Fargate sebagai penyedia komputasi. Fleksibilitas ini memberi Anda pilihan performa dan biaya yang berbeda untuk dijalankan pada tugas Anda.

Klaster virtual

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

Tugas berjalan

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

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

Secara default, log diunggah ke server Riwayat Spark dan dapat diakses dari AWS Management Console. Anda juga dapat mendorong log peristiwa, log eksekusi, dan metrik ke Amazon S3 C3 dan Amazon C3 dan Amazon CloudWatch.

Kontainer Amazon EMR

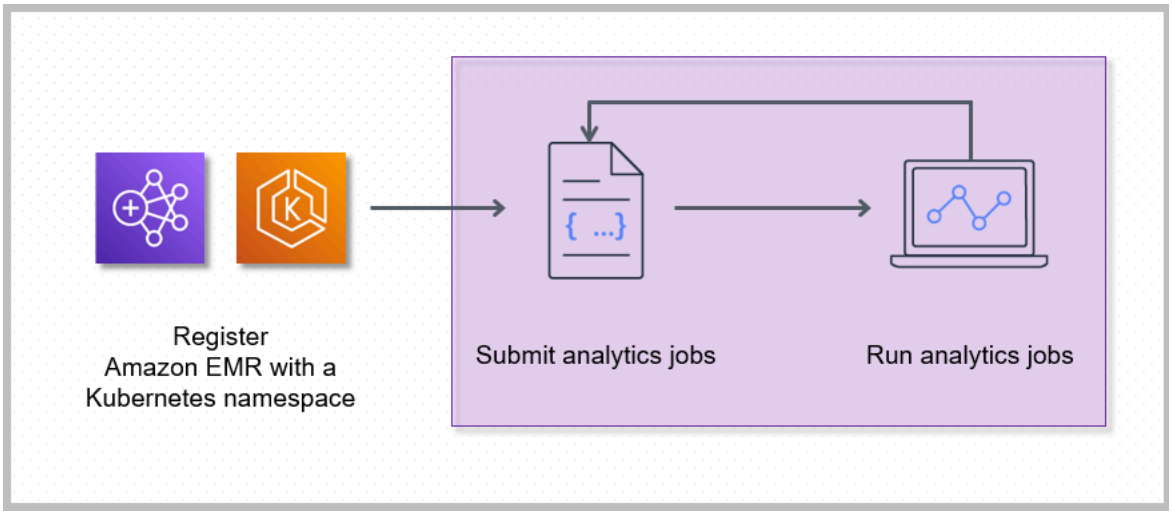
Kontainer Amazon EMR adalah Nama API untuk Amazon EMR di EKS. Prefiks `emr-containers` digunakan dalam skenario berikut:

- Ini adalah prefiks dalam perintah CLI untuk Amazon EMR di EKS. Sebagai contoh, `aws emr-containers start-job-run`.
- Ini adalah prefiks sebelum tindakan kebijakan IAM untuk Amazon EMR di EKS. Sebagai contoh, `"Action": [ "emr-containers:StartJobRun"]`. Untuk informasi selengkapnya, lihat Tindakan kebijakan untuk Amazon EMR di EKS.
- Ini adalah prefiks yang digunakan di Amazon EMR pada titik akhir layanan EKS. Sebagai contoh, `emr-containers.us-east-1.amazonaws.com`. Untuk informasi selengkapnya, lihat Amazon EMR pada Titik Akhir Layanan EKS.

Bagaimana komponen bekerja sama

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

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



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

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

Mulai

Topik ini membantu Anda memulai menggunakan Amazon EMR di EKS dengan menyebarkan aplikasi Spark park park park park park park park park park park park park park park park park park park park park park park park park park park park park park park park park park park park park park Sebelum memulai, pastikan bahwa Anda telah menyelesaikan langkah-langkah dalam Menyiapkan Amazon EMR di EKS. Untuk template lain yang dapat membantu Anda memulai, lihat Panduan Praktik Terbaik Kontainer EMR kami di GitHub.

Anda memerlukan informasi berikut dari langkah-langkah penyiapan:

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

  > **Important**
  >
  > Ketika membuat klaster EKS, pastikan untuk menggunakan m5.xlarge sebagai tipe instans, atau tipe instans lainnya dengan CPU dan memori yang lebih tinggi. Menggunakan jenis instans dengan CPU atau memori yang lebih rendah daripada m5.xlarge dapat menyebabkan kegagalan pekerjaan karena sumber daya yang tidak mencukupi yang tersedia di klaster.

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

> **Important**
>
> Amazon EMR pada pekerjaan EKS menggunakan Amazon CloudWatch dan Amazon S3 sebagai target tujuan untuk pemantauan dan pencatatan log. Anda dapat memantau kemajuan pekerjaan dan memecahkan masalah kegagalan dengan melihat log pekerjaan yang dikirim ke tujuan ini. Untuk mengaktifkan pencatatan, kebijakan IAM yang terkait dengan IAM role untuk pelaksanaan tugas harus memiliki izin yang diperlukan untuk mengakses sumber daya target. Jika kebijakan IAM tidak memiliki izin yang diperlukan, Anda harus

mengikuti langkah-langkah yang diuraikan dalamPerbarui kebijakan kepercayaan dari peran eksekusi tugas, Mengonfigurasi tugas berjalan untuk menggunakan log Amazon S3, dan Mengonfigurasi tugas berjalan untuk menggunakan CloudWatch Log sebelum menjalankan tugas sampel ini.

Jalankan aplikasi Spark park park park park park park

Ambil langkah-langkah berikut untuk menjalankan aplikasi Spark park park park park park park park park park park park park park park park park park park park park park park park park park park park park park `entryPointFile` aplikasi untuk aplikasi Spark Python terletak `dis3://REGION.elasticmapreduce/emr-containers/samples/wordcount/scripts/wordcount.py`. *`REGION`* adalah Wilayah tempat klaster virtual Amazon EMR pada EKS Anda berada, seperti *`us-timur-1`*.

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

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadFromLoggingAndInputScriptBuckets",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::*.elasticmapreduce",
                "arn:aws:s3:::*.elasticmapreduce/*",
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET-OUTPUT",
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET-OUTPUT/*",
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET-LOGGING",
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET-LOGGING/*"
            ]
        },
        {
            "Sid": "WriteToLoggingAndOutputDataBuckets",
            "Effect": "Allow",
            "Action": [
```

```
                "s3:PutObject",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET-OUTPUT/*",
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET-LOGGING/*"
            ]
        },
        {
            "Sid": "DescribeAndCreateCloudwatchLogStream",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:*:*:*"
            ]
        },
        {
            "Sid": "WriteToCloudwatchLogs",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:*:*:log-group:my_log_group_name:log-
stream:my_log_stream_prefix/*"
            ]
        }
    ]
}
```

- Pernyataan pertama`ReadFromLoggingAndInputScriptBuckets` dalam kebijakan ini memberikan`ListBucket` dan`GetObjects` akses ke bucket Amazon S3 berikut:
  - *REGION*`.elasticmapreduce`- ember tempat`entryPoint` file aplikasi berada.
  - *DOC-EXAMPLE-BUCKET-OUTPUT* - bucket yang Anda tentukan untuk data keluaran Anda.
  - *DOC-EXAMPLE-BUCKET-LOGGING* - bucket yang Anda tentukan untuk data logging Anda.
- Pernyataan kedua`WriteToLoggingAndOutputDataBuckets` dalam kebijakan ini memberikan izin pekerjaan untuk menulis data ke output dan mencatat bucket masing-masing.

- Pernyataan ketigaDescribeAndCreateCloudwatchLogStream memberikan pekerjaan dengan izin untuk menggambarkan dan membuat CloudWatch Log Amazon.
- Pernyataan keempatWriteToCloudwatchLogs memberikan izin untuk menulis log ke grup CloudWatch log Amazon yang diberi nama *my_log_group_name*di bawah aliran log bernama *my_log_stream_prefix*.

2. Untuk menjalankan aplikasi Spark Python, gunakan perintah berikut. Ganti semua nilai *miring merah* yang dapat diganti dengan nilai yang sesuai. *REGION* adalah Wilayah tempat klaster virtual Amazon EMR pada EKS Anda berada, seperti *us-timur-1*.

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

Data output dari pekerjaan ini akan tersedia dis3://*DOC-EXAMPLE-BUCKET-OUTPUT*/wordcount_output.

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

3. Untuk menjalankan aplikasi Spark park SQL SQL SQL, gunakan perintah berikut. Ganti semua nilai *miring merah* dengan nilai yang sesuai. *REGION* adalah Wilayah tempat klaster virtual Amazon EMR pada EKS Anda berada, seperti *us-timur-1*.

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

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

```
CREATE DATABASE demo;
CREATE EXTERNAL TABLE IF NOT EXISTS demo.amazonreview( marketplace string,
 customer_id string, review_id  string, product_id  string, product_parent  string,
```

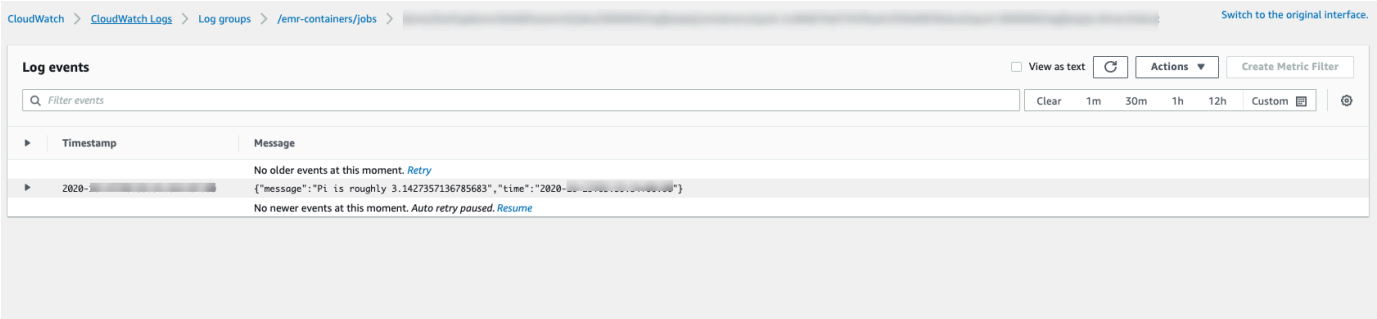
```
 product_title  string, star_rating  integer, helpful_votes  integer, total_votes
  integer, vine  string, verified_purchase  string, review_headline  string,
 review_body  string, review_date  date, year  integer) STORED AS PARQUET LOCATION
 's3://URI to parquet files';
SELECT count(*) FROM demo.amazonreview;
SELECT count(*) FROM demo.amazonreview WHERE star_rating = 3;
```

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

4. Anda juga dapat membuat file JSON dengan parameter tertentu untuk menjalankan tugas Anda. Kemudian jalankan start-job-run perintah dengan jalur ke file JSON. Untuk informasi selengkapnya, lihat Kirim tugas berjalan. Untuk detail lebih lanjut tentang mengonfigurasi parameter untuk menjalankan tugas berjalan, lihat Opsi untuk mengonfigurasi tugas berjalan.

   Untuk memantau kemajuan tugas atau kegagalan debug, Anda dapat memeriksa log yang diunggah ke Amazon S3, CloudWatch Logs, Logs, atau keduanya. Lihat jalur log di Amazon S3 di Konfigurasi tugas berjalan untuk menggunakan log S3 dan untuk Cloudwatch logs di Konfigurasi tugas berjalan untuk menggunakan CloudWatch Log. Untuk melihat log di CloudWatch Log, ikuti petunjuk di bawah ini.

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



Important

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

Tautan ke EMR Amazon di panduan praktik terbaik EKS GitHub

Kami telah membangun EMR Amazon pada Panduan Praktik Terbaik EKS menggunakan kolaborasi komunitas open source sehingga kami dapat mengulangi dengan cepat dan memberikan rekomendasi untuk berbagai kasus penggunaan. Kami menyarankan Anda menggunakan Amazon EMR pada panduan praktik terbaik EKS untuk bagian tersebut. Pilih tautan di setiap bagian untuk pergi ke GitHub situs.

Keamanan

> **Note**
>
> Untuk informasi lebih lanjut tentang keamanan dengan Amazon EMR di EKS, lihat. Praktik terbaik keamanan Amazon EMR di EKS

Praktik terbaik enkripsi: cara menggunakan enkripsi untuk data saat istirahat dan dalam perjalanan.

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

Menggunakan manajer AWS rahasia untuk menyimpan rahasia.

Pengajuan pekerjaan Pyspark

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

Penyimpanan

Menggunakan volume EBS:: cara menggunakan penyediaan statis dan dinamis untuk pekerjaan yang membutuhkan volume EBS.

Menggunakan Amazon FSx for Lustre volume: cara menggunakan penyediaan statis dan dinamis untuk pekerjaan yang membutuhkan Amazon FSx untuk volume Luster.

Menggunakan volume penyimpanan Instance: cara menggunakan volume penyimpanan instance untuk pemrosesan pekerjaan.

Integrasi metastore

Menggunakan Hive metastore: menawarkan berbagai cara untuk menggunakan metastore Hive.

Menggunakan AWS Glue: menawarkan berbagai cara untuk mengkonfigurasi katalog AWS Glue.

Debugging

Menggunakan debugging Spark: cara mengubah level log.

Menghubungkan ke Spark UI pada pod driver.

Cara menggunakan server riwayat Spark yang dihosting sendiri dengan Amazon EMR di EKS.

Memecahkan masalah Amazon EMR pada masalah EKS

Pemecahan masalah.

Penempatan simpul

Menggunakan pemilih node Kubernetes untuk `single-az` dan kasus penggunaan lainnya.

Menggunakan penempatan simpul Fargate.

Performa

Menggunakan Dynamic Resource Allocation (DRA).

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

Optimasi biaya

Menggunakan instans spot: Praktik terbaik instans spot Amazon EC2 dan cara menggunakan fitur dekomisi simpul Spark.

Menggunakan AWS Outposts

Menyesuaikan gambar Docker untuk Amazon di EMR EKS

Anda dapat menggunakan gambar Docker yang disesuaikan dengan EMR Amazon EKS aktif. Menyesuaikan Amazon EMR pada gambar EKS runtime memberikan manfaat berikut:

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

Topik

Cara menyesuaikan gambar Docker

Ambil langkah-langkah berikut untuk menyesuaikan gambar Docker untuk EMR AmazonEKS.

Berikut adalah opsi lain yang mungkin ingin Anda pertimbangkan saat menyesuaikan gambar Docker:

- Sesuaikan gambar Docker untuk titik akhir interaktif
- Bekerja dengan gambar multi-arsitektur

Prasyarat

- Selesaikan Menyiapkan Amazon EMR di EKS langkah-langkah untuk Amazon EMR diEKS.
- Instal Docker di lingkungan Anda. Untuk informasi lebih lanjut, lihat Get Docker.

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

Gambar dasar berisi EMR runtime Amazon dan konektor yang digunakan untuk mengakses AWS layanan lain. Untuk Amazon EMR 6.9.0 dan yang lebih tinggi, Anda bisa mendapatkan gambar dasar dari Galeri ECR Publik Amazon. Jelajahi galeri untuk menemukan tautan gambar dan tarik gambar ke ruang kerja lokal Anda. Misalnya, untuk rilis Amazon EMR 7.2.0, `docker pull` perintah berikut memberi Anda gambar dasar standar terbaru. Anda dapat mengganti `emr-7.2.0:latest` dengan `emr-7.2.0-spark-rapids:latest` untuk mengambil gambar yang memiliki RAPIDS akselerator Nvidia. Anda juga dapat mengganti `emr-7.2.0:latest` dengan `emr-7.2.0-java11:latest` untuk mengambil gambar dengan runtime Java 11.

```
docker pull public.ecr.aws/emr-on-eks/spark/emr-7.2.0:latest
```

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

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

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

   Untuk memilih gambar dasar di Wilayah Anda, lihat Cara memilih gambar dasar URI.
2. Masuk ke ECR repositori Amazon tempat gambar dasar disimpan. Ganti *895885662937* and *us-west-2* dengan akun ECR registri Amazon dan AWS Wilayah yang telah Anda pilih.

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

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

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

Langkah 2: Sesuaikan gambar dasar

Ambil langkah-langkah berikut untuk menyesuaikan gambar dasar yang telah Anda tarik dari AmazonECR.

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

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

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

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

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

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

Langkah 3: (Opsional tapi disarankan) Validasi gambar kustom

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

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

Ambil langkah-langkah berikut untuk memvalidasi gambar kustom Anda.

1. Unduh dan instal Amazon EMR pada gambar EKS khususCLI. Untuk informasi selengkapnya, lihat Amazon EMR pada Panduan CLI Instalasi gambar EKS kustom.
2. Jalankan perintah berikut untuk menguji instalasi.

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

   Berikut ini menunjukkan contoh output.

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

3. Jalankan perintah berikut untuk memvalidasi gambar kustom Anda.

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

   - `-i`menentukan gambar lokal URI yang perlu divalidasi. Ini bisa berupa gambarURI, nama atau tag apa pun yang Anda tentukan untuk gambar Anda.
   - `-r`menentukan versi rilis yang tepat untuk gambar dasar, misalnya,`emr-6.6.0-latest`.
   - `-t`menentukan jenis gambar. Jika ini adalah gambar Spark, masukan`spark`. Nilai default-nya adalah `spark`. Amazon saat ini EMR pada CLI versi gambar EKS khusus hanya mendukung gambar runtime Spark.

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

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

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

Langkah 4: Publikasikan gambar khusus

Publikasikan gambar Docker baru ke ECR registri Amazon Anda.

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

```
aws ecr create-repository \
```

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

Untuk informasi selengkapnya, lihat Membuat repositori di ECRPanduan Pengguna Amazon.

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

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

Untuk informasi selengkapnya, lihat Mengautentikasi ke registri default Anda di Panduan ECR Pengguna Amazon.

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

Tandai gambar.

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

Tekan gambar.

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

Untuk informasi selengkapnya, lihat Mendorong gambar ke Amazon ECR di Panduan ECR Pengguna Amazon.

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

Setelah gambar khusus dibuat dan diterbitkan, Anda dapat mengirimkan Amazon EMR pada EKS pekerjaan menggunakan gambar khusus.

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

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

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

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

```
{
    "name": "spark-custom-image",
    "virtualClusterId": "virtual-cluster-id",
    "executionRoleArn": "execution-role-arn",
    "releaseLabel": "emr-6.6.0-latest",
    "jobDriver": {
      "sparkSubmitJobDriver": {
        "entryPoint": "local:///usr/lib/spark/examples/jars/spark-examples.jar",
```

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

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

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

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

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

Sesuaikan gambar Docker untuk titik akhir interaktif

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

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

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

Anda perlu menggunakan `notebook-spark` gambar dasarURI, bukan`spark`. Gambar dasar berisi runtime Spark dan kernel notebook yang berjalan dengannya. Untuk informasi selengkapnya tentang memilih tag gambar Wilayah dan kontainer, lihatCara memilih gambar dasar URI.

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

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

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

```
{
    "name": "endpoint-name",
    "virtualClusterId": "virtual-cluster-id",
    "type": "JUPYTER_ENTERPRISE_GATEWAY",
    "releaseLabel": "emr-6.6.0-latest",
    "executionRoleArn": "execution-role-arn",
    "certificateArn": "certificate-arn",
    "configurationOverrides": {
        "applicationConfiguration": [
            {
                "classification": "jupyter-kernel-overrides",
                "configurations": [
                    {
                        "classification": "python3",
                        "properties": {
```

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

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

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

Untuk informasi selengkapnya, lihat Membuat titik akhir interaktif untuk klaster virtual Anda.

3. Connect ke endpoint interaktif melalui EMR Studio. Untuk informasi selengkapnya, lihat Menghubungkan dari Studio.

Bekerja dengan gambar multi-arsitektur

EMRAmazon EKS mendukung gambar wadah multi-arsitektur untuk Amazon Elastic Container Registry (AmazonECR). Untuk informasi selengkapnya, lihat Memperkenalkan gambar kontainer multi-arsitektur untuk Amazon ECR.

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

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

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

Berikut adalah outputnya. `arm64`Arsitekturnya untuk contoh Graviton. `amd64`Ini untuk contoh non-Graviton.

```
{
   "schemaVersion": 2,
   "mediaType": "application/vnd.docker.distribution.manifest.list.v2+json",
   "manifests": [
      {
         "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
         "size": 1805,
         "digest":
 "xxx123:6b971cb47d11011ab3d45fff925e9442914b4977ae0f9fbcdcf5cfa99a7593f0",
         "platform": {
            "architecture": "arm64",
            "os": "linux"
         }
      },
      {
         "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
         "size": 1805,
         "digest":
 "xxx123:6f2375582c9c57fa9838c1d3a626f1b4fc281e287d2963a72dfe0bd81117e52f",
         "platform": {
            "architecture": "amd64",
            "os": "linux"
         }
      }
   ]
}
```

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

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

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

   RUN pip3 install boto3 // install customizations here
   ```

```
USER hadoop:hadoop
```

2. Ikuti petunjuk di Memperkenalkan gambar wadah multi-arsitektur untuk Amazon ECR untuk membuat gambar multi-arsitektur.

> **Note**
>
> Anda harus membuat `arm64` gambar pada `arm64` instance. Demikian pula, Anda harus membangun `amd64` gambar pada `amd64` instance.

Anda juga dapat membuat gambar multi-arsitektur tanpa membangun setiap jenis instance tertentu dengan perintah Docker`buildx`. Untuk informasi selengkapnya, lihat Memanfaatkan dukungan CPU multi-arsitektur.

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

Cara memilih gambar dasar URI

> **Note**
>
> Untuk rilis Amazon EMR 6.9.0 dan yang lebih baru, Anda dapat mengambil gambar dasar dari Galeri ECR Publik Amazon, jadi Anda tidak perlu membuat gambar dasar URI sebagai petunjuk langsung di halaman ini. Untuk menemukan tag gambar kontainer untuk gambar dasar Anda, lihat halaman catatan rilis untuk rilis EMR Amazon yang sesuaiEKS.

Gambar Docker dasar yang dapat Anda pilih disimpan di Amazon Elastic Container Registry (AmazonECR). Gambar URI mengikuti format ini:*ECR-registry-account*`.dkr.ecr.`*Region*`.amazonaws.com/spark/`*container-image-tag*, seperti contoh berikut menunjukkan.

```
895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-7.2.0:latest
```

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

```
895885662937.dkr.ecr.us-west-2.amazonaws.com/notebook-spark/emr-7.2.0:latest
```

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

```
895885662937.dkr.ecr.us-west-2.amazonaws.com/notebook-python/emr-7.2.0:latest
```

Untuk menemukan tag gambar kontainer untuk gambar dasar Anda, lihat halaman catatan rilis untuk rilis EMR Amazon yang sesuaiEKS.

Akun ECR registri Amazon berdasarkan Wilayah

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

Wilayah	Akun ECR registri Amazon
ap-northeast-1	059004520145
ap-northeast-2	996579266876
ap-south-1	235914868574
ap-southeast-1	671219180197
ap-southeast-2	038297999601
ca-central-1	351826393999
eu-central-1	107292555468
eu-north-1	830386416364

Wilayah	Akun ECR registri Amazon	
eu-west-1	483788554619	
eu-west-2	118780647275	
eu-west-3	307523725174	
sa-east-1	052806832358	
us-east-1	755674844232	
us-east-2	711395599931	
us-west-1	608033475327	
us-west-2	895885662937	

Pertimbangan

Saat Anda menyesuaikan gambar Docker, Anda dapat memilih runtime yang tepat untuk pekerjaan Anda pada tingkat granular. Ikuti praktik terbaik ini saat Anda menggunakan fitur ini:

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

  - `/var/log/fluentd`
  - `/var/log/spark/user`
  - `/var/log/spark/apps`
  - `/mnt`
  - `/tmp`
  - `/home/hadoop`
- Anda dapat mengunggah gambar yang disesuaikan ke repositori yang kompatibel dengan Docker, seperti Amazon, Docker HubECR, atau repositori perusahaan swasta. Untuk informasi selengkapnya tentang cara mengonfigurasi EKS autentikasi klaster Amazon dengan repositori Docker yang dipilih, lihat Tarik Gambar dari Registri Pribadi.

Menjalankan pekerjaan Flink dengan Amazon EMR di EKS

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

Topik

Operator Flink Kubernetes

Halaman-halaman berikut menjelaskan cara mengatur dan menggunakan operator Flink Kubernetes untuk menjalankan pekerjaan Flink dengan Amazon aktif. EMR EKS

Topik

Menyiapkan operator Flink Kubernetes untuk Amazon di EMR EKS

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

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

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

```
            "Condition": {
                "StringLike": {
                    "OIDC_PROVIDER:sub": "system:serviceaccount:NAMESPACE:emr-
containers-sa-flink-operator"
                }
            }
        }
    ]
}
```

Memulai dengan operator Flink Kubernetes untuk Amazon di EMR EKS

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

Menginstal operator

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

1. Jika Anda belum melakukannya, selesaikan langkah-langkahnyathe section called “Pengaturan”.
2. Instal *cert-manager* (sekali per EKS cluster Amazon) untuk mengaktifkan penambahan komponen webhook.

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

3. Instal bagan Helm.

   ```
   export VERSION=7.2.0 # The Amazon EMR release version
   export NAMESPACE=The Kubernetes namespace to deploy the operator

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

   Contoh output:

   ```
   NAME: flink-kubernetes-operator
   LAST DEPLOYED: Tue May 31 17:38:56 2022
   ```

```
NAMESPACE: $NAMESPACE
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

4. Tunggu hingga penerapan selesai dan verifikasi instalasi bagan.

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

5. Anda akan melihat pesan berikut saat penerapan selesai.

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

6. Gunakan perintah berikut untuk melihat operator yang digunakan.

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

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

```
NAME                            NAMESPACE    REVISION    UPDATED
               STATUS      CHART                                  APP VERSION

flink-kubernetes-operator    $NAMESPACE   1           2023-02-22 16:43:45.24148
 -0500 EST    deployed    flink-kubernetes-operator-emr-7.2.0    x.y.z-amzn-n
```

Menjalankan aplikasi Flink

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

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

Prasyarat — Sebelum Anda dapat menjalankan aplikasi Flink dengan operator Flink Kubernetes, selesaikan langkah-langkah di dan. the section called “Pengaturan” the section called “Menginstal operator”

Application mode

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

1. Buat `FlinkDeployment` file definisi file `basic-example-app-cluster.yaml` dengan contoh konten berikut:

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: basic-example-app-cluster
spec:
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
    state.checkpoints.dir: CHECKPOINT_S3_STORAGE_PATH
    state.savepoints.dir: SAVEPOINT_S3_STORAGE_PATH
  flinkVersion: v1_17
  executionRoleArn: JOB_EXECUTION_ROLE_ARN
  emrReleaseLabel: "emr-6.13.0-flink-latest" # 6.13 or higher
  jobManager:
    storageDir: HIGH_AVAILABILITY_STORAGE_PATH
    resource:
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
  job:
    # if you have your job jar in S3 bucket you can use that path as well
```

```
    jarURI: local:///opt/flink/examples/streaming/StateMachineExample.jar
    parallelism: 2
    upgradeMode: savepoint
    savepointTriggerNonce: 0
  monitoringConfiguration:
    cloudWatchMonitoringConfiguration:
       logGroupName: LOG_GROUP_NAME
```

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

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

3. Akses UI Flink.

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

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

Untuk informasi selengkapnya tentang mengirimkan aplikasi ke Flink melalui operator Flink Kubernetes, lihat contoh operator Flink Kubernetes di folder di. `apache/flink-kubernetes-operator` GitHub

Session mode

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

1. Buat `FlinkDeployment` file definisi file `basic-example-session-cluster.yaml` dengan contoh konten berikut:

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: basic-example-session-cluster
spec:
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
```

```
    state.checkpoints.dir: CHECKPOINT_S3_STORAGE_PATH
    state.savepoints.dir: SAVEPOINT_S3_STORAGE_PATH
  flinkVersion: v1_17
  executionRoleArn: JOB_EXECUTION_ROLE_ARN
  emrReleaseLabel: "emr-6.15.0-flink-latest"
  jobManager:
    storageDir: HIGH_AVAILABILITY_S3_STORAGE_PATH
    resource:
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
  monitoringConfiguration:
    s3MonitoringConfiguration:
       logUri:
    cloudWatchMonitoringConfiguration:
       logGroupName: LOG_GROUP_NAME
```

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

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

3. Gunakan perintah berikut untuk mengonfirmasi bahwa cluster sesi `LIFECYCLE` adalah`STABLE`:

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

Outputnya harus mirip dengan contoh berikut:

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

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

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

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

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

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

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

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

Outputnya harus mirip dengan contoh berikut:

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

7. Akses UI Flink.

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

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

Keamanan

RBAC

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

Peran operator

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

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

```
rules:
- apiGroups:
  - ""
  resources:
  - pods
  - services
  - events
  - configmaps
  - secrets
  - serviceaccounts
  verbs:
  - '*'
- apiGroups:
  - rbac.authorization.k8s.io
  resources:
  - roles
  - rolebindings
  verbs:
  - '*'
- apiGroups:
  - apps
  resources:
  - deployments
  - deployments/finalizers
  - replicasets
  verbs:
```

```
  - '*'
- apiGroups:
  - extensions
  resources:
  - deployments
  - ingresses
  verbs:
  - '*'
- apiGroups:
  - flink.apache.org
  resources:
  - flinkdeployments
  - flinkdeployments/status
  - flinksessionjobs
  - flinksessionjobs/status
  verbs:
  - '*'
- apiGroups:
  - networking.k8s.io
  resources:
  - ingresses
  verbs:
  - '*'
- apiGroups:
  - coordination.k8s.io
  resources:
  - leases
  verbs:
  - '*'
```

Peran Job

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

```
rules:
- apiGroups:
  - ""
  resources:
  - pods
  - configmaps
  verbs:
  - '*'
```

```
- apiGroups:
  - apps
  resources:
  - deployments
  - deployments/finalizers
  verbs:
  - '*'
```

Menghapus instalasi operator Flink Kubernetes untuk Amazon di EMR EKS

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

1. Hapus operator.

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

2. Hapus sumber daya Kubernetes yang tidak dihapus Helm.

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

3. (Opsional) Hapus cert-manager.

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

Kubernetes Asli

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

Topik

- Menyiapkan Flink Native Kubernetes untuk Amazon EMR di EKS
- Memulai dengan Kubernetes asli Flink untuk Amazon EMR di EKS
- Persyaratan keamanan akun JobManager layanan Flink untuk Native Kubernetes

Menyiapkan Flink Native Kubernetes untuk Amazon EMR di EKS

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

- Instal AWS CLI— Jika Anda sudah menginstalAWS CLI, konfirmasikan bahwa Anda memiliki versi terbaru.
- Siapkan cluster Amazon EKS — Ikuti langkah-langkah untuk membuat cluster Kubernetes baru dengan node di Amazon EKS.
- Pilih URI image dasar EMR Amazon (rilis 6.13.0 atau lebih tinggi) — perintah Flink Kubernetes didukung dengan Amazon EMR rilis 6.13.0 dan yang lebih tinggi.
- Konfirmasikan bahwa akun JobManager layanan memiliki izin yang sesuai untuk membuat dan menonton TaskManager pod. Untuk informasi selengkapnya, lihat persyaratan keamanan akun JobManager layanan Flink untuk Kubernetes Asli.
- Siapkan profil AWSkredensial lokal Anda.
- Buat atau perbarui file kubeconfig untuk cluster Amazon EKS tempat Anda ingin menjalankan aplikasi Flink.

Memulai dengan Kubernetes asli Flink untuk Amazon EMR di EKS

Jalankan aplikasi Flink

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

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

```
#Export the FLINK_HOME environment variable to your local installation of Flink
export FLINK_HOME=/usr/local/bin/flink #Will vary depending on your installation
export NAMESPACE=flink
```

```
export CLUSTER_ID=flink-application-cluster
export IMAGE=<123456789012.dkr.ecr.sample-Wilayah AWS-.amazonaws.com/flink/
emr-6.13.0-flink:latest>
export FLINK_SERVICE_ACCOUNT=emr-containers-sa-flink
export FLINK_CLUSTER_ROLE_BINDING=emr-containers-crb-flink
```

4. Buat akun layanan untuk mengelola sumber daya Kubernetes.

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

5. Jalankan `run-application` perintah CLI.

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

6. Periksa sumber daya Kubernetes yang dibuat.

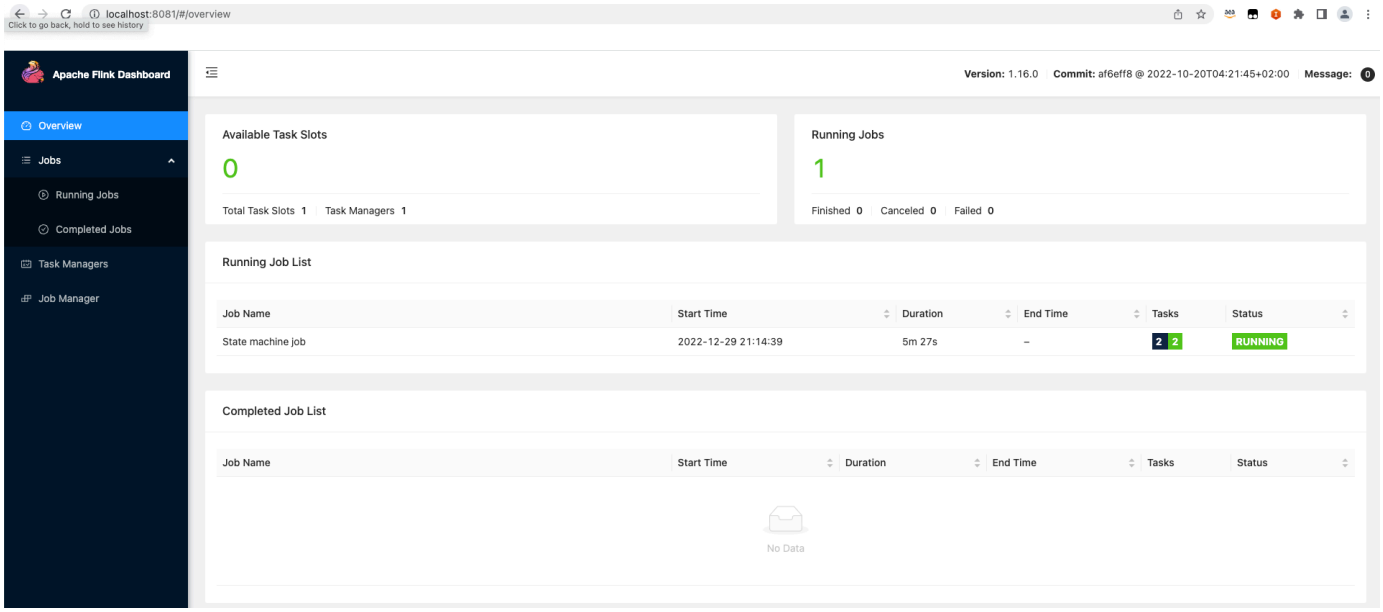
```
kubectl get all -n <namespace>
NAME READY STATUS RESTARTS AGE
pod/flink-application-cluster-546687cb47-w2p2z 1/1 Running 0 3m37s
```

```
pod/flink-application-cluster-taskmanager-1-1 1/1 Running 0 3m24s

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

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

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

7. Port maju ke 8081.

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

8. Akses UI Flink secara lokal.



9. Hapus aplikasi Flink.

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

Untuk informasi selengkapnya tentang mengirimkan aplikasi ke Flink, lihat Native Kubernetes di dokumentasi Apache Flink.

Persyaratan keamanan akun JobManager layanan Flink untuk Native Kubernetes

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

Aturan berikut berlaku untuk akun layanan ini.

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

Menyesuaikan gambar Docker untuk Amazon EKS dengan EMR Apache Flink

Bagian berikut menjelaskan cara menyesuaikan gambar Docker untuk EMR AmazonEKS.

Topik

Menyesuaikan gambar Docker untuk Flink dan FluentD

Ambil langkah-langkah berikut untuk menyesuaikan gambar Docker untuk Amazon EKS dengan EMR Apache Flink atau gambar FluentD.

Topik

Prasyarat

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

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

Langkah 1: Ambil gambar dasar dari Amazon Elastic Container Registry

Gambar dasar berisi EMR runtime Amazon dan konektor yang perlu Anda akses lainnya AWS layanan. Jika Anda menggunakan EMR Amazon EKS dengan Flink versi 6.14.0 atau lebih tinggi, Anda bisa mendapatkan gambar dasar dari Galeri Publik AmazonECR. Jelajahi galeri untuk

menemukan tautan gambar dan tarik gambar ke ruang kerja lokal Anda. Misalnya, untuk rilis Amazon EMR 6.14.0, `docker pull` perintah berikut mengembalikan gambar dasar standar terbaru. Ganti `emr-6.14.0:latest` dengan versi rilis yang Anda inginkan.

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

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

- emr-on-eks/flink/emr-6.14.0-flink
- emr-on-eks/fasih/emr-6.14.0 (

Langkah 2: Sesuaikan gambar dasar

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

1. Buat `Dockerfile` baru di Workspace lokal Anda.
2. Edit `Dockerfile` dan tambahkan konten berikut. Ini `Dockerfile` menggunakan gambar kontainer yang Anda tarik`public.ecr.aws/emr-on-eks/flink/emr-7.2.0-flink:latest`.

   ```
   FROM public.ecr.aws/emr-on-eks/flink/emr-7.2.0-flink:latest
   USER root
   ### Add customization commands here ####
   USER hadoop:hadoop
   ```

   Gunakan konfigurasi berikut jika Anda menggunakan`Fluentd`.

   ```
   FROM public.ecr.aws/emr-on-eks/fluentd/emr-7.2.0:latest
   USER root
   ### Add customization commands here ####
   USER hadoop:hadoop
   ```

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

   ```
   FROM public.ecr.aws/emr-on-eks/flink/emr-7.2.0-flink:latest
   USER root
   RUN pip3 install --upgrade boto3 pandas numpy // For python 3
   ```

```
USER hadoop:hadoop
```

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

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

Langkah 3: Publikasikan gambar kustom Anda

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

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

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

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

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

3. Tekan gambar. Untuk informasi selengkapnya, lihat Mendorong gambar ke Amazon ECR di Panduan Pengguna Amazon Elastic Container Registry.

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

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

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

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

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

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

Memantau pekerjaan operator Flink Kubernetes dan Flink

Bagian ini menjelaskan beberapa cara agar Anda dapat memantau pekerjaan Flink Anda dengan Amazon EMR di EKS.

Topik

- Menggunakan Amazon Managed Service untuk Prometheus untuk memantau pekerjaan Flink
- Menggunakan UI Flink untuk memantau pekerjaan Flink
- Menggunakan konfigurasi pemantauan untuk memantau operator Flink Kubernetes dan pekerjaan Flink

Menggunakan Amazon Managed Service untuk Prometheus untuk memantau pekerjaan Flink

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

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

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

   Jika Anda belum memiliki peran, siapkan peran layanan untuk menelan metrik dari kluster Amazon EKS.

   Sebelum Anda melanjutkan, buat peran IAM yang disebut`amp-iamproxy-ingest-role`.
3. Instal Operator Flink EMR Amazon dengan Amazon Managed Service untuk Prometheus.

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

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

```
kube-prometheus-stack:
    prometheus:
    serviceAccount:
        create: true
        name: "amp-iamproxy-ingest-service-account"
```

```
        annotations:
            eks.amazonaws.com/role-arn: "arn:aws:iam::<AWS_ACCOUNT_ID>:role/amp-
iamproxy-ingest-role"
    remoteWrite:
        - url: <AMAZON_MANAGED_PROMETHEUS_REMOTE_WRITE_URL>
        sigv4:
            region: <AWS_REGION>
        queueConfig:
            maxSamplesPerSend: 1000
            maxShards: 200
            capacity: 2500
```

Gunakan Helm Install --setperintah untuk meneruskan penggantian ke bagan. `flink-kubernetes-operator`

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

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

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

Menggunakan UI Flink untuk memantau pekerjaan Flink

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

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

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

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

Contoh output:

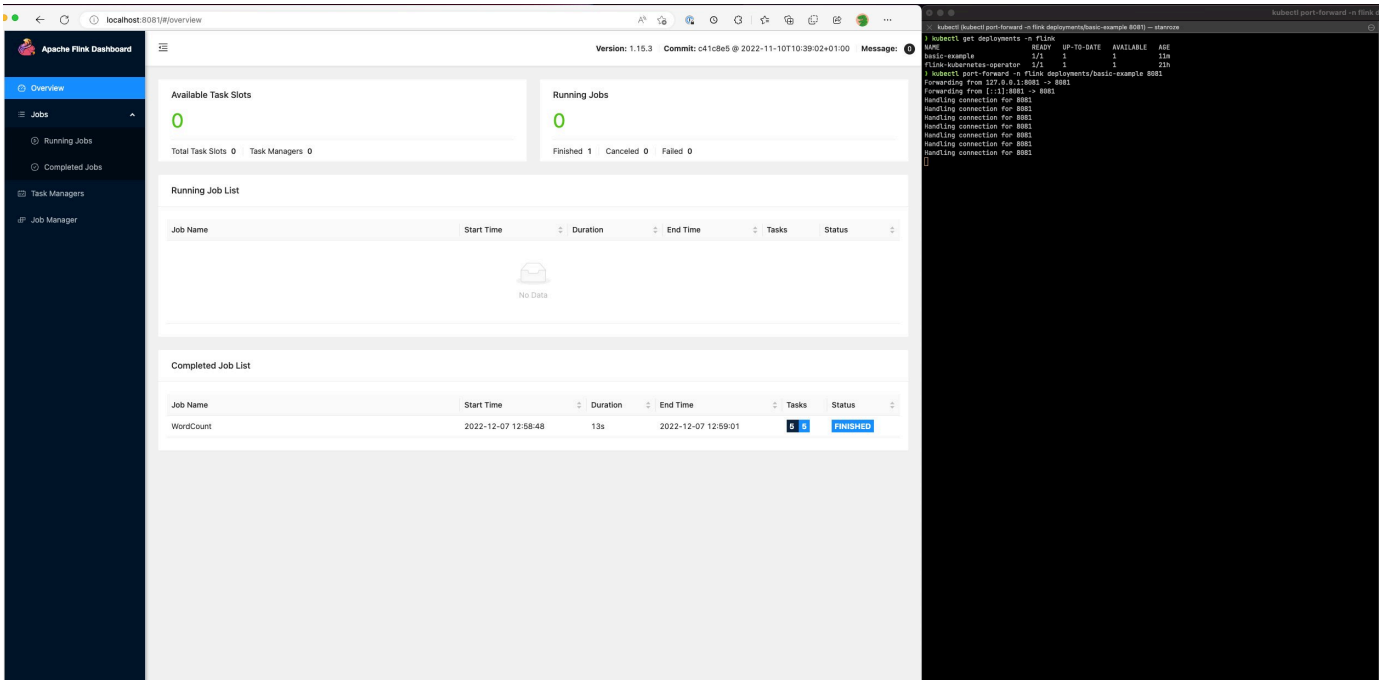
```
kubectl get deployments -n flink-namespace
NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
basic-example               1/1       1            1           11m
flink-kubernetes-operator   1/1       1            1           21h
```

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

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

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

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



Menggunakan konfigurasi pemantauan untuk memantau operator Flink Kubernetes dan pekerjaan Flink

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

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

Log aplikasi Flink

Anda dapat menentukan konfigurasi ini dengan cara berikut.

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: basic-example
spec:
  image: FLINK IMAGE TAG
  imagePullPolicy: Always
  flinkVersion: v1_17
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
  executionRoleArn: JOB EXECUTION ROLE
  jobManager:
    resource:
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
  job:
```

```
    jarURI: local:///opt/flink/examples/streaming/StateMachineExample.jar
  monitoringConfiguration:
    s3MonitoringConfiguration:
      logUri: S3 BUCKET
    cloudWatchMonitoringConfiguration:
      logGroupName: LOG GROUP NAME
      logStreamNamePrefix: LOG GROUP STREAM PREFIX
    sideCarResources:
      limits:
        cpuLimit: 500m
        memoryLimit: 250Mi
    containerLogRotationConfiguration:
        rotationSize: 2GB
        maxFilesToKeep: 10
```

Berikut ini adalah opsi konfigurasi.

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

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

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

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

      Format berikut adalah angka yang bertambah.

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

  - Izin IAM berikut diperlukan untuk menggunakan forwarder ini.

    ```
    {
        "Effect": "Allow",
        "Action": [
            "s3:PutObject"
        ],
        "Resource": [
    ```

```
        "S3_BUCKET_URI/*",
        "S3_BUCKET_URI"
    ]
}
```

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

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

  - Izin IAM berikut diperlukan untuk menggunakan forwarder ini.

    ```
    {
        "Effect": "Allow",
        "Action": [
            "logs:CreateLogStream",
            "logs:CreateLogGroup",
            "logs:PutLogEvents"
        ],
        "Resource": [
            "arn:aws:logs:REGION:ACCOUNT-ID:log-group:{YOUR_LOG_GROUP_NAME}:*",
            "arn:aws:logs:REGION:ACCOUNT-ID:log-group:{YOUR_LOG_GROUP_NAME}"
        ]
    }
    ```

- `sideCarResources`(opsional) — kunci konfigurasi untuk menetapkan batas sumber daya pada wadah sidecar Fluentbit yang diluncurkan.
  - `memoryLimit`(opsional) - nilai default adalah 512Mi. Sesuaikan sesuai dengan kebutuhan Anda.
  - `cpuLimit`(opsional) — opsi ini tidak memiliki default. Sesuaikan sesuai dengan kebutuhan Anda.
- `containerLogRotationConfiguration`(opsional) — mengontrol perilaku rotasi log kontainer. Agen tidak diaktifkan secara default.
  - `rotationSize`(wajib) - menentukan ukuran file untuk rotasi log. Kisaran nilai yang mungkin adalah dari 2KB hingga 2GB. Bagian unit numerik dari parameter RotationSize dilewatkan

sebagai bilangan bulat. Karena nilai desimal tidak didukung, Anda dapat menentukan ukuran rotasi 1,5GB, misalnya, dengan nilai 1500MB. Defaultnya adalah 2GB.

- `maxFilesToKeep`(wajib) — menentukan jumlah maksimum file untuk disimpan dalam wadah setelah rotasi telah terjadi. Nilai minimum adalah 1, dan nilai maksimum adalah 50. Default-nya adalah 10.

Log operator Flink

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

```
monitoringConfiguration:
  s3MonitoringConfiguration:
    logUri: "S3-BUCKET"
    totalFileSize: "1G"
    uploadTimeout: "1m"
  cloudWatchMonitoringConfiguration:
    logGroupName: "flink-log-group"
    logStreamNamePrefix: "example-job-prefix-test-2"
  sideCarResources:
    limits:
      cpuLimit: 1
      memoryLimit: 800Mi
  memoryBufferLimit: 700M
```

Berikut ini adalah opsi konfigurasi yang tersedia di bawah`monitoringConfiguration`.

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

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

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

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

Indeks format berikut adalah angka yang bertambah.

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

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

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

Ketahanan Job

Bagian berikut menguraikan cara membuat pekerjaan Flink Anda lebih andal dan sangat tersedia.

Topik

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

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

Operator Flink ketersediaan tinggi

Kami mengaktifkan ketersediaan tinggi untuk Operator Flink sehingga kami dapat gagal ke Operator Flink siaga untuk meminimalkan waktu henti di loop kontrol operator jika terjadi kegagalan. Ketersediaan tinggi diaktifkan secara default dan jumlah default replika operator awal adalah 2. Anda dapat mengonfigurasi bidang replika di `values.yaml` file Anda untuk bagan helm.

Bidang berikut dapat disesuaikan:

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

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

```
...
imagePullSecrets: []

replicas: 1

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

Penyebaran multi AZ

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

Menentukan replika pemimpin

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

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

Interaksi Flink-S3

Mengkonfigurasi kredenal akses

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

Mengambil stoples pekerjaan dari mode Aplikasi S3

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

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

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

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: python-example
spec:
  image: <YOUR CUSTOM PYFLINK IMAGE>
  emrReleaseLabel: "emr-6.12.0-flink-latest"
  flinkVersion: v1_16
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "1"
  serviceAccount: flink
  jobManager:
    highAvailabilityEnabled: false
```

```
    replicas: 1
    resource:
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
  job:
    jarURI: "s3://<S3-BUCKET>/scripts/pyflink.py" # Note, this will trigger the
 artifact download process
    entryClass: "org.apache.flink.client.python.PythonDriver"
    args: ["-pyclientexec", "/usr/local/bin/python3", "-py", "/opt/flink/usrlib/
pyflink.py"]
    parallelism: 1
    upgradeMode: stateless
```

Konektor Flink S3

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

Checkpointing: Konektor Presto S3

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

Contoh FlinkDeployment spesifikasi:

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

Membaca dan menulis ke S3: konektor Hadoop S3

- Atur skema S3 ke `s3://` atau () `s3a://`

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

Contoh FlinkDeployment spesifikasi

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: basic-example
spec:
  job:
    jarURI: local:///opt/flink/examples/streaming/WordCount.jar
    args: [ "--input", "s3a://<INPUT BUCKET>/PATH", "--output", "s3a://<OUTPUT BUCKET>/
PATH" ]
    parallelism: 2
    upgradeMode: stateless
```

Manajer Pekerjaan Flink

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

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

Konfigurasi HA

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

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

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

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

Lokalitas pod

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

Menentukan replika pemimpin

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

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

Pekerjaan Flink - Kubernetes asli

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

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

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

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

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

`Dkubernetes.jobmanager.replicas`— Jumlah pod Job Manager yang akan dibuat sebagai bilangan bulat lebih besar dari1.

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

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

Ketika tugas gagal atau ketika operasi penskalaan terjadi, Flink mencoba untuk menjalankan kembali tugas dari pos pemeriksaan terakhir selesai. Proses restart bisa memakan waktu satu menit atau lebih lama untuk dijalankan, tergantung pada ukuran status pos pemeriksaan dan jumlah tugas paralel. Selama periode restart, tugas backlog dapat menumpuk untuk pekerjaan itu. Ada beberapa cara, bahwa Flink mengoptimalkan kecepatan pemulihan dan memulai ulang grafik eksekusi untuk meningkatkan stabilitas pekerjaan.

Halaman ini menjelaskan beberapa cara Amazon EMR Flink dapat meningkatkan waktu restart pekerjaan selama pemulihan tugas atau operasi penskalaan.

Topik

Tugas-pemulihan lokal

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

Dengan pos pemeriksaan Flink, setiap tugas menghasilkan snapshot statusnya yang ditulis Flink ke penyimpanan terdistribusi seperti Amazon S3. Dalam kasus pemulihan, tugas mengembalikan keadaan mereka dari penyimpanan terdistribusi. Penyimpanan terdistribusi memberikan toleransi

kesalahan dan dapat mendistribusikan kembali status selama penskalaan ulang karena dapat diakses oleh semua node.

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

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

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

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

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

Pemulihan tugas-lokal dengan pemasangan volume Amazon EBS

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

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

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

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

```
export AWS_REGION=aa-example-1
export FLINK_EKS_CLUSTER_NAME=my-cluster
export AWS_ACCOUNT_ID=111122223333
```

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

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

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

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

4. Buat driver Amazon EBS CSI dengan perintah berikut:

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

5. Buat kelas penyimpanan Amazon EBS dengan perintah berikut:

```
cat # EOF # storage-class.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
```

```
  name: ebs-sc
provisioner: ebs.csi.aws.com
volumeBindingMode: WaitForFirstConsumer
EOF
```

Dan kemudian terapkan kelas:

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

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

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

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

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

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

```
 # Detach Attached Policy
 aws iam detach-role-policy --role-name TLR_${$AWS_REGION}_${FLINK_EKS_CLUSTER_NAME}
--policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy
 # Delete the created Role
 aws iam delete-role --role-name TLR_${$AWS_REGION}_${FLINK_EKS_CLUSTER_NAME}
```

```
  # Delete the created service account
  eksctl delete iamserviceaccount --name ebs-csi-controller-sa --namespace kube-system
 --cluster $FLINK_EKS_CLUSTER_NAME --region $AWS_REGION
  # Delete Addon
  eksctl delete addon --name aws-ebs-csi-driver --cluster $FLINK_EKS_CLUSTER_NAME --
region $AWS_REGION
  # Delete the EBS storage class
  kubectl delete -f storage-class.yaml
```

Pos pemeriksaan inkremental berbasis log generik

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

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

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

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

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

Pemulihan berbutir halus

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

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

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

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

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

Ada dua komponen yang terhubung dengan pipa:`a1 # e1`, dana`2 # e2`. Jika salah satu TM1 atau TM2 gagal, kegagalan hanya berdampak pada 5 tugas dalam pipeline tempat TaskManager sedang berjalan. Strategi restart hanya memulai komponen pipelined yang terpengaruh.

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

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

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

Mekanisme restart gabungan dalam penjadwal adaptif

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

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

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

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

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

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

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

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

Gambaran Umum

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

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

Kemampuan ini memberikan manfaat sebagai berikut:

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

Cara kerjanya

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

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

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

Prasyarat

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

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

Konfigurasi

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

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

Menggunakan Autoscaler untuk aplikasi Flink

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

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

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

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

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

Berikut ini adalah opsi konfigurasi untuk autoscaler.

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

- `job.autoscaler.target.utilization`— pemanfaatan simpul target untuk memberikan kinerja pekerjaan yang stabil dan beberapa buffer untuk fluktuasi beban. Defaultnya `0.7` menargetkan 70% pemanfaatan/pemuatan untuk simpul pekerjaan.
- `job.autoscaler.target.utilization.boundary`— batas pemanfaatan simpul target yang berfungsi sebagai buffer ekstra untuk menghindari penskalaan langsung pada fluktuasi beban. Defaultnya adalah`0.3`, yang berarti 30% deviasi dari pemanfaatan target diperbolehkan sebelum memicu tindakan penskalaan.
- `ob.autoscaler.restart.time`— waktu yang diharapkan untuk me-restart aplikasi. Default adalah 5 menit.
- `job.autoscaler.catch-up.duration`— waktu yang diharapkan untuk catch up, yang berarti sepenuhnya memproses setiap backlog setelah operasi penskalaan selesai. Default adalah 5 menit. Dengan menurunkan durasi catch-up, autoscaler harus memesan lebih banyak kapasitas ekstra untuk tindakan penskalaan.
- `pipeline.max-parallelism`— paralelisme maksimum yang dapat digunakan autoscaler. Autoscaler mengabaikan batas ini jika lebih tinggi dari paralelisme maks yang dikonfigurasi dalam konfigurasi Flink atau langsung pada setiap operator. Defaultnya adalah -1. Perhatikan bahwa autoscaler menghitung paralelisme sebagai pembagi bilangan paralelisme maks oleh karena itu disarankan untuk memilih pengaturan paralelisme maks yang memiliki banyak pembagi daripada mengandalkan default yang disediakan Flink. Kami merekomendasikan penggunaan kelipatan 60 untuk konfigurasi ini, seperti 120, 180, 240, 360, 720 dll.

Untuk halaman referensi konfigurasi yang lebih detail, lihat Konfigurasi Autoscaler.

Autotuning parameter Autoscaler

> **Note**
>
> Amazon EMR 7.2.0 dan yang lebih tinggi menggunakan konfigurasi open source `job.autoscaler.restart.time-tracking.enabled` untuk mengaktifkan estimasi waktu penskalaan ulang. Estimasi waktu penskalaan ulang memiliki fungsionalitas yang sama dengan EMR autotuning Amazon, jadi Anda tidak perlu menetapkan nilai empiris secara manual ke waktu restart.
> Anda masih dapat menggunakan EMR autotuning Amazon jika Anda menggunakan Amazon EMR 7.1.0 atau lebih rendah.

7.2.0 and higher

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

Untuk mengaktifkan pelacakan ini, gunakan perintah berikut:

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

Berikut ini adalah konfigurasi terkait untuk estimasi waktu penskalaan ulang.

Konfigurasi	Diperlukan	Default	Deskripsi
job.autoscaler.restart.time-tracking.enabled	Tidak	False	Menunjukkan apakah Flink Autoscaler harus secara otomatis menyetel konfigura si dari waktu ke waktu untuk mengoptimalkan dessisi penskalaan. Perhatikan bahwa Autoscaler hanya dapat melakukan autotune parameter Autoscaler. `restart.time`
job.autoscaler.restart.time	Tidak	5m	Waktu restart yang diharapka n yang EKS digunakan EMR Amazon hingga operator dapat menentukan waktu restart aktual dari penskalaan sebelumnya.
job.autoscaler.restart.time-tracking.limit	Tidak	15m	Waktu restart maksimum yang diamati saat

Konfigurasi	Diperlukan	Default	Deskripsi
			`job.autoscaler.restart.time-tracking.enabled` diatur ke `true`.

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

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

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

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

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

  executionRoleArn: <JOB ARN>
  emrReleaseLabel: emr-7.2.0-flink-latest
  jobManager:
    highAvailabilityEnabled: false
    storageDir: s3://<s3_bucket>/flink/autoscaling/ha/
    replicas: 1
    resource:
      memory: "1024m"
      cpu: 0.5
  taskManager:
    resource:
```

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

Untuk mensimulasikan tekanan balik, gunakan spesifikasi penerapan berikut.

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

Unggah skrip Python berikut ke bucket S3 Anda.

```
import logging
import sys
import time
import random

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

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

def create_backpressure(i):
```

```
    time.sleep(2)
    return i

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

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

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

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

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

Dapatkan nama pod pemimpin Anda.

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

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

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

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

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

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

7.0.0 and 7.1.0

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

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

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

Untuk mengaktifkan autotuning, Anda harus menyelesaikan yang berikut ini:

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

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

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

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

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

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

    jobmanager.scheduler: adaptive

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

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

Untuk mensimulasikan tekanan balik, gunakan spesifikasi penerapan berikut.

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

Unggah skrip Python berikut ke bucket S3 Anda.

```
import logging
import sys
import time
import random

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

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

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

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

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

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

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

Pertama dapatkan nama pod pemimpin Anda.

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

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

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

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

Anda akan melihat log yang mirip dengan yang berikut ini.

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

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

Pemeliharaan dan pemecahan masalah

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

Migrasi aplikasi Flink

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

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

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

Mode upgrade

Pengenalan opsional

Tanpa kewarganegaraan

Peningkatan aplikasi stateless dari status kosong.

Keadaan terakhir

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

Savepoint

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

Memecahkan masalah

Bagian ini menjelaskan cara memecahkan masalah dengan Amazon EMR di EKS. Untuk informasi tentang cara memecahkan masalah umum dengan Amazon EMR, lihat Memecahkan masalah klaster di Panduan Manajemen EMR Amazon.

- Pekerjaan pemecahan masalah yang menggunakan PersistentVolumeClaims (PVC)
- Memecahkan masalah Amazon EMR pada penskalaan otomatis vertikal EKS
- Memecahkan masalah Amazon EMR pada operator EKS

Memecahkan masalah Apache Flink di Amazon EMR di EKS

Pemetaan sumber daya tidak ditemukan saat menginstal bagan Helm

Anda mungkin menemukan pesan galat berikut saat menginstal bagan Helm.

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

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

ensure CRDs are installed first].
```

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

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

AWS layananakses ditolak kesalahan

Jika Anda melihat access denied kesalahan, konfirmasikan bahwa peran IAM untuk `operatorExecutionRoleArn` dalam `values.yaml` file bagan Helm memiliki izin yang benar. Pastikan juga peran IAM `executionRoleArn` di bawah `FlinkDeployment` spesifikasi Anda memiliki izin yang benar.

`FlinkDeployment`macet

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

1. Edit proses penerapan.

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

2. Hapus finalizer ini.

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

3. Hapus penyebaran.

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

Rilis yang didukung untuk EMR Amazon EKS dengan Apache Flink

Apache Flink tersedia dengan Amazon berikut EMR pada EKS rilis. Untuk informasi tentang semua rilis yang tersedia, lihatAmazon EMR pada EKS rilis.

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

Label rilis	Java	Flink	Operator Flink
emr-6.13.0-flink-k8s-operator-latest	11	-	1.5.0

Menjalankan pekerjaan dengan Amazon EMR di EKS

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

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

> **Note**
>
> Sebelum Anda mengirimkan pekerjaan dengan Amazon EMR di EKS, Anda harus menyelesaikan langkah-langkahnya. Menyiapkan Amazon EMR di EKS

Topik

Menjalankan pekerjaan Spark dengan `StartJobRun`

Topik

- Kirim pekerjaan yang dijalankan dengan StartJobRun

Menyiapkan Amazon EMR di EKS

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

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

1. Instal AWS CLI
2. Instal `eksctl`
3. Siapkan EKS cluster Amazon
4. Aktifkan akses cluster untuk Amazon EMR di EKS
5. Aktifkan IAM Peran untuk Akun Layanan (IRSA) di EKS klaster
6. Untuk membuat peran eksekusi tugas
7. Perbarui kebijakan kepercayaan dari peran eksekusi tugas
8. Berikan pengguna akses ke Amazon EMR di EKS
9. Daftarkan EKS cluster Amazon dengan Amazon EMR

Instal AWS CLI

Anda dapat menginstal versi terbaru AWS CLI untuk macOS, Linux, atau Windows.

> **Important**
>
> Untuk mengatur EMR AmazonEKS, Anda harus AWS CLI menginstal versi terbaru.

Untuk menginstal atau memperbarui AWS CLI untuk macOS

1. Jika saat ini Anda telah AWS CLI menginstal, tentukan versi mana yang telah Anda instal.

   ```
   aws --version
   ```

2. Jika Anda memiliki versi sebelumnya AWS CLI, maka gunakan perintah berikut untuk menginstal AWS CLI versi terbaru 2. Untuk opsi penginstalan lainnya, atau untuk memutakhirkan versi 2 yang saat ini diinstal, lihat Memutakhirkan AWS CLI versi 2 di macOS.

   ```
   curl "https://awscli.amazonaws.com/AWSCLIV2.pkg" -o "AWSCLIV2.pkg"
   sudo installer -pkg AWSCLIV2.pkg -target /
   ```

   Jika Anda tidak dapat menggunakan AWS CLI versi 2, pastikan Anda memiliki versi terbaru dari AWS CLI versi 1 yang diinstal menggunakan perintah berikut.

   ```
   pip3 install awscli --upgrade --user
   ```

Untuk menginstal atau memperbarui AWS CLI untuk Linux

1. Jika saat ini Anda telah AWS CLI menginstal, tentukan versi mana yang telah Anda instal.

   ```
   aws --version
   ```

2. Jika Anda memiliki versi sebelumnya AWS CLI, maka gunakan perintah berikut untuk menginstal AWS CLI versi terbaru 2. Untuk opsi penginstalan lainnya, atau untuk memutakhirkan versi 2 yang saat ini diinstal, lihat Memutakhirkan AWS CLI versi 2 di Linux.

   ```
   curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
   unzip awscliv2.zip
   sudo ./aws/install
   ```

   Jika Anda tidak dapat menggunakan AWS CLI versi 2, pastikan Anda memiliki versi terbaru dari AWS CLI versi 1 yang diinstal menggunakan perintah berikut.

   ```
   pip3 install --upgrade --user awscli
   ```

Untuk menginstal atau memperbarui AWS CLI untuk Windows

1. Jika saat ini Anda telah AWS CLI menginstal, tentukan versi mana yang telah Anda instal.

   ```
   aws --version
   ```

2. Jika Anda memiliki versi sebelumnya AWS CLI, maka gunakan perintah berikut untuk menginstal AWS CLI versi terbaru 2. Untuk opsi penginstalan lainnya, atau untuk memutakhirkan versi 2 yang saat ini diinstal, lihat Memutakhirkan AWS CLI versi 2 di Windows.

   1. Unduh AWS CLI MSI installer untuk Windows (64-bit) di https://awscli.amazonaws.com/AWSCLIV2 .msi
   2. Jalankan MSI installer yang diunduh dan ikuti instruksi di layar. Secara default, AWS CLI instalasi keC:\Program Files\Amazon\AWSCLIV2.

   Jika Anda tidak dapat menggunakan AWS CLI versi 2, pastikan Anda memiliki versi terbaru dari AWS CLI versi 1 yang diinstal menggunakan perintah berikut.

   ```
   pip3 install --user --upgrade awscli
   ```

Konfigurasikan AWS CLI kredensyal Anda

Baik eksctl dan AWS CLI mengharuskan Anda memiliki AWS kredensil yang dikonfigurasi di lingkungan Anda. Perintah `aws configure` adalah cara tercepat untuk mengatur instalasi AWS CLI Anda untuk penggunaan umum.

```
$ aws configure
AWS Access Key ID [None]: <AKIAIOSFODNN7EXAMPLE>
AWS Secret Access Key [None]: <wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY>
Default region name [None]: <region-code>
Default output format [None]: <json>
```

Saat Anda mengetik perintah ini, AWS CLI akan meminta Anda untuk empat informasi: Kunci akses, kunci akses rahasia, AWS Wilayah, dan format output. Informasi ini disimpan dalam profil (kumpulan pengaturan) bernama `default`. Profil ini digunakan saat Anda menjalankan perintah kecuali Anda menentukan perintah lain. Untuk informasi selengkapnya, lihat Mengonfigurasi AWS CLI dalam Panduan AWS Command Line Interface Pengguna.

Instal `eksctl`

Instal versi terbaru utilitas baris `eksctl` perintah di macOS, Linux, atau Windows. Untuk informasi lebih lanjut, lihat https://eksctl.io/.

> **Important**
>
> Kami menyarankan Anda mengunduh eksctl terbaru, karena beberapa fungsi EMR di Amazon EKS memerlukan versi yang lebih baru. Untuk informasi selengkapnya, lihat Instal `eksctl`.

Untuk menginstal atau meningkatkan eksctl di macOS menggunakan Homebrew

Cara termudah untuk memulai dengan Amazon EKS dan macOS adalah dengan menginstal eksctl dengan Homebrew. Resep eksctl Homebrew menginstal eksctl dan dependensi lain yang diperlukan untuk Amazon, seperti kubectl. EKS Resep juga menginstal aws-iam-authenticator, yang diperlukan jika Anda tidak menginstal AWS CLI versi 1.16.156 atau yang lebih baru.

1. Jika Anda belum menginstal Homebrew di macOS, instal dengan perintah berikut.

   ```
   /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
   master/install.sh)"
   ```

2. Instal tap Weaveworks Homebrew.

   ```
   brew tap weaveworks/tap
   ```

3. 1. Menginstal atau meningkatkan eksctl.
      - Instal `eksctl` dengan perintah berikut.

        ```
        brew install weaveworks/tap/eksctl
        ```

      - Jika `eksctl` telah diinstal, jalankan perintah berikut untuk meningkatkan.

        ```
        brew upgrade eksctl & brew link --overwrite eksctl
        ```

   2. Uji apakah instalasi Anda berhasil dengan perintah berikut. Anda harus memiliki `eksctl` versi 0.34.0 atau lebih baru.

```
eksctl version
```

Untuk menginstal atau meningkatkan **`eksctl`** di Linux menggunakan **`curl`**

1. Unduh dan ekstrak rilis terbaru dari eksctl dengan perintah berikut.

```
curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/
download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp
```

2. Pindahkan biner yang diekstrak ke /usr/local/bin.

```
sudo mv /tmp/eksctl /usr/local/bin
```

3. Uji apakah instalasi Anda berhasil dengan perintah berikut. Anda harus memiliki `eksctl` versi 0.34.0 atau lebih baru.

```
eksctl version
```

Untuk menginstal atau meningkatkan **`eksctl`** pada Windows menggunakan Chocolatey

1. Jika Anda belum menginstal Chocolatey pada sistem Windows Anda, lihat Menginstal Chocolatey.
2. Pasang atau tingkatkan `eksctl`.
   - Instal biner dengan perintah berikut.

```
choco install -y eksctl
```

   - Jika biner telah diinstal, jalankan perintah berikut untuk meningkatkan:

```
choco upgrade -y eksctl
```

3. Uji apakah instalasi Anda berhasil dengan perintah berikut. Anda harus memiliki `eksctl` versi 0.34.0 atau lebih baru.

```
eksctl version
```

Siapkan EKS cluster Amazon

Amazon EKS adalah layanan terkelola yang memudahkan Anda menjalankan Kubernetes AWS tanpa perlu menginstal, mengoperasikan, dan memelihara bidang kontrol atau node Kubernetes Anda sendiri. Ikuti langkah-langkah yang diuraikan di bawah ini untuk membuat cluster Kubernetes baru dengan node di Amazon. EKS

Prasyarat

> **Important**
>
> Sebelum Anda membuat EKS klaster Amazon, lengkapi persyaratan dan pertimbangan Amazon EKS VPC dan subnet di Panduan EKS Pengguna Amazon untuk memastikan bahwa EKS klaster Amazon Anda berfungsi dan diskalakan seperti yang diharapkan.

Anda harus menginstal dan mengonfigurasi alat dan sumber daya berikut yang Anda perlukan untuk membuat dan mengelola EKS klaster Amazon:

- Versi terbaru dari AWS CLI.
- `kubectl`versi 1.20 atau yang lebih baru.
- Versi terbaru dari`eksctl`.

Untuk informasi selengkapnya, lihatInstal AWS CLI, Menginstal `kubectl`, danInstal `eksctl`.

Buat EKS cluster Amazon menggunakan **`eksctl`**

Ambil langkah-langkah berikut untuk membuat EKS cluster Amazon menggunakan`eksctl`.

> **Important**
>
> Untuk memulai dengan cepat, Anda dapat membuat EKS cluster dan node dengan pengaturan default. Tetapi untuk penggunaan produksi, kami menyarankan Anda menyesuaikan pengaturan untuk cluster dan node untuk memenuhi persyaratan spesifik Anda. Untuk daftar semua pengaturan dan opsi, jalankan perintah`eksctl create cluster -h`. Untuk informasi selengkapnya, lihat Membuat dan Mengelola Cluster dalam `eksctl` dokumentasi.

1. Buat EC2 key pair Amazon.

   Jika Anda tidak memiliki key pair yang ada, Anda dapat menjalankan perintah berikut untuk membuat key pair baru. Ganti `us-west-2` dengan Region tempat Anda ingin membuat cluster Anda.

   ```
   aws ec2 create-key-pair --region us-west-2 --key-name myKeyPair
   ```

   Simpan output yang dikembalikan dalam file di komputer lokal Anda. Untuk informasi selengkapnya, lihat Membuat atau mengimpor key pair di Panduan EC2 Pengguna Amazon untuk Instans Linux.

   > **Note**
   >
   > Sebuah key pair tidak diperlukan untuk membuat sebuah EKS cluster. Tetapi menentukan key pair memungkinkan Anda untuk SSH ke node setelah mereka dibuat. Anda dapat menentukan key pair hanya ketika Anda membuat grup node.

2. Buat klaster EKS.

   Jalankan perintah berikut untuk membuat EKS cluster dan node. Ganti *my-cluster* and *myKeyPair* dengan nama cluster dan nama key pair Anda sendiri. Ganti *us-west-2* dengan Wilayah tempat Anda ingin membuat cluster Anda. Untuk informasi selengkapnya tentang Wilayah yang EKS didukung Amazon, lihat titik akhir dan kuota Amazon Elastic Kubernetes Service.

   ```
   eksctl create cluster \
   --name my-cluster \
   --region us-west-2 \
   --with-oidc \
   --ssh-access \
   --ssh-public-key myKeyPair \
   --instance-types=m5.xlarge \
   --managed
   ```

   > **Important**
   >
   > Saat membuat EKS cluster, gunakan m5.xlarge sebagai tipe instance, atau jenis instance lainnya dengan memori dan memori yang lebih tinggiCPU. Menggunakan tipe

instans dengan memori lebih rendah CPU atau dibandingkan dengan m5.xlarge dapat menyebabkan kegagalan pekerjaan karena sumber daya yang tersedia di cluster tidak mencukupi. Untuk melihat semua sumber daya yang dibuat, lihat tumpukan bernama `eksctl-my-cluster-cluster` di konsol AWS Cloud Formation.

Proses pembuatan cluster dan node membutuhkan waktu beberapa menit. Anda akan melihat beberapa baris output ketika cluster dan node dibuat. Contoh berikut menunjukkan baris terakhir output.

```
...
[#]  EKS cluster "my-cluster" in "us-west-2" region is ready
```

`eksctl`membuat file `kubectl` konfigurasi di `~/.kube` atau menambahkan konfigurasi cluster baru dalam file konfigurasi yang ada di. `~/.kube`

3. Melihat dan memvalidasi sumber daya

   Jalankan perintah berikut untuk melihat node cluster Anda.

   ```
   kubectl get nodes -o wide
   ```

   Berikut ini menunjukkan contoh output.

   ```
   Amazon EC2 node output

   NAME                                           STATUS   ROLES    AGE    VERSION
             INTERNAL-IP      EXTERNAL-IP     OS-IMAGE         KERNEL-VERSION
            CONTAINER-RUNTIME
   ip-192-168-12-49.us-west-2.compute.internal    Ready    none     6m7s
    v1.18.9-eks-d1db3c   192.168.12.49    52.35.116.65    Amazon Linux 2
    4.14.209-160.335.amzn2.x86_64   docker://19.3.6
   ip-192-168-72-129.us-west-2.compute.internal   Ready    none     6m4s
    v1.18.9-eks-d1db3c   192.168.72.129   44.242.140.21   Amazon Linux 2
    4.14.209-160.335.amzn2.x86_64   docker://19.3.6
   ```

   Untuk informasi selengkapnya, lihat Melihat node.

   Gunakan perintah berikut untuk melihat beban kerja yang berjalan di cluster Anda.

```
kubectl get pods --all-namespaces -o wide
```

Berikut ini menunjukkan contoh output.

```
Amazon EC2 output

NAMESPACE     NAME                       READY   STATUS    RESTARTS   AGE     IP
               NODE                                              NOMINATED NODE
 READINESS GATES
kube-system   aws-node-6ctpm             1/1     Running   0          7m43s
 192.168.72.129   ip-192-168-72-129.us-west-2.compute.internal   none
 none
kube-system   aws-node-cbntg             1/1     Running   0          7m46s
 192.168.12.49    ip-192-168-12-49.us-west-2.compute.internal    none
 none
kube-system   coredns-559b5db75d-26t47   1/1     Running   0          14m
 192.168.78.81    ip-192-168-72-129.us-west-2.compute.internal   none
 none
kube-system   coredns-559b5db75d-9rvnk   1/1     Running   0          14m
 192.168.29.248   ip-192-168-12-49.us-west-2.compute.internal    none
 none
kube-system   kube-proxy-l8pbd           1/1     Running   0          7m46s
 192.168.12.49    ip-192-168-12-49.us-west-2.compute.internal    none
 none
kube-system   kube-proxy-zh85h           1/1     Running   0          7m43s
 192.168.72.129   ip-192-168-72-129.us-west-2.compute.internal   none
 none
```

Untuk informasi selengkapnya tentang apa yang Anda lihat di sini, lihat Melihat beban kerja.

Buat EKS cluster menggunakan AWS Management Console dan AWS CLI

Anda juga dapat AWS CLI menggunakan AWS Management Console dan membuat EKS cluster. Ikuti langkah-langkah di Memulai Amazon EKS — AWS Management Console dan AWS CLI. Cara ini memberi Anda visibilitas tentang bagaimana setiap sumber daya dibuat untuk EKS cluster dan bagaimana sumber daya berinteraksi satu sama lain.

> **Important**
>
> Saat membuat node untuk EKS cluster, gunakan m5.xlarge sebagai tipe instance, atau jenis instance lainnya dengan memori dan memori yang lebih tinggiCPU.

Buat EKS cluster dengan AWS Fargate

Anda juga dapat membuat EKS klaster dengan pod yang berjalan AWS Fargate.

1. Untuk membuat EKS klaster dengan pod yang berjalan di Fargate, ikuti langkah-langkah yang diuraikan di Memulai dengan menggunakan AWS Fargate Amazon. EKS

   > **Note**
   >
   > EMRAmazon EKS membutuhkan Core DNS untuk menjalankan pekerjaan di EKS cluster. Jika Anda ingin menjalankan pod Anda hanya di Fargate, Anda harus mengikuti langkah-langkah di Memperbarui Core. DNS

2. Jalankan perintah berikut untuk melihat node cluster Anda.

   ```
   kubectl get nodes -o wide
   ```

   Berikut ini menunjukkan contoh keluaran Fargate.

   ```
   Fargate node output

   NAME                                                  STATUS   ROLES    AGE
      VERSION              INTERNAL-IP       EXTERNAL-IP   OS-IMAGE         KERNEL-
   VERSION                   CONTAINER-RUNTIME
   fargate-ip-192-168-141-147.us-west-2.compute.internal   Ready    none
    8m3s    v1.18.8-eks-7c9bda   192.168.141.147   none          Amazon Linux 2
    4.14.209-160.335.amzn2.x86_64   containerd://1.3.2
   fargate-ip-192-168-164-53.us-west-2.compute.internal    Ready    none
    7m30s   v1.18.8-eks-7c9bda   192.168.164.53    none          Amazon Linux 2
    4.14.209-160.335.amzn2.x86_64   containerd://1.3.2
   ```

   Untuk informasi selengkapnya, lihat Melihat node.

3. Jalankan perintah berikut untuk melihat beban kerja yang berjalan di cluster Anda.

```
kubectl get pods --all-namespaces -o wide
```

Berikut ini menunjukkan contoh keluaran Fargate.

```
Fargate output

NAMESPACE     NAME                       READY   STATUS    RESTARTS   AGE   IP
           NODE                                                    NOMINATED NODE
 READINESS GATES
kube-system   coredns-69dfb8f894-9z95l   1/1     Running   0          18m
 192.168.164.53    fargate-ip-192-168-164-53.us-west-2.compute.internal    none
         none
kube-system   coredns-69dfb8f894-c8v66   1/1     Running   0          18m
 192.168.141.147   fargate-ip-192-168-141-147.us-west-2.compute.internal   none
         none
```

Untuk informasi selengkapnya, lihat Melihat beban kerja.

Aktifkan akses cluster untuk Amazon EMR di EKS

Aktifkan akses cluster menggunakan EKS Access Entry (disarankan)

> **Note**
>
> `aws-auth` ConfigMap Itu sudah usang. Metode yang disarankan untuk mengelola akses ke Kubernetes APIs adalah Access Entries.

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

> **Note**
>
> EMRIntegrasi Amazon dengan Amazon hanya EKS CAM didukung untuk Amazon baru EMR di cluster EKS virtual. Anda tidak dapat memigrasi klaster virtual yang ada untuk menggunakan integrasi ini.

Prasyarat

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

Pengaturan

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

- Pastikan `authenticationMode` EKS klaster Amazon Anda disetel ke`API_AND_CONFIG_MAP`.

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

  Jika belum, atur `authenticationMode` ke`API_AND_CONFIG_MAP`.

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

  Untuk informasi selengkapnya tentang mode autentikasi, lihat Mode otentikasi klaster.
- Pastikan bahwa IAMperan yang Anda gunakan untuk menjalankan `CreateVirtualCluster` dan `DeleteVirtualCluster` API operasi juga memiliki izin berikut:

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

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

Konsep dan terminologi

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

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

Aktifkan akses cluster menggunakan **`aws-auth`**

Anda harus mengizinkan Amazon EMR EKS mengakses namespace tertentu di klaster Anda dengan mengambil tindakan berikut: membuat peran Kubernetes, mengikat peran tersebut ke pengguna Kubernetes, dan memetakan pengguna Kubernetes dengan peran terkait layanan. `AWSServiceRoleForAmazonEMRContainers` Tindakan ini otomatis `eksctl` ketika perintah pemetaan IAM identitas digunakan `emr-containers` sebagai nama layanan. Anda dapat melakukan operasi ini dengan mudah menggunakan perintah berikut.

```
eksctl create iamidentitymapping \
    --cluster my_eks_cluster \
```

```
    --namespace kubernetes_namespace \
    --service-name "emr-containers"
```

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

> **Important**
>
> Anda harus mengunduh eksctl terbaru menggunakan langkah Instal `eksctl` sebelumnya untuk menggunakan fungsi ini.

Langkah-langkah manual untuk mengaktifkan akses klaster untuk Amazon EMR di EKS

Anda juga dapat menggunakan langkah-langkah manual berikut untuk mengaktifkan akses klaster untuk EMR AmazonEKS.

1. Buat peran Kubernetes di namespace tertentu

   Amazon EKS 1.22 - 1.29

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

```
namespace=my-namespace
cat - >>EOF | kubectl apply -f - >>namespace "${namespace}"
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: emr-containers
  namespace: ${namespace}
rules:
  - apiGroups: [""]
    resources: ["namespaces"]
    verbs: ["get"]
  - apiGroups: [""]
    resources: ["serviceaccounts", "services", "configmaps", "events", "pods",
 "pods/log"]
```

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

Amazon EKS 1.21 and below

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

```
namespace=my-namespace
cat - >>EOF | kubectl apply -f - >>namespace "${namespace}"
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: emr-containers
  namespace: ${namespace}
rules:
  - apiGroups: [""]
```

```
    resources: ["namespaces"]
    verbs: ["get"]
  - apiGroups: [""]
    resources: ["serviceaccounts", "services", "configmaps", "events", "pods",
 "pods/log"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
 "deletecollection", "annotate", "patch", "label"]
  - apiGroups: [""]
    resources: ["secrets"]
    verbs: ["create", "patch", "delete", "watch"]
  - apiGroups: ["apps"]
    resources: ["statefulsets", "deployments"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
 "annotate", "patch", "label"]
  - apiGroups: ["batch"]
    resources: ["jobs"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
 "annotate", "patch", "label"]
  - apiGroups: ["extensions"]
    resources: ["ingresses"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
 "annotate", "patch", "label"]
  - apiGroups: ["rbac.authorization.k8s.io"]
    resources: ["roles", "rolebindings"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
 "deletecollection", "annotate", "patch", "label"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
 "deletecollection", "annotate", "patch", "label"]
EOF
```

2. Buat ikatan peran Kubernetes yang dicakup ke namespace

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

```
namespace=my-namespace
```

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

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

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

   Opsi 1: Menggunakan **`eksctl`**

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

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

   Opsi 2: Tanpa menggunakan eksctl

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

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

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

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

```
apiVersion: v1
data:
  mapRoles: |
    - rolearn: arn:aws:iam::<your-account-id>:role/
AWSServiceRoleForAmazonEMRContainers
      username: emr-containers
    - ... <other previously existing role entries, if there's any>.
```

3. Simpan file , dan tutup editor teks Anda.

Aktifkan IAM Peran untuk Akun Layanan (IRSA) di EKS klaster

Fitur IAM peran untuk akun layanan tersedia di Amazon EKS versi 1.14 dan yang lebih baru dan untuk EKS cluster yang diperbarui ke versi 1.13 atau lebih baru pada atau setelah 3 September 2019. Untuk menggunakan fitur ini, Anda dapat memperbarui EKS cluster yang ada ke versi 1.14 atau yang lebih baru. Untuk informasi selengkapnya, lihat Memperbarui versi Kubernetes EKS klaster Amazon.

Jika klaster Anda mendukung IAM peran untuk akun layanan, ia memiliki penerbit OpenID Connect yang URL terkait dengannya. Anda dapat melihat ini URL di EKS konsol Amazon, atau Anda dapat menggunakan AWS CLI perintah berikut untuk mengambilnya.

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

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

Output yang diharapkan adalah sebagai berikut.

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

Untuk menggunakan IAM peran untuk akun layanan di klaster Anda, Anda harus membuat penyedia OIDC identitas menggunakan eksctl atau. AWS Management Console

Untuk membuat penyedia IAM OIDC identitas untuk klaster Anda dengan **`eksctl`**

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

```
eksctl version
```

Untuk informasi selengkapnya tentang menginstal atau meningkatkan eksctl, lihat Menginstal atau meningkatkan eksctl.

Buat penyedia OIDC identitas Anda untuk klaster Anda dengan perintah berikut. Ganti *`cluster_name`* dengan nilai Anda sendiri.

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

Untuk membuat penyedia IAM OIDC identitas untuk klaster Anda dengan AWS Management Console

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

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

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

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

1. Buka IAM konsol di https://console.aws.amazon.com/iam/.

2. Di panel navigasi, pilih Penyedia Identitas, lalu pilih Buat Penyedia.
    1. Untuk Jenis Penyedia, pilih Pilih jenis penyedia, lalu pilih OpenID Connect.
    2. Untuk Provider URL, tempel OIDC penerbit URL untuk klaster Anda.
    3. Untuk Audiens, ketik sts.amazonaws.com dan pilih Langkah Selanjutnya.
3. Verifikasi bahwa informasi penyedia sudah benar, kemudian pilih Buat untuk membuat penyedia identitas Anda.

Untuk membuat peran eksekusi tugas

Untuk menjalankan beban kerja EMR di AmazonEKS, Anda perlu membuat IAM peran. Kami menyebut peran ini sebagai peran eksekusi tugas dalam dokumentasi ini. Untuk informasi selengkapnya tentang cara membuat IAM peran, lihat Membuat IAM peran di Panduan IAM pengguna.

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

Kebijakan berikut untuk peran eksekusi pekerjaan memungkinkan akses ke target sumber daya, Amazon S3, dan. CloudWatch Izin ini diperlukan untuk memantau tugas dan log akses. Untuk mengikuti proses yang sama menggunakan AWS CLI, Anda juga dapat mengatur peran menggunakan langkah-langkah di bagian Buat IAM Peran untuk pelaksanaan pekerjaan EMR di Amazon di EKS Workshop.

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

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListBucket"
```

```
            ],
            "Resource": "arn:aws:s3:::example-bucket"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents",
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:*:*:*"
            ]
        }
    ]
}
```

Untuk informasi selengkapnya, lihat Menggunakan peran eksekusi pekerjaan, Mengonfigurasi proses pekerjaan untuk menggunakan log S3, dan Mengonfigurasi proses pekerjaan untuk menggunakan CloudWatch Log.

Perbarui kebijakan kepercayaan dari peran eksekusi tugas

Ketika Anda menggunakan IAM Roles for Service Accounts (IRSA) untuk menjalankan lowongan pada namespace Kubernetes, administrator harus membuat hubungan kepercayaan antara peran eksekusi pekerjaan dan identitas akun layanan terkelola. EMR Hubungan kepercayaan dapat dibuat dengan memperbarui kebijakan kepercayaan dari peran pelaksanaan tugas. Perhatikan bahwa akun layanan EMR terkelola secara otomatis dibuat pada pengiriman pekerjaan, dicakup ke namespace tempat pekerjaan dikirimkan.

Jalankan perintah berikut untuk memperbarui kebijakan kepercayaan.

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

Untuk informasi selengkapnya, lihat Menggunakan peran eksekusi tugas dengan Amazon EMR di EKS.

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

Berikan pengguna akses ke Amazon EMR di EKS

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

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

Membuat IAM kebijakan baru dan melampirkannya ke pengguna di konsol IAM

Buat IAM kebijakan baru

1. Masuk ke AWS Management Console dan buka IAM konsol di https://console.aws.amazon.com/iam/.
2. Di panel navigasi kiri IAM konsol, pilih Kebijakan.
3. Pada halaman Kebijakan, pilih Buat Kebijakan.
4. Di jendela Buat Kebijakan, arahkan ke JSON tab Edit. Buat dokumen kebijakan dengan satu atau beberapa JSON pernyataan seperti yang ditunjukkan pada contoh berikut prosedur ini. Selanjutnya, pilih Tinjau kebijakan.
5. Pada layar Tinjau Kebijakan, masukkan Nama kebijakanAnda, misalnya `AmazonEMROnEKSPolicy`. Masukkan deskripsi opsional, lalu pilih Buat kebijakan.

Lampirkan kebijakan ke pengguna atau peran

1. Masuk ke AWS Management Console dan buka IAM konsol di https://console.aws.amazon.com/iam/
2. Di panel navigasi, pilih Kebijakan.

3. Dalam daftar kebijakan, pilih kotak centang di samping kebijakan yang dibuat dalam bagian sebelumnya. Anda bisa memakai menu Filter dan kotak pencarian untuk memfilter daftar kebijakan.
4. Pilih Tindakan kebijakan, lalu pilih Lampirkan.
5. Pilih pengguna atau peran untuk melampirkan kebijakan. Anda bisa memakai menu Filter dan kotak pencarian untuk memfilter daftar entitas prinsipiel. Setelah memilih pengguna atau peran untuk melampirkan kebijakan, pilih Lampirkan kebijakan.

Izin untuk mengelolaklaster virtual

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

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

```
}
```

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

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

Untuk informasi selengkapnya, lihat Mengotomatiskan mengaktifkan akses klaster untuk Amazon EMR. EKS

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

Izin untuk mengirimkan tugas

Untuk mengirimkan pekerjaan di klaster virtual di AWS akun Anda, buat IAM kebijakan dengan izin berikut. Izin ini memungkinkan Anda untuk memulai, mendaftar, menjelaskan, dan membatalkan klaster virtual di akun Anda. Anda harus mempertimbangkan menambahkan izin untuk membuat

daftar atau menjelaskan klaster virtual, yang memungkinkan Anda untuk memeriksa keadaan klaster virtual sebelum mengirimkan tugas.

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

Izin untuk melakukan debug dan pemantauan

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

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

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "emr-containers:DescribeJobRun",
                "elasticmapreduce:CreatePersistentAppUI",
```

```
                "elasticmapreduce:DescribePersistentAppUI",
                "elasticmapreduce:GetPersistentAppUIPresignedURL"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:Get*",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams"
            ],
            "Resource": "*"
        }
    ]
}
```

Untuk informasi selengkapnya tentang cara mengonfigurasi job run untuk mendorong log ke Amazon S3 dan CloudWatch, lihat Mengonfigurasi proses pekerjaan untuk menggunakan log S3 dan Mengonfigurasi proses pekerjaan untuk menggunakan Log. CloudWatch

Daftarkan EKS cluster Amazon dengan Amazon EMR

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

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

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

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

Atau, Anda dapat membuat JSON file yang menyertakan parameter yang diperlukan untuk cluster virtual dan kemudian menjalankan `create-virtual-cluster` perintah dengan path ke JSON file. Untuk informasi selengkapnya, lihat Mengelola klaster virtual.

> **Note**
>
> Untuk memvalidasi keberhasilan pembuatan cluster virtual, lihat status cluster virtual menggunakan `list-virtual-clusters` operasi atau dengan membuka halaman Virtual Clusters di konsol Amazon. EMR

Kirim pekerjaan yang dijalankan dengan **`StartJobRun`**

Untuk mengirimkan pekerjaan yang dijalankan dengan file JSON dengan parameter tertentu

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

   ```
   {
     "name": "myjob",
     "virtualClusterId": "123456",
     "executionRoleArn": "iam_role_name_for_job_execution",
     "releaseLabel": "emr-6.2.0-latest",
     "jobDriver": {
       "sparkSubmitJobDriver": {
         "entryPoint": "entryPoint_location",
         "entryPointArguments": ["argument1", "argument2", ...],
   ```

```
      "sparkSubmitParameters": "--class <main_class> --conf
 spark.executor.instances=2 --conf spark.executor.memory=2G --conf
 spark.executor.cores=2 --conf spark.driver.cores=1"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.driver.memory":"2G"
         }
      }
    ],
    "monitoringConfiguration": {
      "persistentAppUI": "ENABLED",
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "my_log_group",
        "logStreamNamePrefix": "log_stream_prefix"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://my_s3_log_location"
      }
    }
  }
}
```

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

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

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

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

```
aws emr-containers start-job-run \
--virtual-cluster-id 123456 \
--name myjob \
--execution-role-arn execution-role-arn \
```

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

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

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

Menjalankan pekerjaan Spark dengan operator Spark

Amazon EMR merilis 6.10.0 dan lebih tinggi mendukung operator Kubernetes untuk Apache Spark, atau operator Spark, sebagai model pengiriman pekerjaan untuk Amazon di. EMR EKS Dengan operator Spark, Anda dapat menerapkan dan mengelola aplikasi Spark dengan runtime EMR rilis Amazon di kluster Amazon Anda sendiri. EKS Setelah Anda menyebarkan operator Spark di EKS cluster Amazon Anda, Anda dapat langsung mengirimkan aplikasi Spark dengan operator. Operator mengelola siklus hidup aplikasi Spark.

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

Topik

Menyiapkan operator Spark untuk Amazon di EMR EKS

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

- Instal AWS CLI— Jika Anda sudah menginstal AWS CLI, konfirmasikan bahwa Anda memiliki versi terbaru.
- Instal eksctl - eksctl adalah alat baris perintah yang Anda gunakan untuk berkomunikasi dengan Amazon. EKS
- Instal Helm — Manajer paket Helm untuk Kubernetes membantu Anda menginstal dan mengelola aplikasi di klaster Kubernetes Anda.
- Siapkan EKS klaster Amazon — Ikuti langkah-langkah untuk membuat klaster Kubernetes baru dengan node di Amazon. EKS
- Pilih gambar EMR dasar Amazon URI (rilis 6.10.0 atau lebih tinggi) — operator Spark didukung dengan EMR rilis Amazon 6.10.0 dan yang lebih tinggi.

Memulai dengan operator Spark untuk Amazon di EMR EKS

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

Instal operator Spark

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

1. Jika Anda belum melakukannya, selesaikan langkah-langkahnyaMenyiapkan operator Spark untuk Amazon di EMR EKS.
2. Otentikasi klien Helm Anda ke registri AmazonECR. Dalam perintah berikut, ganti *region-id* nilai dengan pilihan Anda Wilayah AWS, dan yang sesuai *ECR-registry-account* nilai untuk Wilayah dari Akun ECR registri Amazon berdasarkan Wilayah halaman.

   ```
   aws ecr get-login-password \
   --region region-id | helm registry login \
   --username AWS \
   --password-stdin ECR-registry-account.dkr.ecr.region-id.amazonaws.com
   ```

3. Instal operator Spark dengan perintah berikut.

   Untuk `--version` parameter bagan Helm, gunakan label EMR rilis Amazon Anda dengan `emr-` awalan dan akhiran tanggal dihapus. Misalnya, dengan `emr-6.12.0-java17-latest` rilis, tentukan`6.12.0-java17`. Contoh dalam perintah berikut menggunakan `emr-7.2.0-latest` rilis, sehingga menentukan `7.2.0` bagan Helm. `--version`

   ```
   helm install spark-operator-demo \
     oci://895885662937.dkr.ecr.region-id.amazonaws.com/spark-operator \
     --set emrContainers.awsRegion=region-id \
     --version 7.2.0 \
     --namespace spark-operator \
     --create-namespace
   ```

   Secara default, perintah membuat akun layanan `emr-containers-sa-spark-operator` untuk operator Spark. Untuk menggunakan akun layanan yang berbeda, berikan argumennya`serviceAccounts.sparkoperator.name`. Sebagai contoh:

   ```
   --set serviceAccounts.sparkoperator.name my-service-account-for-spark-operator
   ```

Jika Anda ingin menggunakan penskalaan otomatis vertikal dengan operator Spark, tambahkan baris berikut ke perintah instalasi untuk mengizinkan webhook untuk operator:

```
--set webhook.enable=true
```

4. Verifikasi bahwa Anda menginstal bagan Helm dengan `helm list` perintah:

```
helm list --namespace spark-operator -o yaml
```

`helm list`Perintah harus mengembalikan informasi rilis bagan Helm yang baru Anda gunakan:

```
app_version: v1beta2-1.3.8-3.1.1
chart: spark-operator-7.2.0
name: spark-operator-demo
namespace: spark-operator
revision: "1"
status: deployed
updated: 2023-03-14 18:20:02.721638196 +0000 UTC
```

5. Instalasi lengkap dengan opsi tambahan apa pun yang Anda butuhkan. Untuk informasi lebih lanjut, lihat `spark-on-k8s-operator`dokumentasi di GitHub.

Jalankan aplikasi Spark

Operator Spark didukung dengan Amazon EMR 6.10.0 atau lebih tinggi. Ketika Anda menginstal operator Spark, itu membuat akun layanan `emr-containers-sa-spark` untuk menjalankan aplikasi Spark secara default. Gunakan langkah-langkah berikut untuk menjalankan aplikasi Spark dengan operator Spark di Amazon EMR pada EKS 6.10.0 atau lebih tinggi.

1. Sebelum Anda dapat menjalankan aplikasi Spark dengan operator Spark, selesaikan langkah-langkah di Menyiapkan operator Spark untuk Amazon di EMR EKS dan. Instal operator Spark
2. Buat file `SparkApplication` definisi `spark-pi.yaml` dengan isi contoh berikut:

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: spark-pi
  namespace: spark-operator
spec:
```

```
  type: Scala
  mode: cluster
  image: "895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.10.0:latest"
  imagePullPolicy: Always
  mainClass: org.apache.spark.examples.SparkPi
  mainApplicationFile: "local:///usr/lib/spark/examples/jars/spark-examples.jar"
  sparkVersion: "3.3.1"
  restartPolicy:
    type: Never
  volumes:
    - name: "test-volume"
      hostPath:
        path: "/tmp"
        type: Directory
  driver:
    cores: 1
    coreLimit: "1200m"
    memory: "512m"
    labels:
      version: 3.3.1
    serviceAccount: emr-containers-sa-spark
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
  executor:
    cores: 1
    instances: 1
    memory: "512m"
    labels:
      version: 3.3.1
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
```

3. Sekarang, kirimkan aplikasi Spark dengan perintah berikut. Ini juga akan membuat `SparkApplication` objek bernama`spark-pi`:

```
kubectl apply -f spark-pi.yaml
```

4. Periksa peristiwa untuk `SparkApplication` objek dengan perintah berikut:

```
kubectl describe sparkapplication spark-pi --namespace spark-operator
```

Untuk informasi selengkapnya tentang mengirimkan aplikasi ke Spark melalui operator Spark, lihat Menggunakan a `SparkApplication` dalam dokumentasi pada. `spark-on-k8s-operator` GitHub

Gunakan Amazon S3 untuk penyimpanan

Untuk menggunakan Amazon S3 sebagai opsi penyimpanan file Anda, tambahkan konfigurasi berikut ke file Anda. YAML

```
hadoopConf:
# EMRFS filesystem
  fs.s3.customAWSCredentialsProvider:
 com.amazonaws.auth.WebIdentityTokenCredentialsProvider
  fs.s3.impl: com.amazon.ws.emr.hadoop.fs.EmrFileSystem
  fs.AbstractFileSystem.s3.impl: org.apache.hadoop.fs.s3.EMRFSDelegate
  fs.s3.buffer.dir: /mnt/s3
  fs.s3.getObject.initialSocketTimeoutMilliseconds: "2000"
  mapreduce.fileoutputcommitter.algorithm.version.emr_internal_use_only.EmrFileSystem:
 "2"
  mapreduce.fileoutputcommitter.cleanup-
failures.ignored.emr_internal_use_only.EmrFileSystem: "true"
sparkConf:
 # Required for EMR Runtime
 spark.driver.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/hadoop-
aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/share/aws/
emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/security/conf:/
usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-glue-datacatalog-
spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-serde.jar:/usr/share/aws/
sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/hadoop/extrajars/*
 spark.driver.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/lib/
native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/native
 spark.executor.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/hadoop-
aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/share/aws/
emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/security/conf:/
usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-glue-datacatalog-
spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-serde.jar:/usr/share/aws/
sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/hadoop/extrajars/*
 spark.executor.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/lib/
native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/native
```

Jika Anda menggunakan Amazon EMR rilis 7.2.0 dan yang lebih tinggi, konfigurasi disertakan secara default. Dalam hal ini, Anda dapat mengatur jalur file `s3://<bucket_name>/<file_path>` alih-alih `local://<file_path>` di YAML file aplikasi Spark.

Kemudian kirimkan aplikasi Spark seperti biasa.

Menggunakan penskalaan otomatis vertikal dengan operator Spark untuk Amazon di EMR EKS

Dimulai dengan Amazon EMR 7.0, Anda dapat menggunakan Amazon EMR pada penskalaan otomatis EKS vertikal untuk menyederhanakan manajemen sumber daya. Ini secara otomatis menyetel memori dan CPU sumber daya untuk beradaptasi dengan kebutuhan beban kerja yang Anda sediakan untuk aplikasi Amazon EMR Spark. Untuk informasi selengkapnya, lihat Menggunakan penskalaan otomatis vertikal dengan pekerjaan Amazon EMR Spark.

Bagian ini menjelaskan cara mengkonfigurasi operator Spark untuk menggunakan penskalaan otomatis vertikal.

Prasyarat

Sebelum melanjutkan, pastikan untuk menyelesaikan pengaturan berikut:

- Selesaikan langkah-langkah dalam Menyiapkan operator Spark untuk Amazon di EMR EKS.
- (Opsional) Jika sebelumnya Anda menginstal versi operator Spark yang lebih lama, hapus SparkApplication/ScheduledSparkApplication CRD.

```
kubectl delete crd sparkApplication
kubectl delete crd scheduledSparkApplication
```

- Selesaikan langkah-langkah dalam Instal operator Spark. Pada langkah 3, tambahkan baris berikut ke perintah instalasi untuk mengizinkan webhook untuk operator:

```
--set webhook.enable=true
```

- Selesaikan langkah-langkah dalam Menyiapkan penskalaan otomatis vertikal untuk Amazon EMR di EKS.
- Berikan akses ke file di lokasi Amazon S3 Anda:

  1. Beri anotasi akun layanan driver dan operator Anda dengan `JobExecutionRole` yang memiliki izin S3.

```
kubectl annotate serviceaccount -n spark-operator emr-containers-sa-spark
 eks.amazonaws.com/role-arn=JobExecutionRole
kubectl annotate serviceaccount -n spark-operator emr-containers-sa-spark-
operator eks.amazonaws.com/role-arn=JobExecutionRole
```

2. Perbarui kebijakan kepercayaan peran eksekusi pekerjaan Anda di namespace tersebut.

```
aws emr-containers update-role-trust-policy \
--cluster-name cluster \
--namespace ${Namespace}\
--role-name iam_role_name_for_job_execution
```

3. Edit kebijakan kepercayaan IAM peran peran eksekusi pekerjaan Anda dan perbarui `serviceaccount` dari `emr-containers-sa-spark-*-*-xxxx` ke`emr-containers-sa-*-*`.

```
{
    "Effect": "Allow",
    "Principal": {
        "Federated": "OIDC-provider"
    },
    "Action": "sts:AssumeRoleWithWebIdentity",
    "Condition": {
        "StringLike": {
            "OIDC": "system:serviceaccount:${Namespace}:emr-containers-sa-*"
        }
    }
}
```

4. Jika Anda menggunakan Amazon S3 sebagai penyimpanan file, tambahkan default berikut ke file yaml Anda.

```
hadoopConf:
# EMRFS filesystem
  fs.s3.customAWSCredentialsProvider:
 com.amazonaws.auth.WebIdentityTokenCredentialsProvider
  fs.s3.impl: com.amazon.ws.emr.hadoop.fs.EmrFileSystem
  fs.AbstractFileSystem.s3.impl: org.apache.hadoop.fs.s3.EMRFSDelegate
  fs.s3.buffer.dir: /mnt/s3
  fs.s3.getObject.initialSocketTimeoutMilliseconds: "2000"
```

```
 mapreduce.fileoutputcommitter.algorithm.version.emr_internal_use_only.EmrFileSystem:
 "2"
  mapreduce.fileoutputcommitter.cleanup-
failures.ignored.emr_internal_use_only.EmrFileSystem: "true"
sparkConf:
 # Required for EMR Runtime
 spark.driver.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/hadoop-
aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/share/
aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/
security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-
glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-
serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/
hadoop/extrajars/*
 spark.driver.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/
lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/
native
 spark.executor.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/
hadoop-aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/
share/aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/
security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-
glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-
serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/
hadoop/extrajars/*
 spark.executor.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-
lzo/lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/
native
```

Jalankan pekerjaan dengan penskalaan otomatis vertikal pada operator Spark

Sebelum Anda dapat menjalankan aplikasi Spark dengan operator Spark, Anda harus menyelesaikan langkah-langkahnya. Prasyarat

Untuk menggunakan penskalaan otomatis vertikal dengan operator Spark, tambahkan konfigurasi berikut ke driver untuk spesifikasi Aplikasi Spark Anda untuk mengaktifkan penskalaan otomatis vertikal:

```
dynamicSizing:
  mode: Off
  signature: "my-signature"
```

Konfigurasi ini memungkinkan penskalaan otomatis vertikal dan merupakan konfigurasi tanda tangan wajib yang memungkinkan Anda memilih tanda tangan untuk pekerjaan Anda.

Untuk informasi selengkapnya tentang konfigurasi dan nilai parameter, lihat Mengonfigurasi penskalaan otomatis vertikal untuk Amazon. EMR EKS Secara default, pekerjaan Anda dikirimkan dalam mode Monitoring-Only Off dari penskalaan otomatis vertikal. Status pemantauan ini memungkinkan Anda menghitung dan melihat rekomendasi sumber daya tanpa melakukan penskalaan otomatis. Untuk informasi selengkapnya, lihat Mode penskalaan otomatis vertikal.

Berikut ini adalah file `SparkApplication` definisi sampel bernama `spark-pi.yaml` dengan konfigurasi yang diperlukan untuk menggunakan penskalaan otomatis vertikal.

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: spark-pi
  namespace: spark-operator
spec:
  type: Scala
  mode: cluster
  image: "895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-7.2.0:latest"
  imagePullPolicy: Always
  mainClass: org.apache.spark.examples.SparkPi
  mainApplicationFile: "local:///usr/lib/spark/examples/jars/spark-examples.jar"
  sparkVersion: "3.4.1"
  dynamicSizing:
    mode: Off
    signature: "my-signature"
  restartPolicy:
    type: Never
  volumes:
    - name: "test-volume"
      hostPath:
        path: "/tmp"
        type: Directory
  driver:
    cores: 1
    coreLimit: "1200m"
    memory: "512m"
    labels:
      version: 3.4.1
    serviceAccount: emr-containers-sa-spark
    volumeMounts:
```

```
      - name: "test-volume"
        mountPath: "/tmp"
  executor:
    cores: 1
    instances: 1
    memory: "512m"
    labels:
      version: 3.4.1
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
```

Sekarang, kirimkan aplikasi Spark dengan perintah berikut. Ini juga akan membuat `SparkApplication` objek bernama`spark-pi`:

```
kubectl apply -f spark-pi.yaml
```

Untuk informasi selengkapnya tentang mengirimkan aplikasi ke Spark melalui operator Spark, lihat Menggunakan a `SparkApplication` dalam dokumentasi pada. `spark-on-k8s-operator` GitHub

Memverifikasi fungsionalitas penskalaan otomatis vertikal

Untuk memverifikasi bahwa penskalaan otomatis vertikal berfungsi dengan benar untuk pekerjaan yang dikirimkan, gunakan kubectl untuk mendapatkan sumber daya `verticalpodautoscaler` kustom dan melihat rekomendasi penskalaan Anda.

```
kubectl get verticalpodautoscalers --all-namespaces \
-l=emr-containers.amazonaws.com/dynamic.sizing.signature=my-signature
```

Output dari kueri ini harus menyerupai yang berikut:

```
NAMESPACE        NAME                                                    MODE
 CPU    MEM         PROVIDED   AGE
spark-operator   ds-p73j6mkosvc4xeb3gr7x4xol2bfcw5evqimzqojrlysvj3giozuq-vpa   Off
     580026651   True       15m
```

Jika output Anda tidak terlihat serupa atau berisi kode kesalahan, lihat langkah-langkah Memecahkan masalah Amazon EMR pada penskalaan otomatis vertikal EKS untuk membantu menyelesaikan masalah.

Untuk menghapus pod dan aplikasi, jalankan perintah berikut:

```
kubectl delete sparkapplication spark-pi
```

Menghapus instalan operator Spark untuk Amazon di EMR EKS

Gunakan langkah-langkah berikut untuk menghapus instalan operator Spark.

1. Hapus operator Spark menggunakan namespace yang benar. Untuk contoh ini, namespace adalah. `spark-operator-demo`

   ```
   helm uninstall spark-operator-demo -n spark-operator
   ```

2. Hapus akun layanan operator Spark:

   ```
   kubectl delete sa emr-containers-sa-spark-operator -n spark-operator
   ```

3. Hapus operator Spark `CustomResourceDefinitions` (CRDs):

   ```
   kubectl delete crd sparkapplications.sparkoperator.k8s.io
   kubectl delete crd scheduledsparkapplications.sparkoperator.k8s.io
   ```

Keamanan dan operator Spark dengan Amazon aktif EMR EKS

Topik

- Menyiapkan izin akses cluster dengan kontrol akses berbasis peran () RBAC
- Menyiapkan izin akses klaster dengan IAM peran untuk akun layanan () IRSA

Menyiapkan izin akses cluster dengan kontrol akses berbasis peran () RBAC

Untuk menyebarkan operator Spark, EMR Amazon EKS membuat dua peran dan akun layanan untuk operator Spark dan aplikasi Spark.

Topik

- Akun dan peran layanan operator
- Akun dan peran layanan Spark

Akun dan peran layanan operator

Amazon EMR on EKS membuat akun layanan operator dan peran `SparkApplications` untuk mengelola pekerjaan Spark dan sumber daya lain seperti layanan.

Nama default untuk akun layanan ini adalah`emr-containers-sa-spark-operator`.

Aturan berikut berlaku untuk peran layanan ini:

```
 rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - services
  - configmaps
  - secrets
  verbs:
  - create
  - get
  - delete
  - update
- apiGroups:
  - extensions
  - networking.k8s.io
  resources:
  - ingresses
  verbs:
  - create
  - get
  - delete
- apiGroups:
  - ""
  resources:
  - nodes
  verbs:
  - get
- apiGroups:
  - ""
```

```
  resources:
  - events
  verbs:
  - create
  - update
  - patch
- apiGroups:
  - ""
  resources:
  - resourcequotas
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - apiextensions.k8s.io
  resources:
  - customresourcedefinitions
  verbs:
  - create
  - get
  - update
  - delete
- apiGroups:
  - admissionregistration.k8s.io
  resources:
  - mutatingwebhookconfigurations
  - validatingwebhookconfigurations
  verbs:
  - create
  - get
  - update
  - delete
- apiGroups:
  - sparkoperator.k8s.io
  resources:
  - sparkapplications
  - sparkapplications/status
  - scheduledsparkapplications
  - scheduledsparkapplications/status
  verbs:
  - "*"
  {{- if .Values.batchScheduler.enable }}
  # required for the `volcano` batch scheduler
```

```
- apiGroups:
  - scheduling.incubator.k8s.io
  - scheduling.sigs.dev
  - scheduling.volcano.sh
  resources:
  - podgroups
  verbs:
  - "*"
  {{- end }}
  {{ if .Values.webhook.enable }}
- apiGroups:
  - batch
  resources:
  - jobs
  verbs:
  - delete
  {{- end }}
```

Akun dan peran layanan Spark

Pod driver Spark membutuhkan akun layanan Kubernetes di namespace yang sama dengan pod. Akun layanan ini membutuhkan izin untuk membuat, mendapatkan, membuat daftar, menambal, dan menghapus pod pelaksana, dan untuk membuat layanan tanpa kepala Kubernetes untuk driver. Driver gagal dan keluar tanpa akun layanan kecuali akun layanan default di namespace pod memiliki izin yang diperlukan.

Nama default untuk akun layanan ini adalah`emr-containers-sa-spark`.

Aturan berikut berlaku untuk peran layanan ini:

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

```
- apiGroups:
  - ""
  resources:
  - configmaps
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - persistentvolumeclaims
  verbs:
  - "*"
```

Menyiapkan izin akses klaster dengan IAM peran untuk akun layanan () IRSA

Bagian ini menggunakan contoh untuk menunjukkan cara mengonfigurasi akun layanan Kubernetes untuk mengambil peran. AWS Identity and Access Management Pod yang menggunakan akun layanan kemudian dapat mengakses AWS layanan apa pun yang perannya memiliki izin untuk diakses.

Contoh berikut menjalankan aplikasi Spark untuk menghitung kata-kata dari file di Amazon S3. Untuk melakukan ini, Anda dapat mengatur IAM peran untuk akun layanan (IRSA) untuk mengautentikasi dan mengotorisasi akun layanan Kubernetes.

> **Note**
>
> Contoh ini menggunakan namespace “spark-operator” untuk operator Spark dan untuk namespace tempat Anda mengirimkan aplikasi Spark.

Prasyarat

Sebelum Anda mencoba contoh di halaman ini, lengkapi prasyarat berikut:

- Siapkan untuk operator Spark.
- Instal operator Spark.
- Buat bucket Amazon S3.
- Simpan puisi favorit Anda dalam file teks bernama`poem.txt`, dan unggah file ke bucket S3 Anda. Aplikasi Spark yang Anda buat di halaman ini akan membaca isi file teks. Untuk informasi

selengkapnya tentang mengunggah file ke S3, lihat Mengunggah objek ke bucket di Panduan Pengguna Layanan Penyimpanan Sederhana Amazon.

Mengkonfigurasi akun layanan Kubernetes untuk mengambil peran IAM

Gunakan langkah-langkah berikut untuk mengonfigurasi akun layanan Kubernetes untuk mengambil IAM peran yang dapat digunakan Pod untuk mengakses AWS layanan yang memiliki izin untuk diakses oleh peran tersebut.

1. Setelah menyelesaikanPrasyarat, gunakan AWS Command Line Interface untuk membuat `example-policy.json` file yang memungkinkan akses hanya-baca ke file yang Anda unggah ke Amazon S3:

```
cat >example-policy.json <<EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::my-pod-bucket",
                "arn:aws:s3:::my-pod-bucket/*"
            ]
        }
    ]
}
EOF
```

2. Kemudian, buat IAM kebijakan`example-policy`:

```
aws iam create-policy --policy-name example-policy --policy-document file://
example-policy.json
```

3. Selanjutnya, buat IAM peran `example-role` dan kaitkan dengan akun layanan Kubernetes untuk driver Spark:

```
eksctl create iamserviceaccount --name driver-account-sa --namespace spark-operator
 \
--cluster my-cluster --role-name "example-role" \
--attach-policy-arn arn:aws:iam::111122223333:policy/example-policy --approve
```

4. Buat file yaml dengan binding peran cluster yang diperlukan untuk akun layanan driver Spark:

```
cat >spark-rbac.yaml <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  name: driver-account-sa
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: spark-role
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: edit
subjects:
  - kind: ServiceAccount
    name: driver-account-sa
    namespace: spark-operator
EOF
```

5. Menerapkan konfigurasi pengikatan peran cluster:

```
kubectl apply -f spark-rbac.yaml
```

Perintah kubectl harus mengonfirmasi keberhasilan pembuatan akun:

```
serviceaccount/driver-account-sa created
clusterrolebinding.rbac.authorization.k8s.io/spark-role configured
```

Menjalankan aplikasi dari operator Spark

Setelah Anda mengkonfigurasi akun layanan Kubernetes, Anda dapat menjalankan aplikasi Spark yang menghitung jumlah kata dalam file teks yang Anda unggah sebagai bagian dari. Prasyarat

1. Buat file baru`word-count.yaml`, dengan `SparkApplication` definisi untuk aplikasi penghitungan kata Anda.

```
cat >word-count.yaml <<EOF
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: word-count
  namespace: spark-operator
spec:
  type: Java
  mode: cluster
  image: "895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.10.0:latest"
  imagePullPolicy: Always
  mainClass: org.apache.spark.examples.JavaWordCount
  mainApplicationFile: local:///usr/lib/spark/examples/jars/spark-examples.jar
  arguments:
    - s3://my-pod-bucket/poem.txt
  hadoopConf:
   # EMRFS filesystem
    fs.s3.customAWSCredentialsProvider:
 com.amazonaws.auth.WebIdentityTokenCredentialsProvider
    fs.s3.impl: com.amazon.ws.emr.hadoop.fs.EmrFileSystem
    fs.AbstractFileSystem.s3.impl: org.apache.hadoop.fs.s3.EMRFSDelegate
    fs.s3.buffer.dir: /mnt/s3
    fs.s3.getObject.initialSocketTimeoutMilliseconds: "2000"

 mapreduce.fileoutputcommitter.algorithm.version.emr_internal_use_only.EmrFileSystem:
 "2"
    mapreduce.fileoutputcommitter.cleanup-
failures.ignored.emr_internal_use_only.EmrFileSystem: "true"
  sparkConf:
    # Required for EMR Runtime
    spark.driver.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/
hadoop-aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/
share/aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/
security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-
glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-
serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/
hadoop/extrajars/*
    spark.driver.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/
lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/native
```

```
    spark.executor.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/
hadoop-aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/
share/aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/
security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-
glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-
serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/
hadoop/extrajars/*
    spark.executor.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-
lzo/lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/
native
  sparkVersion: "3.3.1"
  restartPolicy:
    type: Never
  driver:
    cores: 1
    coreLimit: "1200m"
    memory: "512m"
    labels:
      version: 3.3.1
    serviceAccount: my-spark-driver-sa
  executor:
    cores: 1
    instances: 1
    memory: "512m"
    labels:
      version: 3.3.1
EOF
```

2. Kirim aplikasi Spark.

```
kubectl apply -f word-count.yaml
```

Perintah kubectl harus mengembalikan konfirmasi bahwa Anda berhasil membuat objek yang `SparkApplication` dipanggil. `word-count`

```
sparkapplication.sparkoperator.k8s.io/word-count configured
```

3. Untuk memeriksa peristiwa untuk `SparkApplication` objek, jalankan perintah berikut:

```
kubectl describe sparkapplication word-count -n spark-operator
```

Perintah kubectl harus mengembalikan deskripsi `SparkApplication` dengan peristiwa:

```
Events:
  Type     Reason                                   Age                    From
    Message
  ----     ------                                   ----                   ----
    -------
  Normal   SparkApplicationSpecUpdateProcessed  3m2s (x2 over 17h)     spark-
operator  Successfully processed spec update for SparkApplication word-count
  Warning  SparkApplicationPendingRerun         3m2s (x2 over 17h)     spark-
operator  SparkApplication word-count is pending rerun
  Normal   SparkApplicationSubmitted            2m58s (x2 over 17h)    spark-
operator  SparkApplication word-count was submitted successfully
  Normal   SparkDriverRunning                   2m56s (x2 over 17h)    spark-
operator  Driver word-count-driver is running
  Normal   SparkExecutorPending                 2m50s                  spark-
operator  Executor [javawordcount-fdd1698807392c66-exec-1] is pending
  Normal   SparkExecutorRunning                 2m48s                  spark-
operator  Executor [javawordcount-fdd1698807392c66-exec-1] is running
  Normal   SparkDriverCompleted                 2m31s (x2 over 17h)    spark-
operator  Driver word-count-driver completed
  Normal   SparkApplicationCompleted            2m31s (x2 over 17h)    spark-
operator  SparkApplication word-count completed
  Normal   SparkExecutorCompleted               2m31s (x2 over 2m31s)  spark-
operator  Executor [javawordcount-fdd1698807392c66-exec-1] completed
```

Aplikasi ini sekarang menghitung kata-kata dalam file S3 Anda. Untuk menemukan jumlah kata, lihat file log untuk driver Anda:

```
kubectl logs pod/word-count-driver -n spark-operator
```

Perintah kubectl harus mengembalikan isi file log dengan hasil aplikasi hitungan kata Anda.

```
INFO DAGScheduler: Job 0 finished: collect at JavaWordCount.java:53, took 5.146519 s
               Software: 1
```

Untuk informasi selengkapnya tentang cara mengirimkan aplikasi ke Spark melalui operator Spark, lihat dokumentasi Using a SparkApplication in the Kubernetes Operator for Apache Spark (8s-operator). spark-on-k GitHub

Menjalankan pekerjaan Spark dengan spark-submit

Amazon EMR merilis 6.10.0 dan dukungan yang lebih tinggi `spark-submit` sebagai alat baris perintah yang dapat Anda gunakan untuk mengirimkan dan menjalankan aplikasi Spark ke EMR Amazon di kluster EKS.

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

Topik

Menyiapkan spark-submit untuk Amazon EMR di EKS

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

- Instal AWS CLI— Jika Anda sudah menginstal AWS CLI, konfirmasikan bahwa Anda memiliki versi terbaru.
- Instal eksctl - eksctl adalah alat baris perintah yang Anda gunakan untuk berkomunikasi dengan Amazon EKS.
- Siapkan cluster Amazon EKS — Ikuti langkah-langkah untuk membuat cluster Kubernetes baru dengan node di Amazon EKS.
- Pilih URI gambar dasar EMR Amazon (rilis 6.10.0 atau lebih tinggi) — `spark-submit` perintah ini didukung dengan rilis Amazon EMR 6.10.0 dan yang lebih tinggi.

- Konfirmasikan bahwa akun layanan driver memiliki izin yang sesuai untuk membuat dan menonton pod pelaksana. Untuk informasi selengkapnya, lihat Persyaratan keamanan akun layanan driver Spark untuk pengiriman spark.
- Siapkan profil AWS kredensial lokal Anda.
- Dari konsol Amazon EKS, pilih kluster EKS Anda, lalu temukan titik akhir klaster EKS, yang terletak di bawah Ikhtisar, Detail, lalu titik akhir server API.

Memulai dengan spark-submit untuk Amazon EMR di EKS

Jalankan aplikasi Spark

Amazon EMR 6.10.0 dan yang lebih tinggi mendukung spark-submit untuk menjalankan aplikasi Spark di cluster Amazon EKS. Untuk menjalankan aplikasi Spark, ikuti langkah-langkah berikut:

1. Sebelum Anda dapat menjalankan aplikasi Spark dengan `spark-submit` perintah, selesaikan langkah-langkahnya. Menyiapkan spark-submit untuk Amazon EMR di EKS
2. Jalankan wadah dengan EMR Amazon pada gambar dasar EKS. Lihat Cara memilih URI gambar dasar untuk informasi selengkapnya.

   ```
   kubectl run -it containerName --image=EMROnEKSImage --command -n namespace /bin/
   bash
   ```

3. Tetapkan nilai untuk variabel lingkungan berikut:

   ```
   export SPARK_HOME=spark-home
   export MASTER_URL=k8s://Amazon EKS-cluster-endpoint
   ```

4. Sekarang, kirimkan aplikasi Spark dengan perintah berikut:

   ```
   $SPARK_HOME/bin/spark-submit \
    --class org.apache.spark.examples.SparkPi \
    --master $MASTER_URL \
    --conf spark.kubernetes.container.image=895885662937.dkr.ecr.us-
   west-2.amazonaws.com/spark/emr-6.10.0:latest \
    --conf spark.kubernetes.authenticate.driver.serviceAccountName=spark \
    --deploy-mode cluster \
    --conf spark.kubernetes.namespace=spark-operator \
    local:///usr/lib/spark/examples/jars/spark-examples.jar 20
   ```

Untuk informasi selengkapnya tentang mengirimkan aplikasi ke Spark, lihat Mengirimkan aplikasi dalam dokumentasi Apache Spark.

> **Important**
>
> `spark-submit`hanya mendukung mode cluster sebagai mekanisme pengiriman.

Persyaratan keamanan akun layanan driver Spark untuk pengiriman spark

Pod driver Spark menggunakan akun layanan Kubernetes untuk mengakses server API Kubernetes untuk membuat dan menonton pod pelaksana. Akun layanan driver harus memiliki izin yang sesuai untuk membuat daftar, membuat, mengedit, menambal, dan menghapus pod di klaster Anda. Anda dapat memverifikasi bahwa Anda dapat membuat daftar sumber daya ini dengan menjalankan perintah berikut:

```
kubectl auth can-i list|create|edit|delete|patch pods
```

Verifikasi bahwa Anda memiliki izin yang diperlukan dengan menjalankan setiap perintah.

```
kubectl auth can-i list pods
kubectl auth can-i create pods
kubectl auth can-i edit pods
kubectl auth can-i delete pods
kubectl auth can-i patch pods
```

Aturan berikut berlaku untuk peran layanan ini:

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

```
  - "*"
- apiGroups:
  - ""
  resources:
  - configmaps
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - persistentvolumeclaims
  verbs:
  - "*"
```

Menyiapkan peran IAM untuk akun layanan (IRSA) untuk spark-submit

Bagian berikut menjelaskan cara mengatur peran IAM untuk akun layanan (IRSA) untuk mengautentikasi dan mengotorisasi akun layanan Kubernetes sehingga Anda dapat menjalankan aplikasi Spark yang disimpan di Amazon S3.

Prasyarat

Sebelum mencoba salah satu contoh dalam dokumentasi ini, pastikan Anda telah menyelesaikan prasyarat berikut:

- Selesai menyiapkan spark-submit
- Membuat ember S3 dan mengunggah toples aplikasi percikan

Mengonfigurasi akun layanan Kubernetes untuk mengambil peran IAM

Langkah-langkah berikut mencakup cara mengkonfigurasi akun layanan Kubernetes untuk mengambil peran AWS Identity and Access Management (IAM). Setelah Anda mengonfigurasi pod untuk menggunakan akun layanan, mereka kemudian dapat mengakses apa pun AWS layanan yang memiliki izin untuk diakses oleh peran tersebut.

1. Buat file kebijakan untuk mengizinkan akses hanya-baca ke objek Amazon S3 yang Anda unggah:

   ```
   cat >my-policy.json <<EOF
   {
       "Version": "2012-10-17",
   ```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::<my-spark-jar-bucket>",
                "arn:aws:s3:::<my-spark-jar-bucket>/*"
            ]
        }
    ]
}
EOF
```

2. Buat kebijakan IAM.

```
aws iam create-policy --policy-name my-policy --policy-document file://my-
policy.json
```

3. Buat peran IAM dan kaitkan dengan akun layanan Kubernetes untuk driver Spark

```
eksctl create iamserviceaccount --name my-spark-driver-sa --namespace spark-
operator \
--cluster my-cluster --role-name "my-role" \
--attach-policy-arn arn:aws:iam::111122223333:policy/my-policy --approve
```

4. Buat file YAMAL dengan izin yang diperlukan untuk akun layanan driver Spark:

```
cat >spark-rbac.yaml <<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: emr-containers-role-spark
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - "*"
```

```
- apiGroups:
  - ""
  resources:
  - services
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - configmaps
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - persistentvolumeclaims
  verbs:
  - "*"
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: spark-role-binding
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: emr-containers-role-spark
subjects:
- kind: ServiceAccount
  name: emr-containers-sa-spark
  namespace: default
EOF
```

5. Menerapkan konfigurasi pengikatan peran cluster.

```
kubectl apply -f spark-rbac.yaml
```

6. `kubectl`Perintah harus mengembalikan konfirmasi akun yang dibuat.

```
serviceaccount/emr-containers-sa-spark created
clusterrolebinding.rbac.authorization.k8s.io/emr-containers-role-spark configured
```

Menjalankan aplikasi Spark

Amazon EMR 6.10.0 dan yang lebih tinggi mendukung spark-submit untuk menjalankan aplikasi Spark di cluster Amazon EKS. Untuk menjalankan aplikasi Spark, ikuti langkah-langkah berikut:

1. Pastikan Anda telah menyelesaikan langkah-langkah dalam Menyiapkan spark-submit untuk Amazon EMR di EKS.
2. Tetapkan nilai untuk variabel lingkungan berikut:

```
export SPARK_HOME=spark-home
export MASTER_URL=k8s://Amazon EKS-cluster-endpoint
```

3. Sekarang, kirimkan aplikasi Spark dengan perintah berikut:

```
$SPARK_HOME/bin/spark-submit \
 --class org.apache.spark.examples.SparkPi \
 --master $MASTER_URL \
 --conf spark.kubernetes.container.image=895885662937.dkr.ecr.us-
west-2.amazonaws.com/spark/emr-6.15.0:latest \
 --conf spark.kubernetes.authenticate.driver.serviceAccountName=emr-containers-sa-
spark \
 --deploy-mode cluster \
 --conf spark.kubernetes.namespace=default \
 --conf "spark.driver.extraClassPath=/usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/
hadoop-aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/
share/aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/
security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-
glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-
serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/
hadoop/extrajars/*" \
 --conf "spark.driver.extraLibraryPath=/usr/lib/hadoop/lib/native:/usr/lib/hadoop-
lzo/lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/
native" \
 --conf "spark.executor.extraClassPath=/usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/
hadoop-aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/
share/aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/
security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-
glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-
serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/
hadoop/extrajars/*" \
 --conf "spark.executor.extraLibraryPath=/usr/lib/hadoop/lib/native:/usr/lib/
hadoop-lzo/lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/
lib/native" \
```

```
 --conf
 spark.hadoop.fs.s3.customAWSCredentialsProvider=com.amazonaws.auth.WebIdentityTokenCredent
 \
 --conf spark.hadoop.fs.s3.impl=com.amazon.ws.emr.hadoop.fs.EmrFileSystem \
 --conf
 spark.hadoop.fs.AbstractFileSystem.s3.impl=org.apache.hadoop.fs.s3.EMRFSDelegate \
 --conf spark.hadoop.fs.s3.buffer.dir=/mnt/s3 \
 --conf spark.hadoop.fs.s3.getObject.initialSocketTimeoutMilliseconds="2000" \
 --conf
 spark.hadoop.mapreduce.fileoutputcommitter.algorithm.version.emr_internal_use_only.EmrFile
 \
 --conf spark.hadoop.mapreduce.fileoutputcommitter.cleanup-
failures.ignored.emr_internal_use_only.EmrFileSystem="true" \
 s3://my-pod-bucket/spark-examples.jar 20
```

4. Setelah driver percikan menyelesaikan pekerjaan Spark, Anda akan melihat baris log di akhir kiriman yang menunjukkan bahwa pekerjaan Spark telah selesai.

```
23/11/24 17:02:14 INFO LoggingPodStatusWatcherImpl: Application
 org.apache.spark.examples.SparkPi with submission ID default:org-apache-spark-
examples-sparkpi-4980808c03ff3115-driver finished
23/11/24 17:02:14 INFO ShutdownHookManager: Shutdown hook called
```

Pembersihan

Setelah selesai menjalankan aplikasi, Anda dapat melakukan pembersihan dengan perintah berikut.

```
kubectl delete -f spark-rbac.yaml
```

Menggunakan Apache Livy dengan Amazon di EMR EKS

Dengan Amazon EMR merilis 7.1.0 dan yang lebih tinggi, Anda dapat menggunakan Apache Livy untuk mengirimkan pekerjaan di Amazon. EMR EKS Menggunakan Apache Livy, Anda dapat mengatur REST endpoint Apache Livy Anda sendiri dan menggunakannya untuk menyebarkan dan mengelola aplikasi Spark di cluster Amazon Anda. EKS Setelah Anda menginstal Livy di EKS cluster Amazon Anda, Anda dapat menggunakan titik akhir Livy untuk mengirimkan aplikasi Spark ke server Livy Anda. Server mengelola siklus hidup aplikasi Spark.

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

Topik

Menyiapkan Apache Livy untuk Amazon di EMR EKS

Sebelum Anda dapat menginstal Apache Livy di EKS cluster Amazon Anda, Anda harus menyelesaikan tugas-tugas berikut.

- Instal AWS CLI— Jika Anda sudah menginstal AWS CLI, konfirmasikan bahwa Anda memiliki versi terbaru.
- Instal eksctl - eksctl adalah alat baris perintah yang Anda gunakan untuk berkomunikasi dengan Amazon. EKS
- Instal Helm — Manajer paket Helm untuk Kubernetes membantu Anda menginstal dan mengelola aplikasi di klaster Kubernetes Anda.
- Siapkan EKS klaster Amazon — Ikuti langkah-langkah untuk membuat klaster Kubernetes baru dengan node di Amazon. EKS
- Pilih label EMR rilis Amazon — Apache Livy didukung dengan EMR rilis Amazon 7.1.0 dan yang lebih tinggi.

- Instal ALB controller — ALB controller mengelola AWS Elastic Load Balancing untuk klaster Kubernetes. Ini menciptakan AWS Network Load Balancer (NLB) ketika Anda membuat Kubernetes Ingress saat menyiapkan Apache Livy.

Memulai dengan Apache Livy di Amazon di EMR EKS

Selesaikan langkah-langkah berikut untuk menginstal Apache Livy.

1. Jika Anda belum melakukannya, siapkan Apache Livy untuk Amazon EMR di. EKS
2. Otentikasi klien Helm Anda ke registri AmazonECR. Anda dapat menemukan `ECR-registry-account` nilai yang sesuai untuk akun Wilayah AWS ECR registri Amazon Anda berdasarkan Wilayah.

   ```
   aws ecr get-login-password \--region <AWS_REGION> | helm registry login \
   --username AWS \
   --password-stdin <ECR-registry-account>.dkr.ecr.<region-id>.amazonaws.com
   ```

3. Menyiapkan Livy membuat akun layanan untuk server Livy dan akun lain untuk aplikasi Spark. IRSAUntuk menyiapkan akun layanan, lihat Menyiapkan izin akses dengan IAM peran untuk akun layanan (IRSA).
4. Buat namespace untuk menjalankan beban kerja Spark Anda.

   ```
   kubectl create ns <spark-ns>
   ```

5. Gunakan perintah berikut untuk menginstal Livy.

   Titik akhir Livy ini hanya tersedia secara internal untuk VPC di cluster. EKS Untuk mengaktifkan akses di luarVPC, atur `—-set loadbalancer.internal=false` perintah instalasi Helm Anda.

   > **Note**
   >
   > Secara default, tidak SSL diaktifkan dalam titik akhir Livy ini dan titik akhir hanya terlihat di dalam clusterVPC. EKS Jika Anda mengatur `loadbalancer.internal=false` dan`ssl.enabled=false`, Anda mengekspos titik akhir yang tidak aman di luar Anda. VPC Untuk menyiapkan titik akhir Livy yang aman, lihat Mengonfigurasi titik akhir Apache Livy yang aman dengan/. TLS SSL

```
helm install livy-demo \
  oci://895885662937.dkr.ecr.region-id.amazonaws.com/livy \
  --version 7.2.0 \
  --namespace livy-ns \
  --set image=ECR-registry-account.dkr.ecr.region-id.amazonaws.com/livy/
emr-7.2.0:latest \
  --set sparkNamespace=<spark-ns> \
  --create-namespace
```

Anda akan melihat output berikut.

```
NAME: livy-demo
LAST DEPLOYED: Mon Mar 18 09:23:23 2024
NAMESPACE: livy-ns
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The Livy server has been installed.
Check installation status:
1. Check Livy Server pod is running
  kubectl --namespace livy-ns get pods -l "app.kubernetes.io/instance=livy-demo"
2. Verify created NLB is in Active state and it's target groups are healthy (if
 loadbalancer.enabled is true)

Access LIVY APIs:
    # Ensure your NLB is active and healthy
    # Get the Livy endpoint using command:
    LIVY_ENDPOINT=$(kubectl get svc -n livy-ns -l app.kubernetes.io/
instance=livy-demo,emr-containers.amazonaws.com/type=loadbalancer -o
 jsonpath='{.items[0].status.loadBalancer.ingress[0].hostname}' |  awk '{printf
 "%s:8998\n", $0}')
    # Access Livy APIs using http://$LIVY_ENDPOINT or https://$LIVY_ENDPOINT (if
 SSL is enabled)
    # Note: While uninstalling Livy, makes sure the ingress and NLB are deleted
 after running the helm command to avoid dangling resources
```

Nama akun layanan default untuk server Livy dan sesi Spark adalah `emr-containers-sa-livy` dan. `emr-containers-sa-spark-livy` Untuk menggunakan nama kustom, gunakan `sparkServiceAccount.name` parameter `serviceAccounts.name` dan.

```
--set serviceAccounts.name=my-service-account-for-livy
--set sparkServiceAccount.name=my-service-account-for-spark
```

6. Verifikasi bahwa Anda menginstal bagan Helm.

```
helm list -n livy-ns -o yaml
```

`helm list`Perintah harus mengembalikan informasi tentang bagan Helm baru Anda.

```
app_version: 0.7.1-incubating
chart: livy-emr-7.2.0
name: livy-demo
namespace: livy-ns
revision: "1"
status: deployed
updated: 2024-02-08 22:39:53.539243 -0800 PST
```

7. Pastikan Network Load Balancer aktif.

```
LIVY_NAMESPACE=<livy-ns>
LIVY_APP_NAME=<livy-app-name>
AWS_REGION=<AWS_REGION>

# Get the NLB Endpoint URL
NLB_ENDPOINT=$(kubectl --namespace $LIVY_NAMESPACE get svc -l "app.kubernetes.io/
instance=$LIVY_APP_NAME,emr-containers.amazonaws.com/type=loadbalancer" -o
 jsonpath='{.items[0].status.loadBalancer.ingress[0].hostname}')

# Get all the load balancers in the account's region
ELB_LIST=$(aws elbv2 describe-load-balancers --region $AWS_REGION)

# Get the status of the NLB that matching the endpoint from the Kubernetes service
NLB_STATUS=$(echo $ELB_LIST | grep -A 8 "\"DNSName\": \"$NLB_ENDPOINT\"" | awk '/
Code/{print $2}/}/' | tr -d '"},\n')
echo $NLB_STATUS
```

8. Sekarang verifikasi bahwa kelompok target di Network Load Balancer sehat.

```
LIVY_NAMESPACE=<livy-ns>
LIVY_APP_NAME=<livy-app-name>
AWS_REGION=<AWS_REGION>
```

```
# Get the NLB endpoint
NLB_ENDPOINT=$(kubectl --namespace $LIVY_NAMESPACE get svc -l "app.kubernetes.io/
instance=$LIVY_APP_NAME,emr-containers.amazonaws.com/type=loadbalancer" -o
 jsonpath='{.items[0].status.loadBalancer.ingress[0].hostname}')

# Get all the load balancers in the account's region
ELB_LIST=$(aws elbv2 describe-load-balancers --region $AWS_REGION)

# Get the NLB ARN from the NLB endpoint
NLB_ARN=$(echo $ELB_LIST | grep -B 1 "\"DNSName\": \"$NLB_ENDPOINT\"" | awk
 '/"LoadBalancerArn":/,/"/'| awk '/:/{print $2}' | tr -d \",)

# Get the target group from the NLB. Livy setup only deploys 1 target group
TARGET_GROUP_ARN=$(aws elbv2 describe-target-groups --load-balancer-arn $NLB_ARN
 --region $AWS_REGION | awk '/"TargetGroupArn":/,/"/'| awk '/:/{print $2}' | tr -d
 \",)

# Get health of target group
aws elbv2 describe-target-health --target-group-arn $TARGET_GROUP_ARN
```

Berikut ini adalah contoh output yang menunjukkan status kelompok target:

```
{
    "TargetHealthDescriptions": [
        {
            "Target": {
                "Id": "<target IP>",
                "Port": 8998,
                "AvailabilityZone": "us-west-2d"
            },
            "HealthCheckPort": "8998",
            "TargetHealth": {
                "State": "healthy"
            }
        }
    ]
}
```

Setelah status Anda NLB menjadi `active` dan kelompok target Anda`healthy`, Anda dapat melanjutkan. Mungkin butuh beberapa menit.

9. Ambil titik akhir Livy dari instalasi Helm. Apakah titik akhir Livy Anda aman atau tidak tergantung pada apakah Anda mengaktifkan. SSL

```
LIVY_NAMESPACE=<livy-ns>
 LIVY_APP_NAME=livy-app-name
 LIVY_ENDPOINT=$(kubectl get svc -n livy-ns -l app.kubernetes.io/
instance=livy-app-name,emr-containers.amazonaws.com/type=loadbalancer -o
 jsonpath='{.items[0].status.loadBalancer.ingress[0].hostname}' |  awk '{printf
 "%s:8998\n", $0}')
 echo "$LIVY_ENDPOINT"
```

10. Ambil akun layanan Spark dari instalasi Helm

```
SPARK_NAMESPACE=spark-ns
LIVY_APP_NAME=<livy-app-name>
SPARK_SERVICE_ACCOUNT=$(kubectl --namespace $SPARK_NAMESPACE
 get sa -l "app.kubernetes.io/instance=$LIVY_APP_NAME" -o
 jsonpath='{.items[0].metadata.name}')
echo "$SPARK_SERVICE_ACCOUNT"
```

    Anda akan melihat sesuatu yang mirip dengan output berikut:

```
emr-containers-sa-spark-livy
```

11. Jika Anda mengatur `internalALB=true` untuk mengaktifkan akses dari luarVPC, buat EC2 instans Amazon dan pastikan Network Load Balancer memungkinkan lalu lintas jaringan yang berasal dari instans. EC2 Anda harus melakukannya misalnya untuk memiliki akses ke titik akhir Livy Anda. Untuk informasi selengkapnya tentang mengekspos titik akhir Anda dengan aman di luarVPC, lihat Menyiapkan dengan titik akhir Apache Livy yang aman dengan/. TLS SSL
12. Menginstal Livy membuat akun layanan `emr-containers-sa-spark` untuk menjalankan aplikasi Spark. Jika aplikasi Spark Anda menggunakan AWS sumber daya apa pun seperti S3 atau panggilan AWS API atau CLI operasi, Anda harus menautkan IAM peran dengan izin yang diperlukan ke akun layanan spark Anda. Untuk informasi selengkapnya, lihat Menyiapkan izin akses dengan IAM peran untuk akun layanan (IRSA).

Apache Livy mendukung konfigurasi tambahan yang dapat Anda gunakan saat menginstal Livy. Untuk informasi selengkapnya, lihat Properti instalasi untuk Apache Livy EMR di Amazon pada EKS rilis.

Menjalankan aplikasi Spark dengan Apache Livy untuk Amazon EMR EKS

Sebelum Anda dapat menjalankan aplikasi Spark dengan Apache Livy, pastikan bahwa Anda telah menyelesaikan langkah-langkah dalam Menyiapkan Apache Livy untuk Amazon EKS dan Memulai dengan Apache Livy untuk Amazon EMR di. EMR EKS

Anda dapat menggunakan Apache Livy untuk menjalankan dua jenis aplikasi:

- Batch session — jenis beban kerja Livy untuk mengirimkan pekerjaan batch Spark.
- Sesi interaktif — jenis beban kerja Livy yang menyediakan antarmuka terprogram dan visual untuk menjalankan kueri Spark.

> **Note**
>
> Pod driver dan eksekutor dari sesi yang berbeda dapat berkomunikasi satu sama lain. Namespace tidak menjamin keamanan antar pod. Kubernetes tidak mengizinkan izin selektif pada subset pod di dalam namespace tertentu.

Menjalankan sesi batch

Untuk mengirimkan pekerjaan batch, gunakan perintah berikut.

```
curl -s -k -H 'Content-Type: application/json' -X POST \
      -d '{
            "name": "my-session",
            "file": "entryPoint_location (S3 or local)",
            "args": ["argument1", "argument2", ...],
            "conf": {
                "spark.kubernetes.namespace": "<spark-namespace>",
                "spark.kubernetes.container.image": "public.ecr.aws/emr-on-eks/spark/
emr-7.2.0:latest",
                "spark.kubernetes.authenticate.driver.serviceAccountName": "<spark-
service-account>"
            }
         }' <livy-endpoint>/batches
```

Untuk memantau pekerjaan batch Anda, gunakan perintah berikut.

```
curl -s -k -H 'Content-Type: application/json' -X GET <livy-endpoint>/batches/my-
session
```

Menjalankan sesi interaktif

Untuk menjalankan sesi interaktif dengan Apache Livy, lihat langkah-langkah berikut.

1. Pastikan Anda memiliki akses ke notebook Jupyter yang dihosting sendiri atau yang dikelola, seperti notebook Jupyter. SageMaker Notebook jupyter Anda harus memiliki sparkmagic yang diinstal.
2. Buat ember untuk konfigurasi `spark.kubernetes.file.upload.path` Spark. Pastikan akun layanan Spark telah membaca dan menulis akses ke bucket. Untuk detail selengkapnya tentang cara mengonfigurasi akun layanan spark, lihat Menyiapkan izin akses dengan IAM peran untuk akun layanan () IRSA
3. Muat sparkmagic di notebook Jupyter dengan perintah. `%load_ext sparkmagic.magics`
4. Jalankan perintah `%manage_spark` untuk mengatur titik akhir Livy Anda dengan notebook Jupyter. Pilih tab Tambah Titik Akhir, pilih jenis autentikasi yang dikonfigurasi, tambahkan titik akhir Livy ke buku catatan, lalu pilih Tambah titik akhir.
5. Jalankan `%manage_spark` lagi untuk membuat konteks Spark dan kemudian pergi ke sesi Create. Pilih titik akhir Livy, tentukan nama sesi unik pilih bahasa, lalu tambahkan properti berikut.

   ```
   {
     "conf": {
       "spark.kubernetes.namespace": "livy-namespace",
       "spark.kubernetes.container.image": "public.ecr.aws/emr-on-eks/spark/
   emr-7.2.0:latest",
       "spark.kubernetes.authenticate.driver.serviceAccountName": "<spark-service-
   account>",
       "spark.kubernetes.file.upload.path": "<URI_TO_S3_LOCATION_>"
     }
   }
   ```

6. Kirim aplikasi dan tunggu sampai membuat konteks Spark.
7. Untuk memantau status sesi interaktif, jalankan perintah berikut.

   ```
   curl -s -k -H 'Content-Type: application/json' -X GET livy-endpoint/sessions/my-
   interactive-session
   ```

Memantau aplikasi Spark

Untuk memantau kemajuan aplikasi Spark Anda dengan Livy UI, gunakan tautan. `http://<livy-endpoint>/ui`

Menghapus instalasi Apache Livy dengan Amazon pada EMR EKS

Ikuti langkah-langkah ini untuk menghapus instalasi Apache Livy.

1. Hapus pengaturan Livy menggunakan nama namespace dan nama aplikasi Anda. Dalam contoh ini, nama aplikasi adalah `livy-demo` dan namespace adalah. `livy-ns`

   ```
   helm uninstall livy-demo -n livy-ns
   ```

2. Saat mencopot pemasangan, EMR Amazon akan EKS menghapus layanan Kubernetes di Livy, AWS load balancer, dan grup target yang Anda buat selama instalasi. Menghapus sumber daya dapat memakan waktu beberapa menit. Pastikan sumber daya dihapus sebelum menginstal Livy di namespace lagi.
3. Hapus namespace Spark.

   ```
   kubectl delete namespace spark-ns
   ```

Keamanan untuk Apache Livy dengan Amazon di EMR EKS

Lihat halaman berikut untuk mempelajari lebih lanjut tentang mengonfigurasi keamanan Apache Livy dengan Amazon di EMR EKS

Topik

- Menyiapkan endpoint Apache Livy yang aman dengan/TLSSSL
- Menyiapkan izin aplikasi Apache Livy dan Spark dengan kontrol akses berbasis peran () RBAC
- Menyiapkan izin akses dengan IAM peran untuk akun layanan () IRSA

Menyiapkan endpoint Apache Livy yang aman dengan/TLSSSL

Lihat bagian berikut untuk mempelajari lebih lanjut tentang pengaturan Apache Livy untuk EMR Amazon EKS dengan end-to-end TLS dan SSL enkripsi.

Menyiapkan TLS dan SSL enkripsi

Untuk mengatur SSL enkripsi pada titik akhir Apache Livy Anda, ikuti langkah-langkah berikut.

- Instal Secrets Store CSI Driver dan AWS Secrets and Configuration Provider (ASCP) — ASCP Secrets Store CSI Driver dan simpan JKS sertifikat dan kata sandi Livy dengan aman yang perlu diaktifkan oleh pod server Livy. SSL Anda juga dapat menginstal hanya CSI Driver Secrets Store dan menggunakan penyedia rahasia lain yang didukung.
- Buat ACM sertifikat — sertifikat ini diperlukan untuk mengamankan koneksi antara klien dan ALB titik akhir.
- Siapkan JKS sertifikat, kata sandi kunci, dan kata sandi keystore untuk AWS Secrets Manager — diperlukan untuk mengamankan koneksi antara ALB titik akhir dan server Livy.
- Tambahkan izin ke akun layanan Livy untuk mengambil rahasia dari AWS Secrets Manager — server Livy memerlukan izin ini untuk mengambil rahasia dari ASCP dan menambahkan konfigurasi Livy untuk mengamankan server Livy. Untuk menambahkan IAM izin ke akun layanan, lihat Menyiapkan izin akses dengan IAM peran untuk akun layanan ()IRSA.

Menyiapkan JKS sertifikat dengan kunci dan kata sandi keystore untuk AWS Secrets Manager

Ikuti langkah-langkah ini untuk menyiapkan JKS sertifikat dengan kunci dan kata sandi keystore.

1. Hasilkan file keystore untuk server Livy.

```
keytool -genkey -alias <host> -keyalg RSA -keysize 2048 -dname
 CN=<host>,OU=hw,O=hw,L=<your_location>,ST=<state>,C=<country> -
keypass <keyPassword> -keystore <keystore_file> -storepass <storePassword> --
validity 3650
```

2. Buat sertifikat.

```
keytool -export -alias <host> -keystore mykeystore.jks -rfc -
file mycertificate.cert -storepass <storePassword>
```

3. Buat file truststore.

```
keytool -import -noprompt -alias <host>-file <cert_file> -
keystore <truststore_file> -storepass <truststorePassword>
```

4. Simpan JKS sertifikat di AWS Secrets Manager. Ganti `livy-jks-secret` dengan rahasia Anda dan `fileb://mykeystore.jks` dengan jalur ke JKS sertifikat keystore Anda.

```
aws secretsmanager create-secret \
--name livy-jks-secret \
--description "My Livy keystore JKS secret" \
--secret-binary fileb://mykeystore.jks
```

5. Simpan keystore dan kata sandi kunci di Secrets Manager. Pastikan untuk menggunakan parameter Anda sendiri.

```
aws secretsmanager create-secret \
--name livy-jks-secret \
--description "My Livy key and keystore password secret" \
--secret-string "{\"keyPassword\":\"<test-key-password>\",\"keyStorePassword\":
\"<test-key-store-password>\"}"
```

6. Buat namespace server Livy dengan perintah berikut.

```
kubectl create ns <livy-ns>
```

7. Buat `ServiceProviderClass` objek untuk server Livy yang memiliki JKS sertifikat dan kata sandi.

```
cat >livy-secret-provider-class.yaml << EOF
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: aws-secrets
spec:
  provider: aws
  parameters:
    objects: |
        - objectName: "livy-jks-secret"
          objectType: "secretsmanager"
        - objectName: "livy-passwords"
          objectType: "secretsmanager"

EOF
kubectl apply -f livy-secret-provider-class.yaml -n <livy-ns>
```

Memulai dengan SSL Apache Livy yang diaktifkan

Setelah mengaktifkan SSL di server Livy Anda, Anda harus mengatur `serviceAccount` untuk memiliki akses ke `keyStore` dan `keyPasswords` rahasia. AWS Secrets Manager

1. Buat namespace server Livy.

```
kubectl create namespace <livy-ns>
```

2. Siapkan akun layanan Livy untuk memiliki akses ke rahasia di Secrets Manager. Untuk informasi selengkapnya tentang pengaturanIRSA, lihat Menyiapkan IRSA saat menginstal Apache Livy.

```
aws ecr get-login-password \--region region-id | helm registry login \
--username AWS \
--password-stdin ECR-registry-account.dkr.ecr.region-id.amazonaws.com
```

3. Instal Livy. Untuk parameter Helm chart --version, gunakan label EMR rilis Amazon Anda, seperti. `7.1.0` Anda juga harus mengganti ID akun ECR registri Amazon dan ID Wilayah dengan ID Anda sendiriIDs. Anda dapat menemukan `ECR-registry-account` nilai yang sesuai untuk akun Wilayah AWS ECR registri Amazon Anda berdasarkan Wilayah.

```
helm install <livy-app-name> \
  oci://895885662937.dkr.ecr.region-id.amazonaws.com/livy \
  --version 7.2.0 \
  --namespace livy-namespace-name \
  --set image=<ECR-registry-account.dkr.ecr>.<region>.amazonaws.com/livy/
emr-7.2.0:latest \
  --set sparkNamespace=spark-namespace \
  --set ssl.enabled=true
  --set ssl.CertificateArn=livy-acm-certificate-arn
  --set ssl.secretProviderClassName=aws-secrets
  --set ssl.keyStoreObjectName=livy-jks-secret
  --set ssl.keyPasswordsObjectName=livy-passwords
  --create-namespace
```

4. Lanjutkan dari langkah 5 dari Menginstal Apache Livy di Amazon EMR di. EKS

Menyiapkan izin aplikasi Apache Livy dan Spark dengan kontrol akses berbasis peran () RBAC

Untuk menyebarkan Livy, EMR Amazon EKS membuat akun dan peran layanan server serta akun dan peran layanan Spark. Peran ini harus memiliki RBAC izin yang diperlukan untuk menyelesaikan pengaturan dan menjalankan aplikasi Spark.

RBACizin untuk akun dan peran layanan server

Amazon EMR on EKS membuat akun layanan server Livy dan berperan untuk mengelola sesi Livy untuk pekerjaan Spark dan merutekan lalu lintas ke dan dari ingress dan sumber daya lainnya.

Nama default untuk akun layanan ini adalah`emr-containers-sa-livy`. Itu harus memiliki izin berikut.

```
rules:
- apiGroups:
  - ""
  resources:
  - "namespaces"
  verbs:
  - "get"
- apiGroups:
  - ""
  resources:
  - "serviceaccounts"
    "services"
    "configmaps"
    "events"
    "pods"
    "pods/log"
  verbs:
  - "get"
    "list"
    "watch"
    "describe"
    "create"
    "edit"
    "delete"
    "deletecollection"
    "annotate"
    "patch"
```

```
    "label"
 - apiGroups:
   - ""
   resources:
   - "secrets"
   verbs:
   - "create"
     "patch"
     "delete"
     "watch"
 - apiGroups:
   - ""
   resources:
   - "persistentvolumeclaims"
   verbs:
   - "get"
     "list"
     "watch"
     "describe"
     "create"
     "edit"
     "delete"
     "annotate"
     "patch"
     "label"
```

RBACizin untuk akun dan peran layanan spark

Pod driver Spark membutuhkan akun layanan Kubernetes di namespace yang sama dengan pod. Akun layanan ini memerlukan izin untuk mengelola pod pelaksana dan sumber daya apa pun yang diperlukan oleh pod driver. Kecuali akun layanan default di namespace memiliki izin yang diperlukan, driver gagal dan keluar. RBACIzin berikut diperlukan.

```
rules:
- apiGroups:
  - ""
    "batch"
    "extensions"
    "apps"
  resources:
  - "configmaps"
    "serviceaccounts"
    "events"
```

```
    "pods"
    "pods/exec"
    "pods/log"
    "pods/portforward"
    "secrets"
    "services"
    "persistentvolumeclaims"
    "statefulsets"
  verbs:
  - "create"
    "delete"
    "get"
    "list"
    "patch"
    "update"
    "watch"
    "describe"
    "edit"
    "deletecollection"
    "patch"
    "label"
```

Menyiapkan izin akses dengan IAM peran untuk akun layanan () IRSA

Secara default, server Livy dan driver dan pelaksana aplikasi Spark tidak memiliki akses ke sumber daya. AWS Akun layanan server dan akun layanan spark mengontrol akses ke AWS sumber daya untuk server Livy dan pod aplikasi spark. Untuk memberikan akses, Anda perlu memetakan akun layanan dengan IAM peran yang memiliki AWS izin yang diperlukan.

Anda dapat mengatur IRSA pemetaan sebelum menginstal Apache Livy, selama instalasi, atau setelah Anda menyelesaikan instalasi.

Menyiapkan IRSA saat menginstal Apache Livy (untuk akun layanan server)

> **Note**
>
> Pemetaan ini hanya didukung untuk akun layanan server.

1. Pastikan Anda telah selesai menyiapkan Apache Livy untuk Amazon EKS dan EMR sedang menginstal Apache Livy dengan Amazon aktif. EMR EKS

2. Buat namespace Kubernetes untuk server Livy. Dalam contoh ini, nama namespace adalah. `livy-ns`
3. Buat IAM kebijakan yang menyertakan izin yang AWS layanan ingin diakses oleh pod Anda. Contoh berikut membuat IAM kebijakan untuk mendapatkan sumber daya Amazon S3 untuk titik masuk Spark.

```
cat >my-policy.json <<EOF{
"Version": "2012-10-17",
    "Statement": [
        {
"Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::my-spark-entrypoint-bucket"
        }
    ]
}
EOF

aws iam create-policy --policy-name my-policy --policy-document file://my-
policy.json
```

4. Gunakan perintah berikut untuk mengatur Akun AWS ID Anda ke variabel.

```
account_id=$(aws sts get-caller-identity --query "Account" --output text)
```

5. Setel penyedia identitas OpenID Connect (OIDC) klaster Anda ke variabel lingkungan.

```
oidc_provider=$(aws eks describe-cluster --name my-cluster --region $AWS_REGION --
query "cluster.identity.oidc.issuer" --output text | sed -e "s/^https:\/\///")
```

6. Tetapkan variabel untuk namespace dan nama akun layanan. Pastikan untuk menggunakan nilai Anda sendiri.

```
export namespace=default
export service_account=my-service-account
```

7. Buat file kebijakan kepercayaan dengan perintah berikut. Jika Anda ingin memberikan akses peran ke semua akun layanan dalam namespace, salin perintah berikut, dan ganti dengan dan ganti `StringEquals` dengan`StringLike`. `$service_account *`

```
cat >trust-relationship.json <<EOF
```

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::$account_id:oidc-provider/$oidc_provider"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "$oidc_provider:aud": "sts.amazonaws.com",
          "$oidc_provider:sub": "system:serviceaccount:$namespace:$service_account"
        }
      }
    }
  ]
}
EOF
```

8. Buat peran.

```
aws iam create-role --role-name my-role --assume-role-policy-document file://trust-
relationship.json --description "my-role-description"
```

9. Gunakan perintah Helm install berikut untuk mengatur `serviceAccount.executionRoleArn` to mapIRSA. Berikut ini adalah contoh perintah Helm install. Anda dapat menemukan `ECR-registry-account` nilai yang sesuai untuk akun Wilayah AWS ECR registri Amazon Anda berdasarkan Wilayah.

```
helm install livy-demo \
  oci://895885662937.dkr.ecr.us-west-2.amazonaws.com/livy \
  --version 7.2.0 \
  --namespace livy-ns \
  --set image=ECR-registry-account.dkr.ecr.region-id.amazonaws.com/livy/
emr-7.2.0:latest \
  --set sparkNamespace=spark-ns \
  --set serviceAccount.executionRoleArn=arn:aws:iam::123456789012:role/my-role
```

Pemetaan IRSA ke akun layanan Spark

Sebelum Anda memetakan IRSA ke akun layanan Spark, pastikan Anda telah menyelesaikan item berikut:

- Pastikan Anda telah selesai menyiapkan Apache Livy untuk Amazon EKS dan EMR sedang menginstal Apache Livy dengan Amazon aktif. EMR EKS
- Anda harus memiliki provdider IAM OpenID Connect (OIDC) yang ada untuk klaster Anda. Untuk melihat apakah Anda sudah memilikinya atau cara membuatnya, lihat Membuat IAM OIDC penyedia untuk klaster Anda.
- Pastikan Anda telah menginstal versi 0.171.0 atau yang lebih baru dari yang `eksctl` CLI diinstal atau. AWS CloudShell Untuk menginstal atau memperbarui`eksctl`, lihat Instalasi `eksctl` dokumentasi.

Ikuti langkah-langkah berikut untuk memetakan IRSA ke akun layanan Spark Anda:

1. Gunakan perintah berikut untuk mendapatkan akun layanan Spark.

```
SPARK_NAMESPACE=<spark-ns>
LIVY_APP_NAME=<livy-app-name>
kubectl --namespace $SPARK_NAMESPACE describe sa -l "app.kubernetes.io/instance=
$LIVY_APP_NAME" | awk '/^Name:/ {print $2}'
```

2. Tetapkan variabel Anda untuk namespace dan nama akun layanan.

```
export namespace=default
export service_account=my-service-account
```

3. Gunakan perintah berikut untuk membuat file kebijakan kepercayaan untuk IAM peran tersebut. Contoh berikut memberikan izin ke semua akun layanan dalam namespace untuk menggunakan peran. Untuk melakukannya, ganti `StringEquals` dengan `StringLike` dan ganti `$service_account` dengan*.

```
cat >trust-relationship.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
```

```
        "Federated": "arn:aws:iam::$account_id:oidc-provider/$oidc_provider"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "$oidc_provider:aud": "sts.amazonaws.com",
          "$oidc_provider:sub": "system:serviceaccount:$namespace:$service_account"
        }
      }
    }
  ]
}
EOF
```

4. Buat peran.

```
aws iam create-role --role-name my-role --assume-role-policy-document file://trust-relationship.json --description "my-role-description"
```

5. Petakan server atau akun layanan spark dengan `eksctl` perintah berikut. Pastikan untuk menggunakan nilai Anda sendiri.

```
eksctl create iamserviceaccount --name spark-sa \
--namespace spark-namespace --cluster livy-eks-cluster \
--attach-role-arn arn:aws:iam::0123456789012:role/my-role \
--approve --override-existing-serviceaccounts
```

Properti instalasi untuk Apache Livy di Amazon EMR pada rilis EKS

Instalasi Apache Livy memungkinkan Anda memilih versi bagan Livy Helm. Bagan Helm menawarkan berbagai properti untuk menyesuaikan pengalaman instalasi dan penyiapan Anda. Properti ini didukung untuk Amazon EMR pada EKS rilis 7.1.0 dan yang lebih tinggi.

Topik

- Properti instalasi Amazon EMR 7.1.0

Properti instalasi Amazon EMR 7.1.0

Tabel berikut menjelaskan semua properti Livy yang didukung. Saat menginstal Apache Livy, Anda dapat memilih versi bagan Livy Helm. Untuk mengatur properti selama instalasi, gunakan perintah`--set <property>=<value>`.

Properti	Deskripsi	Default
gambar	EMRRilis Amazon URI dari server Livy. Ini adalah konfigurasi yang diperlukan.	""
sparkNamespace	Namespace untuk menjalankan sesi Livy Spark. Misalnya, tentukan “livy”. Ini adalah konfigurasi yang diperlukan.	""
nameOverride	Berikan nama, bukan`livy`. Nama ditetapkan sebagai label untuk semua sumber daya Livy	“hidup”
fullnameOverride	Berikan nama untuk digunakan alih-alih nama lengkap sumber daya.	""
ssl.enabled	Aktifkan end-to-end SSL dari titik akhir Livy ke server Livy.	FALSE
ssl. certificateArn	Jika SSL diaktifkan, ini adalah ACM sertifikat ARN untuk yang NLB dibuat oleh layanan..	""
ssl. secretProviderClassNama	Jika SSL diaktifkan, ini adalah nama kelas penyedia rahasia NLB untuk mengamankan koneksi server Livy. SSL	""

Properti	Deskripsi	Default
ssl. keyStoreObjectNama	Jika SSL diaktifkan, nama objek untuk sertifikat keystore di kelas penyedia rahasia.	""
ssl. keyPasswordsObject Nama	Jika SSL diaktifkan, nama objek untuk rahasia yang memiliki keystore dan kata sandi kunci.	""
rbac.create	Jika benar, menciptakan RBAC sumber daya.	FALSE
serviceAccount.buat	Jika benar, buat akun layanan Livy.	TRUE
serviceAccount.nama	Nama akun layanan yang akan digunakan untuk Livy. Jika Anda tidak menyetel properti ini dan membuat akun layanan, EMR Amazon EKS secara otomatis membuat nama menggunakan properti `fullname` override.	"emr-containers-sa-livy"
serviceAccount.executionRol eArn	Peran ARN eksekusi akun layanan Livy.	""
sparkServiceAccount.buat	JIKA benar, buat akun layanan Spark di `.Release. Namespace`	TRUE

Properti	Deskripsi	Default
sparkServiceAccount.nama	Nama akun layanan yang akan digunakan untuk Spark. Jika Anda tidak menyetel properti ini dan membuat akun layanan Spark, EMR Amazon EKS secara otomatis menghasilkan nama dengan `fullnameOverride` properti dengan `-spark-livy` akhiran.	“emr-containers-sa-spark-hidup”
service.name	Nama layanan Livy	`"emr-containers-livy"`
service.annotations	Anotasi layanan Livy	`{}`
loadbalancer.enabled	Apakah akan membuat penyeimbang beban untuk layanan Livy yang digunakan untuk mengekspos titik akhir Livy di luar klaster Amazon. EKS	`FALSE`
loadbalancer.internal	Apakah akan mengonfigurasi titik akhir Livy sebagai internal ke VPC atau eksternal. Menyetel properti ini untuk `FALSE` mengekspos titik akhir ke sumber di luar. VPC Kami merekomendasikan mengamankan titik akhir Anda denganTLS/. SSL Untuk informasi selengkapnya, lihat Menyiapkan TLS dan SSL enkripsi.	`FALSE`

Properti	Deskripsi	Default
imagePullSecrets	Daftar `imagePullSecret` nama yang akan digunakan untuk menarik gambar Livy dari repositori pribadi.	[]
sumber daya	Permintaan sumber daya dan batasan untuk kontainer Livy.	{}
nodeSelector	Node untuk menjadwalkan pod Livy.	{}
toleransi	Daftar yang berisi toleransi pod Livy untuk didefinisikan.	[]
afinitas	Aturan afinitas pod Livy.	{}
persistence.enabled	Jika benar, aktifkan persistansi untuk direktori sesions.	FALSE
kegigihan. subPath	PVCSubpath untuk dipasang ke direktori sesi.	""
kegigihan. existingClaim	PVCUntuk menggunakan alih-alih membuat yang baru.	{}

Properti	Deskripsi	Default
kegigihan. storageClass	Kelas penyimpanan yang akan digunakan. Untuk menentukan parameter ini, gunakan format`storageClassName: <storageClass>` . Menyetel parameter ini untuk "-" menonaktifkan penyediaan dinamis. Jika Anda menyetel parameter ini ke null atau tidak menentukan apa pun, Amazon EMR on EKS tidak menyetel storageClassName dan menggunakan penyedia default.	""
kegigihan. accessMode	Mode PVC akses.	`ReadWriteOnce`
ketekunan.ukuran	PVCUkurannya.	20Gi
persistence.annotations	Anotasi tambahan untuk. PVC	{}
env. *	Env tambahan untuk disetel ke wadah Livy. Untuk informasi selengkapnya, lihat Memasukkan konfigurasi Livy dan Spark Anda sendiri saat menginstal Livy.	{}
envFrom.*	Env tambahan untuk disetel ke Livy dari peta konfigurasi Kubernetes atau rahasia.	[]
livyConf.*	Entri livy.conf tambahan untuk disetel dari peta konfigurasi Kubernetes atau rahasia yang dipasang.	{}

Properti	Deskripsi	Default
sparkDefaultsConf.*	`spark-defaults.conf` Entri tambahan untuk disetel dari peta konfigurasi Kubernetes atau rahasia yang dipasang.	{}

Pemecahan Masalah

Memasukkan konfigurasi Livy dan Spark Anda sendiri saat menginstal Livy

Anda dapat mengonfigurasi variabel lingkungan Apache Livy atau Apache Spark dengan properti Helm. `env.*` Ikuti langkah-langkah di bawah ini untuk mengonversi konfigurasi contoh `example.config.with-dash.withUppercase` ke format variabel lingkungan yang didukung.

1. Ganti huruf besar dengan huruf 1 dan huruf kecil. Misalnya, `example.config.with-dash.withUppercase` menjadi `example.config.with-dash.with1uppercase`.
2. Ganti tanda hubung (-) dengan 0. Misalnya, `example.config.with-dash.with1uppercase` menjadi `example.config.with0dash.with1uppercase`
3. Ganti titik (.) dengan garis bawah (_). Misalnya, `example.config.with0dash.with1uppercase` menjadi `example_config_with0dash_with1uppercase`.
4. Ganti semua huruf kecil dengan huruf besar.
5. Tambahkan awalan `LIVY_` ke nama variabel.
6. Gunakan variabel saat menginstal Livy melalui bagan helm menggunakan format --set env.*YOUR_VARIABLE_NAME*.nilai =*yourvalue*

Misalnya, untuk mengatur konfigurasi Livy dan Spark `livy.server.recovery.state-store = filesystem` dan`spark.kubernetes.executor.podNamePrefix = my-prefix`, gunakan properti Helm ini:

```
–set env.LIVY_LIVY_SERVER_RECOVERY_STATE0STORE.value=filesystem
–set env.LIVY_SPARK_KUBERNETES_EXECUTOR_POD0NAME0PREFIX.value=myprefix
```

Mengelola Amazon EMR pada pekerjaan EKS

Bagian berikut mencakup topik yang membantu Anda mengelola Amazon EMR Anda di EKS pekerjaan berjalan.

Topik

Mengelola pekerjaan berjalan denganAWS CLI

Halaman ini mencakup cara mengelola pekerjaan berjalan denganAWS Command Line Interface(AWS CLI).

Pilihan untuk mengonfigurasi tugas berjalan

Gunakan opsi berikut untuk mengonfigurasi parameter tugas berjalan:

- `--execution-role-arn`: Anda harus menyediakan IAM role yang digunakan untuk menjalankan tugas. Untuk informasi selengkapnya, lihat Menggunakan peran eksekusi tugas dengan Amazon EMR di EKS.
- `--release-label`: Anda dapat menyebarkan Amazon EMR di EKS dengan Amazon EMR versi 5.32.0 dan 6.2.0 dan lebih baru. Amazon EMR di EKS tidak didukung dalam versi rilis Amazon EMR sebelumnya. Untuk informasi selengkapnya, lihat Amazon EMR pada EKS rilis.
- `--job-driver`: Driver tugas digunakan untuk memberikan input pada tugas utama. Ini adalah bidang jenis serikat di mana Anda hanya dapat meloloskan salah satu nilai untuk jenis tugas yang ingin Anda jalankan. Jenis tugas yang didukung meliputi:
  - Tugas Spark submit - Digunakan untuk menjalankan perintah melalui Spark submit. Anda dapat menggunakan jenis pekerjaan ini untuk menjalankan Scala,PySpark, SparKr, SparkSQL dan pekerjaan lain yang didukung melalui Spark Submit. Tugas ini memiliki parameter berikut:
    - Entrypoint - Ini adalah referensi HCFS (Hadoop sistem file yang kompatibel) ke file jar/py file utama yang ingin Anda jalankan.

- EntryPointArguments- Ini adalah array argumen yang ingin Anda sampaikan ke file jar/py utama Anda. Anda harus menangani membaca parameter ini menggunakan kode entrypoint Anda. Setiap argumen dalam array harus dipisahkan dengan koma. EntryPointArgumentstidak dapat berisi tanda kurung atau tanda kurung, seperti (), {}, atau [].
- SparkSubmitParameters- Ini adalah parameter percikan tambahan yang ingin Anda kirim ke pekerjaan. Gunakan parameter ini untuk menimpa properti default Spark seperti memori driver atau jumlah pelaksana seperti —conf atau —class. Untuk informasi tambahan, lihat Peluncuran Aplikasi dengan spark-submit.

- Spark SQL jobs - Digunakan untuk menjalankan file query SQL melalui Spark SQL. Anda dapat menggunakan jenis pekerjaan ini untuk menjalankan pekerjaan SparkSQL. Tugas ini memiliki parameter berikut:
  - Entrypoint - ini adalah HCFS (Hadoop sistem file yang kompatibel) referensi ke file query SQL Anda ingin menjalankan.

    Untuk daftar parameter Spark tambahan yang dapat Anda gunakan untuk pekerjaan Spark SQL, lihatMenjalankan skrip Spark SQL melaluiStartJobRunAPI.

- `--configuration-overrides`: Anda dapat menimpa konfigurasi default untuk aplikasi dengan menyediakan objek konfigurasi. Anda dapat menggunakan sintaks singkatan untuk menyediakan konfigurasi atau Anda dapat mereferensikan objek konfigurasi dalam file JSON. Objek konfigurasi terdiri dari klasifikasi, properti, dan konfigurasi bersarang opsional. Properti terdiri dari pengaturan yang ingin Anda timpa dalam file tersebut. Anda dapat menentukan beberapa klasifikasi untuk beberapa aplikasi dalam objek JSON tunggal. Klasifikasi konfigurasi yang tersedia bervariasi berdasarkan versi rilis Amazon EMR. Untuk daftar klasifikasi konfigurasi yang tersedia untuk setiap versi rilis Amazon EMR, lihat Amazon EMR pada EKS rilis.

  Jika Anda melewati konfigurasi yang sama dalam penimpaan aplikasi dan di parameter kirim Spark, parameter kirim Spark diutamakan. Daftar prioritas konfigurasi lengkap mengikuti, dalam urutan prioritas tertinggi ke prioritas terendah.

  - Konfigurasi disediakan saat membuat `SparkSession`.
  - Konfigurasi disediakan sebagai bagian dari `sparkSubmitParameters` menggunakan `–conf`.
  - Konfigurasi disediakan sebagai bagian dari penimpaan aplikasi.
  - Konfigurasi yang dioptimalkan dipilih oleh Amazon EMR untuk rilisan.
  - Konfigurasi sumber terbuka default untuk aplikasi.

Untuk memantau pekerjaan yang berjalan menggunakan AmazonCloudWatchatau Amazon S3, Anda harus memberikan detail konfigurasi untukCloudWatch. Untuk informasi selengkapnya, lihat Mengonfigurasi pekerjaan yang dijalankan untuk menggunakan log Amazon S3 dan Mengonfigurasi pekerjaan yang dijalankan untuk menggunakan AmazonCloudWatchLog. Jika ember S3 atauCloudWatchgrup log tidak ada, lalu Amazon EMR membuatnya sebelum mengunggah log ke bucket.

- Untuk daftar tambahan opsi konfigurasi Kubernetes, lihat Properti Spark di Kubernetes.

  Konfigurasi Spark berikut tidak didukung.

  - `spark.kubernetes.authenticate.driver.serviceAccountName`
  - `spark.kubernetes.authenticate.executor.serviceAccountName`
  - `spark.kubernetes.namespace`
  - `spark.kubernetes.driver.pod.name`
  - `spark.kubernetes.container.image.pullPolicy`
  - `spark.kubernetes.container.image`

  > **Note**
  >
  > Anda dapat menggunakan`spark.kubernetes.container.image`untuk gambar Docker yang disesuaikan. Untuk informasi selengkapnya, lihat Menyesuaikan gambar Docker untuk Amazon di EMR EKS.

Mengonfigurasi pekerjaan yang dijalankan untuk menggunakan log Amazon S3

Untuk dapat memantau kemajuan pekerjaan dan memecahkan masalah kegagalan, Anda harus mengkonfigurasi pekerjaan Anda untuk mengirim informasi log ke Amazon S3, AmazonCloudWatchLog, atau keduanya. Topik ini membantu Anda memulai penerbitan log aplikasi ke Amazon S3 pada pekerjaan Anda yang diluncurkan dengan Amazon EMR di EKS.

S3 log kebijakan IAM

Sebelum tugas Anda dapat mengirim data log ke Amazon S3, izin berikut harus disertakan dalam kebijakan perizinan untuk peran eksekusi tugas. Ganti*DOC-CONTOH-EMBER-LOGGING*dengan nama bucket logging Anda.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET-LOGGING",
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET-LOGGING/*",
            ]
        }
    ]
}
```

> **Note**
>
> Amazon EMR di EKS juga dapat membuat bucket Amazon S3. Jika bucket Amazon S3 tidak tersedia, sertakan`"s3:CreateBucket"`izin dalam kebijakan IAM.

Setelah Anda memberikan peran eksekusi izin yang tepat untuk mengirim log ke Amazon S3, data log Anda akan dikirim ke lokasi Amazon S3 berikut saats3MonitoringConfigurationdilewatkan dimonitoringConfigurationbagian daristart-job-runpermintaan, seperti yang ditunjukkan padaMengelola pekerjaan berjalan denganAWS CLI.

- Log Pengontrol -/*LogURI*/*virtual-cluster-id*/pekerjaan/*id pekerjaan*/wadah/*nama pod*/ (stderr.gz/stdout.gz)
- Log Pengemudi -/*LogURI*/*virtual-cluster-id*/pekerjaan/*id pekerjaan*/wadah/*spark-application-id*/percikan api -*id pekerjaan*-driver/ (stderr.gz/stdout.gz)
- Log Pelaksana -/*LogURI*/*virtual-cluster-id*/pekerjaan/*id pekerjaan*/wadah/*spark-application-id*/*executor-pod-name*/(stderr.gz/stdout.gz)

Mengonfigurasi pekerjaan yang dijalankan untuk menggunakan AmazonCloudWatchLog

Untuk memantau kemajuan pekerjaan dan memecahkan masalah kegagalan, Anda harus mengonfigurasi pekerjaan Anda untuk mengirim informasi log ke Amazon S3, AmazonCloudWatchLog, atau keduanya. Topik ini membantu Anda mulai menggunakanCloudWatchLog pada pekerjaan Anda yang diluncurkan dengan Amazon EMR di EKS. Untuk informasi lebih lanjut tentangCloudWatchLog, lihatMemantau File Logdi AmazonCloudWatchPanduan Pengguna.

CloudWatchLog kebijakan IAM

Agar pekerjaan Anda dapat mengirim data log keCloudWatchLog, izin berikut harus disertakan dalam kebijakan izin untuk peran eksekusi pekerjaan. Ganti*my_log_group_name*dan*my_log_stream_prefiks*dengan nama-namaCloudWatchlog kelompok dan log nama aliran, masing-masing. Amazon EMR di EKS menciptakan grup log dan stream log jika mereka tidak ada selama peran eksekusi ARN memiliki izin yang sesuai.

```
{
     "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:*:*:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:*:*:log-group:my_log_group_name:log-
stream:my_log_stream_prefix/*"
            ]
        }
```

```
    ]
}
```

> **Note**
>
> Amazon EMR di EKS juga dapat membuat aliran log. Jika aliran log tidak ada, kebijakan IAM harus menyertakan`"logs:CreateLogGroup"`izin.

Setelah Anda memberikan peran eksekusi izin yang tepat, aplikasi Anda akan mengirimkan data lognya keCloudWatchLog saat`cloudWatchMonitoringConfiguration`dilewatkan di`monitoringConfiguration`bagian dari`start-job-run`permintaan, seperti yang ditunjukkan padaMengelola pekerjaan berjalan denganAWS CLI.

Dalam`StartJobRun`API,*`log_group_name`*adalah nama grup log untukCloudWatch, dan*`log_stream_prefiks`*adalah awalan nama aliran log untukCloudWatch. Anda dapat melihat dan mencari log ini di AWS Management Console.

- Log pengontrol -*`LogGroup`*/*`logStreamPrefix`*/*`virtual-cluster-id`*/pekerjaan/*`id pekerjaan`*/wadah/*`nama pod`*/(stderr/stdout)
- Log driver -*`LogGroup`*/*`logStreamPrefix`*/*`virtual-cluster-id`*/pekerjaan/*`id pekerjaan`*/wadah/*`spark-application-id`*/percikan api -*`id pekerjaan`*-driver/ (stderrstdout)
- Log pelaksana -*`LogGroup`*/*`logStreamPrefix`*/*`virtual-cluster-id`*/pekerjaan/*`id pekerjaan`*/wadah/*`spark-application-id`*/*`executor-pod-name`*/(stderr/stdout)

Daftar tugas berjalan

Anda dapat menjalankan `list-job-run` untuk menunjukkan keadaan tugas berjalan, seperti yang ditunjukkan contoh berikut.

```
aws emr-containers list-job-runs --virtual-cluster-id <cluster-id>
```

Jelaskan tugas berjalan

Anda dapat menjalankan `describe-job-run` untuk mendapatkan detail lebih lanjut tentang tugas, seperti status tugas, detail tugas, dan nama tugas, seperti yang ditunjukkan contoh berikut.

```
aws emr-containers describe-job-run --virtual-cluster-id cluster-id --id job-run-id
```

Membatalkan tugas berjalan

Anda dapat menjalankan `cancel-job-run` untuk membatalkan tugas berjalan, seperti yang ditunjukkan contoh berikut.

```
aws emr-containers cancel-job-run --virtual-cluster-id cluster-id --id job-run-id
```

Menjalankan skrip Spark SQL melaluiStartJobRunAPI

Amazon EMR pada rilis EKS 6.7.0 dan yang lebih tinggi menyertakan driver pekerjaan Spark SQL sehingga Anda dapat menjalankan skrip Spark SQL melalui`StartJobRun`API. Anda dapat menyediakan file entry-point SQL untuk langsung menjalankan kueri Spark SQL di Amazon EMR di EKS dengan`StartJobRun`API, tanpa modifikasi pada skrip SQL Spark yang ada. Tabel berikut mencantumkan parameter Spark yang didukung untuk pekerjaan Spark SQL melaluiStartJobRunAPI.

Anda dapat memilih dari parameter Spark berikut untuk dikirim ke pekerjaan Spark SQL. Gunakan parameter ini untuk mengganti properti Spark default.

Opsi	Deskripsi
--nama NAMA	Nama Aplikasi
-guci	Daftar dipisahkan koma guci untuk disertakan dengan driver dan mengeksekusi classpath.
--paket	Daftar dipisahkan koma koordinat maven guci untuk memasukkan pada driver dan classpaths pelaksana.
--exclude-paket	Daftar grup yang dipisahkan koma: artifactI d, untuk dikecualikan saat menyelesaikan dependensi yang disediakan dalam —packages untuk menghindari konflik ketergant ungan.
--repositori	Daftar dipisahkan koma dari repositori jarak jauh tambahan untuk mencari koordinat maven yang diberikan dengan —packages.

Opsi	Deskripsi
-file FILE	Daftar file yang dipisahkan koma untuk ditempatkan di direktori kerja masing-masing pelaksana.
--conf PROP = NILAI	Properti konfigurasi percikan.
--properti-berkas BERKAS	Path ke file dari mana untuk memuat properti tambahan.
-driver-memori MEM	Memori untuk pengemudi. Default 1024MB.
--driver-java-options	Opsi Java tambahan untuk diteruskan ke pengemudi.
--driver-library-path	Ekstra entri jalur perpustakaan untuk diteruskan ke pengemudi.
--driver-class-path	Entri classpath ekstra untuk diteruskan ke pengemudi.
--executor-memori MEM	Memori per pelaksana. Default 1GB.
-driver-core NUM	Jumlah core yang digunakan oleh pengemudi.
--total-executor-coresNUM	Total core untuk semua pelaksana.
--executor-core NUM	Jumlah core yang digunakan oleh masing-ma sing pelaksana.
-num-pelaksana NUM	Jumlah pelaksana yang akan diluncurkan.
-hivevar <key=nilai>	Substitusi variabel untuk diterapkan pada perintah Hive, misalnya,`-hivevar A=B`
-hiveconf <properti = nilai>	Nilai yang digunakan untuk properti yang diberikan.

Untuk pekerjaan Spark SQL, buatstart-job-run-request.json mengajukan dan menentukan parameter yang diperlukan untuk menjalankan pekerjaan Anda, seperti dalam contoh berikut:

```
{
  "name": "myjob",
  "virtualClusterId": "123456",
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "emr-6.7.0-latest",
  "jobDriver": {
    "sparkSqlJobDriver": {
      "entryPoint": "entryPoint_location",
       "sparkSqlParameters": "--conf spark.executor.instances=2 --conf
 spark.executor.memory=2G --conf spark.executor.cores=2 --conf spark.driver.cores=1"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.driver.memory":"2G"
         }
      }
    ],
    "monitoringConfiguration": {
      "persistentAppUI": "ENABLED",
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "my_log_group",
        "logStreamNamePrefix": "log_stream_prefix"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://my_s3_log_location"
      }
    }
  }
}
```

Status tugas berjalan

Ketika Anda mengirimkan tugas berjalan ke antrean tugas Amazon EMR di EKS, tugas berjalan memasuki status `PENDING`. Kemudian melewati status-status berikut sampai berhasil (keluar dengan kode `0`) atau gagal (keluar dengan kode bukan nol).

Tugas berjalan dapat memiliki status berikut:

- `PENDING` - Status tugas awal saat tugas berjalan dikirimkan ke Amazon EMR di EKS. Tugas sedang menunggu untuk dikirimkan ke klaster virtual, dan Amazon EMR di EKS sedang bekerja untuk mengirimkan tugas ini.
- `SUBMITTED` - Tugas berjalan yang telah berhasil dikirimkan ke klaster virtual. Penjadwal klaster kemudian mencoba untuk menjalankan tugas ini di cluster.
- `RUNNING` - Tugas berjalan yang berjalan di klaster virtual. Dalam aplikasi Spark, ini berarti bahwa proses driver Spark ada di status `running`.
- `FAILED` - Tugas berjalan yang gagal untuk dikirimkan ke klaster virtual atau yang gagal diselesaikan. LihatlahStateDetailsdanFailureReasonuntuk menemukan informasi tambahan tentang kegagalan pekerjaan ini.
- `COMPLETED` - Tugas berjalan yang telah berhasil diselesaikan.
- `CANCEL_PENDING` - Tugas berjalan telah diminta untuk pembatalan. Amazon EMR di EKS sedang mencoba untuk membatalkan tugas pada klaster virtual.
- `CANCELLED` - Tugas berjalan yang berhasil dibatalkan.

Melihat tugas di konsol Amazon EMR

Untuk melihat pekerjaan di konsol Amazon EMR, lakukan langkah-langkah berikut.

1. Di menu kiri konsol Amazon EMR, di bawah Amazon EMR di EKS, pilihCluster virtual.
2. Dari daftar cluster virtual, pilih cluster virtual yang ingin Anda lihat pekerjaan.
3. Pada tabel Tugas berjalan, pilih Lihat log untuk melihat detail tugas.

> **Note**
>
> Dukungan untuk pengalaman satu klik diaktifkan secara default. Ini dapat dimatikan dengan mengatur `persistentAppUI` ke `DISABLED` dalam `monitoringConfiguration` selama pengiriman tugas. Untuk informasi selengkapnya, lihat Melihat Antarmuka Pengguna Aplikasi Persisten.

Kesalahan umum saat menjalankan tugas

Kesalahan berikut dapat terjadi ketika Anda menjalankan API `StartJobRun`.

Pesan Kesalahan	Kondisi Kesalahan	Langkah Selanjutnya yang Disarankan
kesalahan: argumen –*argumen* diperlukan	Parameter yang diperlukan hilang.	Tambahkan argumen yang hilang ke permintaan API.
Terjadi galat (AccessDeniedException) saat memanggilStartJobRunoperasi: Pengguna:*ARN*tidak berwenang untuk melakukan: emr-kontainer:StartJobRun	Peran eksekusi hilang.	Lihat Menggunakan Menggunakan peran eksekusi tugas dengan Amazon EMR di EKS.
Terjadi galat (AccessDeniedException) saat memanggilStartJobRunoperasi: Pengguna:*ARN*tidak berwenang untuk melakukan: emr-kontainer:StartJobRun	Pemanggil tidak memiliki izin untuk peran eksekusi [format valid / tidak valid] melalui kunci kondisi.	Lihat Menggunakan peran eksekusi tugas dengan Amazon EMR di EKS.
Terjadi galat (AccessDeniedException) saat memanggilStartJobRunoperasi: Pengguna:*ARN*tidak berwenang untuk melakukan: emr-kontainer:StartJobRun	Pengirim tugas dan Peran eksekusi ARN berasal dari akun yang berbeda.	Pastikan bahwa pengirim tugas dan peran eksekusi ARN adalah dari akun AWS yang sama.
1 kesalahan validasi terdeteksi: Nilai*Peran*di 'executionRoleArn'gagal memenuhi pola ekspresi reguler ARN: ^arn: (aws [a-za-Z0-9-] *) :iam:: (\ d {12})? : (peran ((\ u002f) | (\ u002f [\ u0021-\	Pemanggil memiliki izin untuk peran eksekusi melalui kunci kondisi, tetapi peran tidak memenuhi batasan format ARN.	Berikan peran eksekusi mengikuti format ARN berikut. Lihat Menggunakan peran eksekusi tugas dengan Amazon EMR di EKS.

Pesan Kesalahan	Kondisi Kesalahan	Langkah Selanjutnya yang Disarankan
u007f] +\ u002f)) [\ w+=, .@-] +)		
Terjadi galat (Resource NotFoundException) saat memanggilStartJobR unoperasi: Cluster virtual*ID Klaster Virtual*tidak ada.	ID klaster virtual tidak ditemukan.	Menyediakan klaster virtual ID terdaftar dengan Amazon EMR di EKS.
Terjadi galat (Validati onException) saat memanggil StartJobRunoperasi: Status cluster virtual*negara*tidak valid untuk membuat sumber dayaJobRun.	Klaster virtual tidak siap untuk melaksanakan tugas.	Lihat Status klaster virtual.
Terjadi galat (Resource NotFoundException) saat memanggilStartJobR unOperasi: Rilis*MELEPASKA N* tidak ada.	Rilis yang ditentukan dalam pengiriman tugas tidak benar.	Lihat Amazon EMR pada EKS rilis.

Pesan Kesalahan	Kondisi Kesalahan	Langkah Selanjutnya yang Disarankan
Terjadi galat (AccessDeniedException) saat memanggilStartJobRunoperasi: Pengguna:*ARN*tidak berwenang untuk melakukan: emr-kontainer:StartJobRunpada sumber daya:*ARN*dengan penolakan eksplisit. Terjadi galat (AccessDeniedException) saat memanggilStartJobRunoperasi: Pengguna:*ARN*tidak berwenang untuk melakukan: emr-kontainer:StartJobRunpada sumber daya:*ARN*	Pengguna tidak berwenang untuk meneleponStartJobRun.	Lihat Menggunakan peran eksekusi tugas dengan Amazon EMR di EKS.
Terjadi galat (ValidationException) saat memanggil StartJobRunoperasi: ConfigurationOverrides.MonitoringConfiguration.S3MonitoringConfiguration.logUri gagal memenuhi kendala: %s	Jalur S3 sintaks URI tidak valid.	logURI harus dalam format s3://...

Kesalahan berikut dapat terjadi ketika Anda menjalankan API `DescribeJobRun` sebelum tugas berjalan.

Pesan Kesalahan	Kondisi Kesalahan	Langkah Selanjutnya yang Disarankan
StateDetails:JobRunpengajuan gagal.	Parameter diStartJobRuntidak valid.	Lihat Amazon EMR pada EKS rilis.

Pesan Kesalahan	Kondisi Kesalahan	Langkah Selanjutnya yang Disarankan
Klasifikasi *klasifikasi* tidak didukung. failureReason: VALIDATIO N_ERROR status: GAGAL.		
stateDetails: Klaster *ID Klaster EKS* tidak ada. failureReason: CLUSTER_U NAVAILABLE status: GAGAL	Klaster EKS tidak tersedia.	Periksa apakah klaster EKS ada dan memiliki izin yang tepat. Untuk informasi selengkapnya, lihat Menyiapka n Amazon EMR di EKS.
stateDetails: Klaster *ID Klaster EKS* tidak memiliki izin yang cukup. failureReason: CLUSTER_U NAVAILABLE status: GAGAL	Amazon EMR tidak memiliki izin untuk mengakses klaster EKS.	Verifikasi bahwa izin diatur untuk Amazon EMR pada namespace terdaftar. Untuk informasi selengkapnya, lihat Menyiapkan Amazon EMR di EKS.
stateDetails: Klaster *ID Klaster EKS* saat ini tidak dapat dijangkau. failureReason: CLUSTER_U NAVAILABLE status: GAGAL	Klaster EKS tidak dapat dijangkau.	Periksa apakah Klaster EKS ada dan memiliki izin yang tepat. Untuk informasi selengkapnya, lihat Menyiapka n Amazon EMR di EKS.

Pesan Kesalahan	Kondisi Kesalahan	Langkah Selanjutnya yang Disarankan
StateDetails:JobRunpengajuan gagal karena kesalahan internal. failureReason: INTERNAL_ERROR status: GAGAL	Kesalahan internal telah terjadi dengan klaster EKS.	N/A
stateDetails: Klaster *ID Klaster EKS* tidak memiliki sumber daya yang cukup. failureReason: USER_ERROR status: GAGAL	Ada sumber daya yang tidak mencukupi di klaster EKS untuk menjalankan tugas.	Tambahkan lebih banyak kapasitas ke grup simpul EKS atau atur EKS Autoscaler. Untuk informasi lebih lanjut, lihat Klaster Autoscaler.

Kesalahan berikut dapat terjadi ketika Anda menjalankan API `DescribeJobRun` setelah tugas berjalan.

Pesan Kesalahan	Kondisi Kesalahan	Langkah Selanjutnya yang Disarankan
StateDetails: Kesulitan memonitorJobRun. Klaster *ID Klaster EKS* tidak ada. failureReason: CLUSTER_UNAVAILABLE status: GAGAL	Klaster EKS tidak ada.	Periksa apakah Klaster EKS ada dan memiliki izin yang tepat. Untuk informasi selengkapnya, lihat Menyiapkan Amazon EMR di EKS.

Pesan Kesalahan	Kondisi Kesalahan	Langkah Selanjutnya yang Disarankan
StateDetails: Kesulitan memonitorJobRun. Klaster *ID Klaster EKS* tidak memiliki izin yang cukup. failureReason: CLUSTER_UNAVAILABLE status: GAGAL	Amazon EMR tidak memiliki izin untuk mengakses klaster EKS.	Verifikasi bahwa izin diatur untuk Amazon EMR pada namespace terdaftar. Untuk informasi selengkapnya, lihat Menyiapkan Amazon EMR di EKS.
StateDetails: Kesulitan memonitorJobRun. Klaster *ID Klaster EKS* saat ini tidak dapat dijangkau. failureReason: CLUSTER_UNAVAILABLE status: GAGAL	Klaster EKS tidak dapat dijangkau.	Periksa apakah Klaster EKS ada dan memiliki izin yang tepat. Untuk informasi selengkapnya, lihat Menyiapkan Amazon EMR di EKS.
StateDetails: Kesulitan memonitorJobRunkarena kesalahan internal failureReason: INTERNAL_ERROR status: GAGAL	Kesalahan internal telah terjadi dan mencegahJobRunpemantauan.	T/A

Kesalahan berikut dapat terjadi ketika pekerjaan tidak dapat dimulai dan pekerjaan menunggu dalam keadaan DIKIRIM selama 15 menit. Hal ini dapat disebabkan oleh kurangnya sumber daya cluster.

Pesan Kesalahan	Kondisi Kesalahan	Langkah Selanjutnya yang Disarankan
batas waktu klaster	Pekerjaan telah di negara diserahkan selama 15 menit atau lebih.	Anda dapat mengganti pengaturan default 15 menit untuk parameter ini dengan penggantian konfigurasi yang ditunjukkan di bawah ini.

Gunakan konfigurasi berikut untuk mengubah pengaturan batas waktu klaster menjadi 30 menit. Perhatikan bahwa Anda memberikan yang baru`job-start-timeout`nilai dalam detik:

```
{
"configurationOverrides": {
  "applicationConfiguration": [{
      "classification": "emr-containers-defaults",
      "properties": {
          "job-start-timeout":"1800"
      }
  }]
}
```

Menggunakan klasifikasi pengirim pekerjaan

Gambaran Umum

Amazon EMR di EKS`StartJobRun`permintaan menciptakan*pengirim pekerjaan*pod (juga dikenal sebagai*pekerjaan-pelari*pod) untuk menelurkan driver Spark. Kamu dapat mengonfigurasi selektor node untuk pod submitter pekerjaanmu dengan`emr-job-submitter`klasifikasi.

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

`jobsubmitter.node.selector.[`*`labelKey`*`]`

Menambahkan ke pemilih node dari pod submitter pekerjaan, dengan kunci*`labelKey`*dan nilai sebagai nilai konfigurasi untuk konfigurasi. Misalnya, Anda dapat mengatur`jobsubmitter.node.selector.identifier`kepada`myIdentifier`dan pod

job submitter akan memiliki pemilih node dengan nilai pengenal kuncimy`Identifier`. Untuk menambahkan beberapa kunci pemilih node, atur beberapa konfigurasi dengan awalan ini.

Sebagai praktik terbaik, kami merekomendasikan bahwa pod pengirim pekerjaan memilikipenempatan node pada Instans On Demanddan bukan pada Instans Spot. Hal ini dikarenakan suatu pekerjaan akan gagal jika Pod submitter job mengalami interupsi Spot Instance. Anda juga bisatempatkan pod pengirim pekerjaan dalam satu Availability Zone, ataumenggunakan label Kubernetes yang diterapkan pada node.

Contoh klasifikasi pengirim pekerjaan

Dalam bagian ini

- StartJobRunrequest dengan penempatan Node On-Demand untuk pod job submitter
- StartJobRunrequest dengan penempatan node Single-AZ untuk pod job submitter
- StartJobRunpermintaan dengan penempatan tipe instans Single-AZ dan Amazon EC2 untuk pod pengirim pekerjaan

`StartJobRun`request dengan penempatan Node On-Demand untuk pod job submitter

```
cat >spark-python-in-s3-nodeselector-job-submitter.json << EOF
{
  "name": "spark-python-in-s3-nodeselector",
  "virtualClusterId": "virtual-cluster-id",
  "executionRoleArn": "execution-role-arn",
  "releaseLabel": "emr-6.11.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "s3://S3-prefix/trip-count.py",
       "sparkSubmitParameters": "--conf spark.driver.cores=5  --conf
 spark.executor.memory=20G --conf spark.driver.memory=15G --conf
 spark.executor.cores=6"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.dynamicAllocation.enabled":"false"
```

```
        }
      },
      {
        "classification": "emr-job-submitter",
        "properties": {
            "jobsubmitter.node.selector.eks.amazonaws.com/capacityType": "ON_DEMAND"
        }
      }
    ],
    "monitoringConfiguration": {
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "/emr-containers/jobs",
        "logStreamNamePrefix": "demo"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://joblogs"
      }
    }
  }
}
EOF
aws emr-containers start-job-run --cli-input-json file:///spark-python-in-s3-
nodeselector-job-submitter.json
```

StartJobRunrequest dengan penempatan node Single-AZ untuk pod job submitter

```
cat >spark-python-in-s3-nodeselector-job-submitter-az.json << EOF
{
  "name": "spark-python-in-s3-nodeselector",
  "virtualClusterId": "virtual-cluster-id",
  "executionRoleArn": "execution-role-arn",
  "releaseLabel": "emr-6.11.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "s3://S3-prefix/trip-count.py",
       "sparkSubmitParameters": "--conf spark.driver.cores=5  --conf
 spark.executor.memory=20G --conf spark.driver.memory=15G --conf
 spark.executor.cores=6"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
```

```
        "classification": "spark-defaults",
        "properties": {
          "spark.dynamicAllocation.enabled":"false"
         }
      },
      {
        "classification": "emr-job-submitter",
        "properties": {
            "jobsubmitter.node.selector.topology.kubernetes.io/zone": "Availability
 Zone"
        }
      }
    ],
    "monitoringConfiguration": {
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "/emr-containers/jobs",
        "logStreamNamePrefix": "demo"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://joblogs"
      }
    }
  }
}
EOF
aws emr-containers start-job-run --cli-input-json file:///spark-python-in-s3-
nodeselector-job-submitter-az.json
```

StartJobRunpermintaan dengan penempatan tipe instans Single-AZ dan Amazon EC2 untuk pod pengirim pekerjaan

```
{
  "name": "spark-python-in-s3-nodeselector",
  "virtualClusterId": "virtual-cluster-id",
  "executionRoleArn": "execution-role-arn",
  "releaseLabel": "emr-6.11.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "s3://S3-prefix/trip-count.py",
       "sparkSubmitParameters": "--conf spark.driver.cores=5  --conf
 spark.kubernetes.pyspark.pythonVersion=3 --conf spark.executor.memory=20G
 --conf spark.driver.memory=15G --conf spark.executor.cores=6 --conf
 spark.sql.shuffle.partitions=1000"
```

```
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.dynamicAllocation.enabled":"false",
         }
      },
      {
        "classification": "emr-job-submitter",
        "properties": {
            "jobsubmitter.node.selector.topology.kubernetes.io/zone": "Availability
 Zone",
            "jobsubmitter.node.selector.node.kubernetes.io/instance-type":"m5.4xlarge"
        }
      }
    ],
    "monitoringConfiguration": {
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "/emr-containers/jobs",
        "logStreamNamePrefix": "demo"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://joblogs"
      }
    }
  }
}
```

Menggunakan templat pekerjaan

Template pekerjaan menyimpan nilai yang dapat dibagikan`StartJobRun`Pemanggilan API saat memulai pekerjaan. Ini mendukung dua kasus penggunaan:

- Untuk mencegah berulang berulang`StartJobRun`Nilai permintaan API.
- Untuk menegakkan aturan bahwa nilai-nilai tertentu harus disediakan melalui`StartJobRun`Permintaan API.

Template pekerjaan memungkinkan Anda untuk menentukan template yang dapat digunakan kembali untuk pekerjaan berjalan untuk menerapkan kustomisasi tambahan, misalnya:

- Mengkonfigurasi kapasitas komputasi pelaksana dan driver
- Menetapkan properti keamanan dan tata kelola seperti peran IAM
- Menyesuaikan image buruh pelabuhan untuk digunakan di beberapa aplikasi dan pipeline data

Membuat dan menggunakan template pekerjaan untuk memulai pekerjaan

Bagian ini menjelaskan pembuatan template pekerjaan dan menggunakan template untuk memulai pekerjaan yang dijalankan denganAWS Command Line Interface(AWS CLI).

Untuk membuat template pekerjaan

1. Buat`create-job-template-request.json`file dan menentukan parameter yang diperlukan untuk template pekerjaan Anda, seperti yang ditunjukkan dalam contoh file JSON berikut. Untuk informasi tentang semua parameter yang tersedia, lihatCreateJobTemplateAPI.

   Sebagian besar nilai yang diperlukan untuk`StartJobRun`API juga diperlukan untuk`jobTemplateData`. Jika Anda ingin menggunakan placeholder untuk parameter apa pun dan memberikan nilai saat memohonStartJobRunmenggunakan template pekerjaan, silakan lihat bagian berikutnya pada parameter template pekerjaan.

```
{
   "name": "mytemplate",
   "jobTemplateData": {
        "executionRoleArn": "iam_role_arn_for_job_execution",
        "releaseLabel": "emr-6.7.0-latest",
        "jobDriver": {
            "sparkSubmitJobDriver": {
                "entryPoint": "entryPoint_location",
                "entryPointArguments": [ "argument1","argument2",...],
                "sparkSubmitParameters": "--class <main_class> --conf
 spark.executor.instances=2 --conf spark.executor.memory=2G --conf
 spark.executor.cores=2 --conf spark.driver.cores=1"
            }
       },
       "configurationOverrides": {
            "applicationConfiguration": [
                {
```

```
                    "classification": "spark-defaults",
                    "properties": {
                         "spark.driver.memory":"2G"
                    }
                }
            ],
            "monitoringConfiguration": {
                "persistentAppUI": "ENABLED",
                "cloudWatchMonitoringConfiguration": {
                    "logGroupName": "my_log_group",
                    "logStreamNamePrefix": "log_stream_prefix"
                },
                "s3MonitoringConfiguration": {
                    "logUri": "s3://my_s3_log_location/"
                }
            }
        }
    }
}
```

2. Gunakancreate-job-templateperintah dengan path kecreate-job-template-request.jsonfile yang disimpan secara lokal.

```
aws emr-containers create-job-template \
--cli-input-json file://./create-job-template-request.json
```

Untuk memulai pekerjaan yang dijalankan menggunakan template pekerjaan

Menyediakan id cluster virtual, id template pekerjaan, dan nama pekerjaan diStartJobRunperintah, seperti yang ditunjukkan pada contoh berikut.

```
aws emr-containers start-job-run \
--virtual-cluster-id 123456 \
--name myjob \
--job-template-id 1234abcd
```

Mendefinisikan parameter template pekerjaan

Parameter template pekerjaan memungkinkan Anda untuk menentukan variabel dalam template pekerjaan. Nilai untuk variabel parameter ini perlu ditentukan saat memulai pekerjaan yang dijalankan menggunakan template pekerjaan itu. Parameter template pekerjaan ditentukan

dalam`${parameterName}`format. Anda dapat memilih untuk menentukan nilai apa pun dalam`jobTemplateData`lapangan sebagai parameter template pekerjaan. Untuk masing-masing variabel parameter template pekerjaan, tentukan tipe datanya (`STRING`atau`NUMBER`) dan opsional nilai default. Contoh di bawah ini menunjukkan bagaimana Anda dapat menentukan parameter template pekerjaan untuk lokasi titik masuk, kelas utama, dan nilai lokasi log S3.

Untuk menentukan lokasi titik masuk, kelas utama, dan lokasi log Amazon S3 sebagai parameter template pekerjaan

1. Buat`create-job-template-request.json`file dan menentukan parameter yang diperlukan untuk template pekerjaan Anda, seperti yang ditunjukkan dalam contoh file JSON berikut. Untuk informasi lebih lanjut tentang parameter, lihatCreateJobTemplateAPI.

```
{
   "name": "mytemplate",
   "jobTemplateData": {
        "executionRoleArn": "iam_role_arn_for_job_execution",
        "releaseLabel": "emr-6.7.0-latest",
        "jobDriver": {
            "sparkSubmitJobDriver": {
                "entryPoint": "${EntryPointLocation}",
                "entryPointArguments": [ "argument1","argument2",...],
                "sparkSubmitParameters": "--class ${MainClass} --conf
 spark.executor.instances=2 --conf spark.executor.memory=2G --conf
 spark.executor.cores=2 --conf spark.driver.cores=1"
            }
        },
        "configurationOverrides": {
            "applicationConfiguration": [
                {
                    "classification": "spark-defaults",
                    "properties": {
                         "spark.driver.memory":"2G"
                    }
                }
            ],
            "monitoringConfiguration": {
                "persistentAppUI": "ENABLED",
                "cloudWatchMonitoringConfiguration": {
                    "logGroupName": "my_log_group",
                    "logStreamNamePrefix": "log_stream_prefix"
                },
```

```
                "s3MonitoringConfiguration": {
                    "logUri": "${LogS3BucketUri}"
                }
            }
        },
        "parameterConfiguration": {
            "EntryPointLocation": {
                "type": "STRING"
            },
            "MainClass": {
                "type": "STRING",
                "defaultValue":"Main"
            },
            "LogS3BucketUri": {
                "type": "STRING",
                "defaultValue":"s3://my_s3_log_location/"
            }
        }
    }
}
```

2. Gunakan perintah `create-job-template` dengan jalur ke file `create-job-template-request.json` yang disimpan secara lokal atau di Amazon S3.

```
aws emr-containers create-job-template \
--cli-input-json file://./create-job-template-request.json
```

Untuk memulai menjalankan pekerjaan menggunakan template pekerjaan dengan parameter template pekerjaan

Untuk memulai pekerjaan yang dijalankan dengan template pekerjaan yang berisi parameter template pekerjaan, tentukan id template pekerjaan serta nilai untuk parameter template pekerjaan di`StartJobRun`Permintaan API seperti yang ditunjukkan di bawah ini.

```
aws emr-containers start-job-run \
--virtual-cluster-id 123456 \
--name myjob \
--job-template-id 1234abcd \
--job-template-parameters '{"EntryPointLocation": "entry_point_location","MainClass":
 "ExampleMainClass","LogS3BucketUri": "s3://example_s3_bucket/"}'
```

Mengontrol akses ke templat pekerjaan

`StartJobRun`kebijakan memungkinkan Anda menegakkan bahwa pengguna atau peran hanya dapat menjalankan pekerjaan menggunakan template pekerjaan yang Anda tentukan dan tidak dapat dijalankan`StartJobRun`operasi tanpa menggunakan template pekerjaan yang ditentukan. Untuk mencapai hal ini, pertama pastikan bahwa Anda memberikan pengguna atau peran izin baca untuk template pekerjaan tertentu seperti yang ditunjukkan di bawah ini.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "emr-containers:DescribeJobTemplate",
            "Resource": [
                "job_template_1_arn",
                "job_template_2_arn",
                ...
            ]
        }
    ]
}
```

Untuk menegakkan bahwa pengguna atau peran dapat memohon`StartJobRun`operasi hanya ketika menggunakan template pekerjaan tertentu, Anda dapat menetapkan berikut`StartJobRun`izin kebijakan untuk pengguna atau peran tertentu.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "emr-containers:StartJobRun",
            "Resource": [
                "virtual_cluster_arn",
            ],
            "Condition": [
                "StringEquals": {
                    "emr-containers:JobTemplateArn": [
                        "job_template_1_arn",
                        "job_template_2_arn",
                        ...
```

```
                    ]
                }
                ]
            }
        }
    ]
}
```

Jika template pekerjaan menentukan parameter template pekerjaan di dalam bidang peran eksekusi ARN, maka pengguna akan dapat memberikan nilai untuk parameter ini dan dengan demikian dapat memanggil`StartJobRun`menggunakan peran eksekusi sewenang-wenang. Untuk membatasi peran eksekusi yang dapat diberikan pengguna, lihatMengontrol akses ke peran eksekusidi[Menggunakan peran eksekusi tugas dengan Amazon EMR di EKS](#).

Jika tidak ada kondisi yang ditentukan di atas`StartJobRun`kebijakan tindakan untuk pengguna tertentu atau peran, pengguna atau peran akan diizinkan untuk memohon`StartJobRun`tindakan pada cluster virtual yang ditentukan menggunakan template pekerjaan sewenang-wenang bahwa mereka telah membaca akses ke atau menggunakan peran eksekusi sewenang-wenang.

Menggunakan templat pod

Dimulai dengan Amazon EMR versi 5.33.0 atau 6.3.0, Amazon EMR di EKS mendukung fitur templat pod Spark. Sebuah pod adalah sekelompok satu atau lebih kontainer, dengan penyimpanan bersama dan sumber daya jaringan, dan spesifikasi cara menjalankan kontainer. Templat pod adalah spesifikasi yang menentukan cara menjalankan setiap pod. Anda dapat menggunakan file templat pod untuk menentukan konfigurasi driver atau pelaksana pod yang tidak didukung konfigurasi Spark. Untuk informasi selengkapnya tentang fitur templat pod Spark, lihat Templat Pod.

> **Note**
>
> Fitur templat pod hanya bekerja dengan driver dan pelaksana pod. Anda tidak dapat mengonfigurasi pod pengendali tugas menggunakan templat pod.

Skenario umum

Anda dapat menentukan cara menjalankan tugas Spark di klaster EKS yang dibagikan dengan menggunakan templat pod dengan Amazon EMR di EKS dan menghemat biaya dan meningkatkan pemanfaatan sumber daya dan performa.

- Untuk mengurangi biaya, Anda dapat menjadwalkan tugas driver Spark untuk dijalankan di Instans Sesuai Permintaan Amazon EC2 sambil menjadwalkan tugas pelaksana Spark untuk dijalankan di Instans Spot Amazon EC2.
- Untuk meningkatkan pemanfaatan sumber daya, Anda dapat mendukung beberapa tim yang menjalankan beban kerja mereka pada klaster EKS yang sama. Setiap tim akan mendapatkan grup simpul Amazon EC2 yang ditunjuk untuk tempat menjalankan beban kerja mereka. Anda dapat menggunakan templat pod untuk menerapkan toleransi yang sesuai untuk beban kerja mereka.
- Untuk meningkatkan pemantauan, Anda dapat menjalankan kontainer pencatatan terpisah untuk meneruskan log pada aplikasi pemantauan yang ada.

Sebagai contoh, file templat pod berikut menunjukkan skenario penggunaan umum.

```
apiVersion: v1
kind: Pod
spec:
  volumes:
    - name: source-data-volume
      emptyDir: {}
    - name: metrics-files-volume
      emptyDir: {}
  nodeSelector:
    eks.amazonaws.com/nodegroup: emr-containers-nodegroup
  containers:
  - name: spark-kubernetes-driver # This will be interpreted as driver Spark main
 container
    env:
      - name: RANDOM
        value: "random"
    volumeMounts:
      - name: shared-volume
        mountPath: /var/data
      - name: metrics-files-volume
        mountPath: /var/metrics/data
  - name: custom-side-car-container # Sidecar container
    image: <side_car_container_image>
    env:
      - name: RANDOM_SIDECAR
        value: random
    volumeMounts:
      - name: metrics-files-volume
        mountPath: /var/metrics/data
```

```
    command:
      - /bin/sh
      - '-c'
      -  <command-to-upload-metrics-files>
  initContainers:
  - name: spark-init-container-driver # Init container
    image: <spark-pre-step-image>
    volumeMounts:
      - name: source-data-volume # Use EMR predefined volumes
        mountPath: /var/data
    command:
      - /bin/sh
      - '-c'
      -  <command-to-download-dependency-jars>
```

Templat pod menyelesaikan tugas berikut:

- Tambahkan kontainer initbaru yang dieksekusi sebelum kontainer utama Spark dimulai. Kontainer init berbagiEmptyDirvolumedisebut`source-data-volume`dengan wadah utama Spark. Anda dapat meminta kontainer init Anda menjalankan langkah inisialisasi, seperti mengunduh dependensi atau menghasilkan data input. Kemudian kontainer utama Spark mengkonsumsi data.
- Tambahkan kontainer sidecar lain yang dieksekusi bersama dengan kontainer utama Spark. Kedua kontainer berbagi volume `EmptyDir` lain yang disebut `metrics-files-volume`. Tugas Spark Anda dapat menghasilkan metrik, seperti metrik Prometheus. Kemudian tugas Spark dapat menempatkan metrik ke dalam file dan meminta kontainer sidecar mengunggah file ke sistem BI Anda sendiri untuk analisis di masa mendatang.
- Tambahkan variabel lingkungan baru ke kontainer utama Spark. Anda dapat meminta tugas Anda mengkonsumsi variabel lingkungan.
- Definisikan sebuah simpul pemilih, sehingga pod hanya dijadwalkan pada grup simpul `emr-containers-nodegroup`. Hal ini membantu untuk mengisolasi sumber daya komputasi di seluruh tugas dan tim.

Mengaktifkan templat pod dengan Amazon EMR di EKS

Untuk mengaktifkan fitur templat pod dengan Amazon EMR pada EKS, konfigurasi properti Spark `spark.kubernetes.driver.podTemplateFile` dan `spark.kubernetes.executor.podTemplateFile` untuk menunjuk ke file templat pod di

Amazon S3. Spark kemudian mengunduh file templat pod dan menggunakannya untuk membangun driver dan pod pelaksana.

> **Note**
>
> Spark menggunakan peran eksekusi tugas untuk memuat templat pod, sehingga peran eksekusi pekerjaan harus memiliki izin untuk mengakses Amazon S3 untuk memuat templat pod. Untuk informasi selengkapnya, lihat Untuk membuat peran eksekusi tugas.

Anda dapat menggunakan `SparkSubmitParameters` untuk menentukan jalur Amazon S3 ke templat pod, seperti yang ditunjukkan file tugas berjalan JSON berikut.

```
{
  "name": "myjob",
  "virtualClusterId": "123456",
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "release_label",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "entryPoint_location",
      "entryPointArguments": ["argument1", "argument2", ...],
       "sparkSubmitParameters": "--class <main_class> \
         --conf
 spark.kubernetes.driver.podTemplateFile=s3://path_to_driver_pod_template \
         --conf
 spark.kubernetes.executor.podTemplateFile=s3://path_to_executor_pod_template \
         --conf spark.executor.instances=2 \
         --conf spark.executor.memory=2G \
         --conf spark.executor.cores=2 \
         --conf spark.driver.cores=1"
    }
  }
}
```

Atau, Anda dapat menggunakan `configurationOverrides` untuk menentukan jalur Amazon S3 ke templat pod, seperti yang ditunjukkan file tugas berjalan JSON berikut.

```
{
  "name": "myjob",
  "virtualClusterId": "123456",
```

```
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "release_label",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "entryPoint_location",
      "entryPointArguments": ["argument1", "argument2", ...],
       "sparkSubmitParameters": "--class <main_class> \
         --conf spark.executor.instances=2 \
         --conf spark.executor.memory=2G \
         --conf spark.executor.cores=2 \
         --conf spark.driver.cores=1"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.driver.memory":"2G",
          "spark.kubernetes.driver.podTemplateFile":"s3://path_to_driver_pod_template",

 "spark.kubernetes.executor.podTemplateFile":"s3://path_to_executor_pod_template"
         }
      }
    ]
  }
}
```

> **Note**
>
> 1. Anda harus mengikuti pedoman keamanan saat menggunakan fitur templat pod dengan Amazon EMR di EKS, seperti mengisolasi kode aplikasi yang tidak dipercaya. Untuk informasi selengkapnya, lihat Praktik terbaik keamanan Amazon EMR di EKS.
> 2. Anda tidak dapat mengubah nama kontainer utama Spark dengan menggunakan `spark.kubernetes.driver.podTemplateContainerName` dan `spark.kubernetes.executor.podTemplateContainerName`, karena nama-nama ini di-hardcode sebagai `spark-kubernetes-driver` dan `spark-kubernetes-executors`. Jika Anda ingin menyesuaikan kontainer utama Spark, Anda harus menentukan kontainer dalam templat pod dengan nama-nama di-hardcode ini.

Bidang templat pod

Pertimbangkan pembatasan bidang berikut ketika mengonfigurasi templat pod dengan Amazon EMR di EKS.

- Amazon EMR di EKS hanya mengizinkan bidang-bidang berikut dalam templat pod untuk mengaktifkan penjadwalan tugas yang tepat.

  Ini adalah bidang tingkat pod yang diizinkan:

  - `apiVersion`
  - `kind`
  - `metadata`
  - `spec.activeDeadlineSeconds`
  - `spec.affinity`
  - `spec.containers`
  - `spec.enableServiceLinks`
  - `spec.ephemeralContainers`
  - `spec.hostAliases`
  - `spec.hostname`
  - `spec.imagePullSecrets`
  - `spec.initContainers`
  - `spec.nodeName`
  - `spec.nodeSelector`
  - `spec.overhead`
  - `spec.preemptionPolicy`
  - `spec.priority`
  - `spec.priorityClassName`
  - `spec.readinessGates`
  - `spec.runtimeClassName`
  - `spec.schedulerName`
  - `spec.subdomain`
  - `spec.terminationGracePeriodSeconds`

- `spec.tolerations`
- `spec.topologySpreadConstraints`
- `spec.volumes`

Ini adalah bidang tingkat kontainer utama Spark yang diizinkan:

- `env`
- `envFrom`
- `name`
- `lifecycle`
- `livenessProbe`
- `readinessProbe`
- `resources`
- `startupProbe`
- `stdin`
- `stdinOnce`
- `terminationMessagePath`
- `terminationMessagePolicy`
- `tty`
- `volumeDevices`
- `volumeMounts`
- `workingDir`

Saat Anda menggunakan bidang yang tidak diizinkan dalam templat pod, Spark melempar pengecualian dan tugas gagal. Contoh berikut menunjukkan pesan kesalahan dalam log pengendali Spark karena bidang tidak diizinkan.

```
Executor pod template validation failed.
Field container.command in Spark main container not allowed but specified.
```

- Amazon EMR di EKS menentukan terlebih dahulu parameter berikut dalam templat pod. Bidang yang Anda tentukan dalam templat pod tidak boleh tumpang tindih dengan bidang ini.

  Ini adalah nama volume yang telah ditetapkan:

  - `emr-container-communicate`

- `config-volume`
- `emr-container-application-log-dir`
- `emr-container-event-log-dir`
- `temp-data-dir`
- `mnt-dir`
- `home-dir`
- `emr-container-s3`

Ini adalah pemasangan volume yang telah ditetapkan yang hanya berlaku untuk kontainer utama Spark:

- Nama:`emr-container-communicate`;MountPath:`/var/log/fluentd`
- Nama:`emr-container-application-log-dir`;MountPath:`/var/log/spark/user`
- Nama:`emr-container-event-log-dir`;MountPath:`/var/log/spark/apps`
- Nama:`mnt-dir`;MountPath:`/mnt`
- Nama:`temp-data-dir`;MountPath:`/tmp`
- Nama:`home-dir`;MountPath:`/home/hadoop`

Ini adalah variabel lingkungan yang telah ditetapkan yang hanya berlaku untuk kontainer utama Spark:

- `SPARK_CONTAINER_ID`
- `K8S_SPARK_LOG_URL_STDERR`
- `K8S_SPARK_LOG_URL_STDOUT`
- `SIDECAR_SIGNAL_FILE`

> **Note**
>
> Anda masih dapat menggunakan volume standar yang telah ditentukan ini dan memasangnya ke kontainer sidecar tambahan Anda. Misalnya, Anda dapat menggunakan `emr-container-application-log-dir` dan memasangnya ke kontainer sidecar Anda sendiri yang didefinisikan dalam templat pod.

Jika bidang yang Anda tentukan bertentangan dengan salah satu bidang yang telah ditetapkan dalam templat pod, Spark melempar pengecualian dan tugas gagal. Contoh berikut menunjukkan

pesan kesalahan dalam log aplikasi Spark karena bertentangan dengan bidang yang telah ditentukan.

```
Defined volume mount path on main container must not overlap with reserved mount
 paths: [<reserved-paths>]
```

Pertimbangan kontainer sidecar

Amazon EMR mengendalikan siklus hidup pod yang disediakan oleh Amazon EMR di EKS. Kontainer sidecar harus mengikuti siklus hidup yang sama dengan kontainer utama Spark. Jika Anda menyuntikkan kontainer sidecar tambahan ke pod Anda, kami sarankan Anda mengintegrasikan dengan manajemen siklus hidup pod yang didefinisikan Amazon EMR sehingga kontainer sidecar dapat berhenti sendiri ketika kontainer utama Spark keluar.

Untuk mengurangi biaya, sebaiknya Anda menerapkan proses yang mencegah driver pod dengan kontainer sidecar terus berjalan setelah tugas Anda selesai. Driver Spark menghapus pod pelaksana saat pelaksana selesai. Namun, ketika program driver selesai, kontainer sidecar tambahan terus berjalan. Pod ditagihkan sampai Amazon EMR di EKS membersihkan pod driver, biasanya kurang dari satu menit setelah kontainer utama driver Spark selesai. Untuk mengurangi biaya, Anda dapat mengintegrasikan kontainer sidecar tambahan Anda dengan mekanisme manajemen siklus hidup yang didefinisikan Amazon EMR di EKS untuk driver dan pelaksana pod, seperti yang dijelaskan pada bagian berikut.

Kontainer utama Spark dalam driver dan pelaksana pod mengirimkan `heartbeat` ke `/var/log/fluentd/main-container-terminated` file setiap dua detik. Dengan menambahkan Amazon EMR yang telah ditetapkan `emr-container-communicate` pemasangan volume ke kontainer sidecar Anda, Anda dapat menentukan sub-proses kontainer sidecar Anda untuk secara berkala melacak waktu terakhir diubah untuk file ini. Sub-proses kemudian berhenti sendiri jika menemukan bahwa kontainer utama Spark menghentikan `heartbeat` untuk durasi yang lebih lama.

Contoh berikut menunjukkan sub-proses yang melacak file detak jantung dan berhenti sendiri. Ganti *`your_volume_mount`* dengan jalur di mana anda memasang volume yang telah ditentukan sebelumnya. Script dipaketkan di dalam gambar yang digunakan oleh kontainer sidecar. Dalam file templat pod, Anda dapat menentukan kontainer sidecar dengan perintah `sub_process_script.sh` dan `main_command` berikut.

```
MOUNT_PATH="your_volume_mount"
FILE_TO_WATCH="$MOUNT_PATH/main-container-terminated"
```

```
INITIAL_HEARTBEAT_TIMEOUT_THRESHOLD=60
HEARTBEAT_TIMEOUT_THRESHOLD=15
SLEEP_DURATION=10

function terminate_main_process() {
  # Stop main process
}

# Waiting for the first heartbeat sent by Spark main container
echo "Waiting for file $FILE_TO_WATCH to appear..."
start_wait=$(date +%s)
while ! [[ -f "$FILE_TO_WATCH" ]]; do
    elapsed_wait=$(expr $(date +%s) - $start_wait)
    if [ "$elapsed_wait" -gt "$INITIAL_HEARTBEAT_TIMEOUT_THRESHOLD" ]; then
        echo "File $FILE_TO_WATCH not found after $INITIAL_HEARTBEAT_TIMEOUT_THRESHOLD
 seconds; aborting"
        terminate_main_process
        exit 1
    fi
    sleep $SLEEP_DURATION;
done;
echo "Found file $FILE_TO_WATCH; watching for heartbeats..."

while [[ -f "$FILE_TO_WATCH" ]]; do
    LAST_HEARTBEAT=$(stat -c %Y $FILE_TO_WATCH)
    ELAPSED_TIME_SINCE_AFTER_HEARTBEAT=$(expr $(date +%s) - $LAST_HEARTBEAT)
    if [ "$ELAPSED_TIME_SINCE_AFTER_HEARTBEAT" -gt "$HEARTBEAT_TIMEOUT_THRESHOLD" ];
 then
        echo "Last heartbeat to file $FILE_TO_WATCH was more than
 $HEARTBEAT_TIMEOUT_THRESHOLD seconds ago at $LAST_HEARTBEAT; terminating"
        terminate_main_process
        exit 0
    fi
    sleep $SLEEP_DURATION;
done;
echo "Outside of loop, main-container-terminated file no longer exists"

# The file will be deleted once the fluentd container is terminated

echo "The file $FILE_TO_WATCH doesn't exist any more;"
terminate_main_process
exit 0
```

Menggunakan kebijakan coba ulang pekerjaan

Di Amazon EMR pada EKS versi 6.9.0 dan yang lebih baru, Anda dapat menetapkan kebijakan coba ulang untuk pekerjaan Anda berjalan. Coba lagi kebijakan menyebabkan pod driver pekerjaan dimulai ulang secara otomatis jika gagal atau dihapus. Ini membuat pekerjaan streaming Spark yang berjalan lama lebih tahan terhadap kegagalan.

Menetapkan kebijakan coba ulang untuk suatu pekerjaan

Untuk mengonfigurasi kebijakan coba ulang, Anda menyediakan`RetryPolicyConfiguration`bidang menggunakan[StartJobRun](#)API. Contoh`retryPolicyConfiguration`ditampilkan di sini:

```
aws emr-containers start-job-run \
--virtual-cluster-id cluster_id \
--name sample-job-name \
--execution-role-arn execution-role-arn \
--release-label emr-6.9.0-latest \
--job-driver '{
  "sparkSubmitJobDriver": {
    "entryPoint": "local:///usr/lib/spark/examples/src/main/python/pi.py",
    "entryPointArguments": [ "2" ],
    "sparkSubmitParameters": "--conf spark.executor.instances=2 --conf
 spark.executor.memory=2G --conf spark.executor.cores=2 --conf spark.driver.cores=1"
  }
}' \
--retry-policy-configuration '{
    "maxAttempts": 5
  }' \
--configuration-overrides '{
  "monitoringConfiguration": {
    "cloudWatchMonitoringConfiguration": {
      "logGroupName": "my_log_group_name",
      "logStreamNamePrefix": "my_log_stream_prefix"
    },
    "s3MonitoringConfiguration": {
       "logUri": "s3://DOC-EXAMPLE-BUCKET-LOGGING"
    }
  }
}'
```

> **Note**
>
> `retryPolicyConfiguration`hanya tersedia dariAWS CLI1.27.68 versi dan seterusnya. Untuk memperbaruiAWS CLIke versi terbaru, lihatMenginstal atau memperbarui versi terbaru dariAWS CLI

KonfigurasikanmaxAttemptsfield dengan jumlah maksimum yang kamu inginkan agar pod driver job di-restart jika gagal atau dihapus. Interval eksekusi antara dua upaya coba ulang pengemudi pekerjaan adalah interval percobaan ulang eksponensial (10 detik, 20 detik, 40 detik...) yang dibatasi pada 6 menit, seperti yang dijelaskan dalamDokumentasi Kubernetes.

> **Note**
>
> Setiap eksekusi pengemudi pekerjaan tambahan akan ditagih sebagai pekerjaan lain, dan akan dikenakanAmazon EMR pada harga EKS.

Coba lagi nilai konfigurasi kebijakan

- Kebijakan coba ulang default untuk suatu pekerjaan: `StartJobRun`menyertakan kebijakan coba lagi yang ditetapkan ke 1 upaya maksimum secara default. Anda dapat mengonfigurasi kebijakan coba lagi sesuai keinginan.

  > **Note**
  >
  > Jika`maxAttempts`dari`retryPolicyConfiguration`diatur ke 1, itu berarti tidak ada percobaan ulang yang akan dilakukan untuk memunculkan pod driver pada kegagalan.

- Menonaktifkan kebijakan coba lagi untuk suatu pekerjaan:Untuk menonaktifkan kebijakan coba lagi, tetapkan nilai upaya maks diretryPolicyConfigurationuntuk 1.

  ```
  "retryPolicyConfiguration": {
      "maxAttempts": 1
  }
  ```

- Tetapkan MaxUpaya untuk pekerjaan dalam rentang yang valid: `StartJobRun`panggilan akan gagal jika`maxAttempts`nilai berada di luar rentang yang valid. Yang valid`maxAttempts`Rentang adalah dari 1 hingga 2.147.483.647 (32-bit integer), rentang yang didukung untuk Kubernetes

'`backOffLimit`pengaturan konfigurasi. Untuk informasi lebih lanjut, lihatKebijakan kegagalan Pod backoffdalam dokumentasi Kubernetes. Jika`maxAttempts`nilai tidak valid, pesan galat berikut dikembalikan:

```
{
 "message": "Retry policy configuration's parameter value of maxAttempts is invalid"
}
```

Mengambil status kebijakan percobaan ulang untuk suatu pekerjaan

Anda dapat melihat status upaya percobaan ulang untuk pekerjaan dengan`ListJobRuns`dan`DescribeJobRun`API. Setelah Anda meminta pekerjaan dengan konfigurasi kebijakan coba ulang yang diaktifkan,`ListJobRun`dan`DescribeJobRun`tanggapan akan berisi status kebijakan coba lagi di`RetryPolicyExecution`bidang. Selain itu,`DescribeJobRun`respon akan berisi`RetryPolicyConfiguration`yang masukan di`StartJobRun`permintaan untuk pekerjaan itu.

Respons sampel

ListJobRuns response

```
{
  "jobRuns": [
    ...
    ...
    "retryPolicyExecution" : {
      "currentAttemptCount": 2
    }
    ...
    ...
  ]
}
```

DescribeJobRun response

```
{
  ...
  ...
  "retryPolicyConfiguration": {
    "maxAttempts": 5
```

```
    },
    "retryPolicyExecution" : {
     "currentAttemptCount": 2
   },
   ...
   ...
}
```

Bidang ini tidak akan terlihat ketika kebijakan coba lagi dinonaktifkan dalam pekerjaan, seperti yang dijelaskan di bawah ini diCoba lagi nilai konfigurasi kebijakan.

Memantau pekerjaan dengan kebijakan coba lagi

Saat Anda mengaktifkan kebijakan coba ulang, aCloudWatchacara yang dihasilkan untuk setiap driver pekerjaan yang dibuat. Untuk berlangganan acara ini, siapkanCloudWatchaturan acara menggunakan perintah berikut:

```
aws events put-rule \
--name cwe-test \
--event-pattern '{"detail-type": ["EMR Job Run New Driver Attempt"]}'
```

Acara ini akan mengembalikan informasi tentang`newDriverPodName`,`newDriverCreatedAt`stempel waktu,`previousDriverFailureMessage`, dan`currentAttemptCount`dari pengemudi pekerjaan. Peristiwa ini tidak akan dibuat jika kebijakan coba lagi dinonaktifkan.

Untuk informasi lebih lanjut tentang cara memantau pekerjaan Anda denganCloudWatchperistiwa, lihatPantau pekerjaan dengan Amazon CloudWatch Events.

Menemukan log untuk driver dan pelaksana

Nama pod driver mengikuti format`spark-<job id>-driver-<random-suffix>`. Sama`random-suffix`ditambahkan ke nama pod pelaksana yang dihasilkan pengemudi. Bila Anda menggunakan ini`random-suffix`, Anda dapat menemukan log untuk driver dan pelaksana terkait. Yang`random-suffix`hanya hadir jikakebijakan coba lagi diaktifkanuntuk pekerjaan; jika tidak,`random-suffix`tidak ada.

Untuk informasi selengkapnya tentang cara mengonfigurasi pekerjaan dengan konfigurasi pemantauan untuk pencatatan, lihatJalankan aplikasi Spark park park park park park park.

Menggunakan rotasi log peristiwa Spark

Dengan Amazon EMR 6.3.0 dan versi lebih baru, Anda dapat mengaktifkan fitur rotasi log peristiwa Spark untuk Amazon EMR di EKS. Alih-alih menghasilkan file log peristiwa tunggal, fitur ini merotasikan file berdasarkan interval waktu terkonfigurasi Anda dan menghapus file log peristiwa terlama.

Merotasi log peristiwa Spark dapat membantu Anda menghindari potensi masalah dengan file log peristiwa Spark besar yang dihasilkan untuk tugas yang dijalankan atau di-stream dalam jangka panjang. Misalnya, Anda memulai tugas Spark berjangka panjang dengan log peristiwa yang diaktifkan dengan parameter `persistentAppUI`. Driver Spark menghasilkan file log peristiwa. Jika tugas berjalan selama berjam-jam atau berhari-hari dan ada ruang disk terbatas pada simpul Kubernetes, file log peristiwa dapat mengkonsumsi semua ruang disk yang tersedia. Mengaktifkan fitur rotasi log peristiwa Spark memecahkan masalah dengan membelah file log ke beberapa file dan menghapus file terlama.

> **Note**
>
> Fitur ini hanya berfungsi dengan Amazon EMR di EKS. Amazon EMR yang berjalan di Amazon EC2 tidak mendukung rotasi log peristiwa Spark.

Untuk mengaktifkan fitur rotasi log peristiwa Spark, konfigurasi parameter Spark berikut:

- `spark.eventLog.rotation.enabled` - menyalakan rotasi log. Ini dinonaktifkan secara default dalam file konfigurasi Spark. Atur ke betul untuk mengaktifkan fitur ini.
- `spark.eventLog.rotation.interval` - menentukan interval waktu untuk rotasi log. Nilai minimum adalah 60 detik. Nilai default adalah 300 detik.
- `spark.eventLog.rotation.minFileSize` - menentukan ukuran file minimum untuk merotasikan file log. Nilai minimum dan default adalah 1 MB.
- `spark.eventLog.rotation.maxFilesToRetain` - menentukan berapa banyak dile log yang dirotasikan untuk disimpan selama pembersihan. Rentang validnya adalah 1 hingga 10. Nilai default adalah 2.

Anda dapat menentukan parameter ini dalam bagian `sparkSubmitParameters` dari API `StartJobRun`, seperti yang ditunjukkan contoh berikut.

```
"sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi --conf
 spark.eventLog.rotation.enabled=true --conf spark.eventLog.rotation.interval=300 --
conf spark.eventLog.rotation.minFileSize=1m --conf
 spark.eventLog.rotation.maxFilesToRetain=2"
```

Menggunakan rotasi log kontainer Spark

Dengan Amazon EMR 6.11.0 dan versi lebih baru, Anda dapat mengaktifkan fitur rotasi log kontainer Spark untuk Amazon EMR di EKS. Alih-alih menghasilkan satu`stdout`atau`stderr`log file, fitur ini memutar file berdasarkan ukuran rotasi Anda dikonfigurasi dan menghapus file log tertua dari wadah.

Memutar log kontainer Spark dapat membantu Anda menghindari potensi masalah dengan file log Spark besar yang dihasilkan untuk pekerjaan yang berjalan lama atau streaming. Misalnya, Anda mungkin memulai pekerjaan Spark yang berjalan lama, dan driver Spark menghasilkan file log kontainer. Jika pekerjaan berjalan selama berjam-jam atau hari dan ada ruang disk terbatas pada node Kubernetes, file log kontainer dapat menggunakan semua ruang disk yang tersedia. Saat mengaktifkan rotasi log kontainer Spark, Anda membagi file log menjadi beberapa file, dan menghapus file terlama.

Untuk mengaktifkan fitur rotasi log kontainer Spark, konfigurasikan parameter Spark berikut:

`containerLogRotationConfiguration`

Sertakan parameter ini di`monitoringConfiguration`untuk mengaktifkan rotasi log. Ini dinonaktifkan secara default. Anda harus menggunakan`containerLogRotationConfiguration`selain`s3MonitoringConfiguration`.

`rotationSize`

Yang`rotationSize`parameter menentukan ukuran file untuk rotasi log. Kisaran nilai yang mungkin adalah dari`2KB`kepada`2GB`. Bagian satuan numerik dari`rotationSize`parameter dilewatkan sebagai integer. Karena nilai desimal tidak didukung, Anda dapat menentukan ukuran rotasi 1,5GB, misalnya, dengan nilai`1500MB`.

`maxFilesToKeep`

Yang`maxFilesToKeep`parameter menentukan jumlah maksimum file untuk mempertahankan dalam wadah setelah rotasi telah terjadi. Nilai minimum adalah 1, dan nilai maksimumnya adalah 50.

Anda dapat menentukan parameter ini dalam bagian `monitoringConfiguration` dari API `StartJobRun`, seperti yang ditunjukkan contoh berikut. Dalam contoh ini, dengan`rotationSize = "10 MB"`dan`maxFilesToKeep = 3`, Amazon EMR di EKS memutar log Anda pada 10 MB, menghasilkan file log baru, dan kemudian membersihkan file log tertua setelah jumlah file log mencapai 3.

```
{
  "name": "my-long-running-job",
  "virtualClusterId": "123456",
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "emr-6.11.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "entryPoint_location",
      "entryPointArguments": ["argument1", "argument2", ...],
       "sparkSubmitParameters": "--class main_class --conf spark.executor.instances=2
 --conf spark.executor.memory=2G --conf spark.executor.cores=2 --conf
 spark.driver.cores=1"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.driver.memory":"2G"
         }
      }
    ],
    "monitoringConfiguration": {
      "persistentAppUI": "ENABLED",
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "my_log_group",
        "logStreamNamePrefix": "log_stream_prefix"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://my_s3_log_location"
      },
      "containerLogRotationConfiguration": {
        "rotationSize":"10MB",
        "maxFilesToKeep":"3"
      }
    }
```

```
  }
}
```

Untuk memulai pekerjaan yang dijalankan dengan rotasi log kontainer Spark, sertakan jalur ke file json yang Anda konfigurasikan dengan parameter ini di`StartJobRun`perintah.

```
aws emr-containers start-job-run \
--cli-input-json file://path-to-json-request-file
```

Menggunakan penskalaan otomatis vertikal dengan pekerjaan Amazon EMR Spark

Amazon EMR pada penskalaan otomatis vertikal EKS secara otomatis menyetel memori dan sumber daya CPU untuk beradaptasi dengan kebutuhan beban kerja yang Anda sediakan untuk aplikasi Amazon EMR Spark. Ini menyederhanakan manajemen sumber daya.

Untuk melacak pemanfaatan sumber daya real-time dan historis dari aplikasi Amazon EMR Spark Anda, penskalaan otomatis vertikal memanfaatkan Kubernetes Vertical Pod Autoscaler (VPA). Kemampuan penskalaan otomatis vertikal menggunakan data yang dikumpulkan VPA untuk secara otomatis menyetel memori dan sumber daya CPU yang ditetapkan ke aplikasi Spark Anda. Proses yang disederhanakan ini meningkatkan keandalan dan mengoptimalkan biaya.

Topik

- Menyiapkan penskalaan otomatis vertikal untuk Amazon EMR di EKS
- Memulai dengan penskalaan otomatis vertikal untuk Amazon EMR di EKS
- Mengkonfigurasi penskalaan otomatis vertikal untuk Amazon EMR di EKS
- Memantau penskalaan otomatis vertikal untuk Amazon EMR di EKS
- Copot pemasangan EMR Amazon pada operator penskalaan otomatis vertikal EKS

Menyiapkan penskalaan otomatis vertikal untuk Amazon EMR di EKS

Topik ini membantu Anda menyiapkan klaster Amazon EKS Anda untuk mengirimkan pekerjaan Amazon EMR Spark dengan penskalaan otomatis vertikal. Proses penyiapan mengharuskan Anda untuk mengonfirmasi atau menyelesaikan tugas di bagian berikut:

Topik

Prasyarat

Selesaikan tugas-tugas berikut sebelum Anda menginstal operator Kubernetes penskalaan otomatis vertikal di klaster Anda. Jika Anda telah menyelesaikan salah satu prasyarat, Anda dapat melewatinya dan melanjutkan ke yang berikutnya.

- Instal AWS CLI— Jika Anda sudah menginstalAWS CLI, konfirmasikan bahwa Anda memiliki versi terbaru.
- Instal kubectl — kubectl adalah alat baris perintah yang Anda gunakan untuk berkomunikasi dengan server API Kubernetes. Anda memerlukan kubectl untuk menginstal dan memantau artefak terkait penskalaan otomatis vertikal di cluster Amazon EKS Anda.
- Instal Operator SDK — Amazon EMR di EKS menggunakan Operator SDK sebagai manajer paket untuk masa pakai operator penskalaan otomatis vertikal yang Anda instal di cluster Anda.
- Instal Docker - Anda memerlukan akses ke CLI Docker untuk mengautentikasi dan mengambil gambar Docker terkait penskalaan otomatis vertikal untuk dipasang di cluster Amazon EKS Anda.
- Instal server Kubernetes Metrics — Anda harus menginstal server metrik terlebih dahulu agar autoscaler pod vertikal dapat mengambil metrik dari server API Kubernetes.
- Siapkan EKS cluster Amazon(versi 1.24 atau lebih tinggi) - Penskalaan otomatis vertikal didukung dengan Amazon EKS versi 1.24 dan lebih tinggi. Setelah Anda membuat cluster, daftarkan untuk digunakan dengan Amazon EMR.
- Pilih URI gambar dasar EMR Amazon (rilis 6.10.0 atau lebih tinggi) — Penskalaan otomatis vertikal didukung dengan rilis Amazon EMR 6.10.0 dan yang lebih tinggi.

Instal Operator Lifecycle Manager (OLM) di kluster Amazon EKS

Gunakan Operator SDK CLI untuk menginstal Operator Lifecycle Manager (OLM) di Amazon EMR pada kluster EKS tempat Anda ingin mengatur penskalaan otomatis vertikal, seperti yang ditunjukkan pada contoh berikut. Setelah mengaturnya, Anda dapat menggunakan OLM untuk menginstal dan mengelola siklus hidup operator penskalaan otomatis vertikal Amazon EMR.

```
operator-sdk olm install
```

Untuk memvalidasi instalasi, jalankan `olm status` perintah:

```
operator-sdk olm status
```

Verifikasi bahwa perintah mengembalikan hasil yang sukses, mirip dengan contoh output berikut:

```
INFO[0007] Successfully got OLM status for version X.XX
```

Jika instalasi Anda tidak berhasil, lihatMemecahkan masalah Amazon EMR pada penskalaan otomatis vertikal EKS.

Instal EMR Amazon pada operator penskalaan otomatis vertikal EKS

Gunakan langkah-langkah berikut untuk menginstal operator penskalaan otomatis vertikal di klaster Amazon EKS Anda:

1. Siapkan variabel lingkungan berikut yang akan Anda gunakan untuk menyelesaikan instalasi:
   - **`$REGION`**menunjuk ke Wilayah AWS untuk cluster Anda. Misalnya, `us-west-2`.
   - **`$ACCOUNT_ID`**menunjuk ke ID akun Amazon ECR untuk Wilayah Anda. Untuk informasi selengkapnya, lihat Akun ECR registri Amazon berdasarkan Wilayah.
   - **`$RELEASE`**menunjuk ke rilis EMR Amazon yang ingin Anda gunakan untuk cluster Anda. Dengan penskalaan otomatis vertikal, Anda harus menggunakan Amazon EMR rilis 6.10.0 atau lebih tinggi.
2. Selanjutnya, dapatkan token otentikasi ke registri Amazon ECR untuk operator.

   ```
   aws ecr get-login-password \
    --region region-id | docker login \
    --username AWS \
    --password-stdin $ACCOUNT_ID.dkr.ecr.region-id.amazonaws.com
   ```
3. Instal EMR Amazon pada operator penskalaan otomatis vertikal EKS dengan perintah berikut:

   ```
   ECR_URL=$ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com && \
   REPO_DEST=dynamic-sizing-k8s-operator-olm-bundle && \
   BUNDLE_IMG=emr-$RELEASE-dynamic-sizing-k8s-operator && \
   operator-sdk run bundle \
   $ECR_URL/$REPO_DEST/$BUNDLE_IMG\:latest
   ```

Ini akan membuat rilis operator penskalaan otomatis vertikal di namespace default kluster Amazon EKS Anda. Gunakan perintah ini untuk menginstal di namespace yang berbeda:

```
operator-sdk run bundle \
$ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/dynamic-sizing-k8s-operator-olm-bundle/
emr-$RELEASE-dynamic-sizing-k8s-operator:latest \
-n operator-namespace
```

> **Note**
>
> Jika namespace yang Anda tentukan tidak ada, OLM tidak akan menginstal operator. Untuk informasi selengkapnya, lihat Namespace Kubernetes tidak ditemukan.

4. Verifikasi bahwa Anda berhasil menginstal operator dengan alat baris perintah kubectl Kubernetes.

```
kubectl get csv -n operator-namespace
```

`kubectl`Perintah harus mengembalikan operator autoscaler vertikal yang baru digunakan dengan status Fase Succeeded. Jika Anda mengalami masalah dengan instalasi atau pengaturan, lihatMemecahkan masalah Amazon EMR pada penskalaan otomatis vertikal EKS.

Memulai dengan penskalaan otomatis vertikal untuk Amazon EMR di EKS

Mengirimkan pekerjaan Spark dengan autoscaling vertikal

Saat Anda mengirimkan pekerjaan melalui StartJobRunAPI, tambahkan dua konfigurasi berikut ke driver untuk pekerjaan Spark Anda untuk mengaktifkan penskalaan otomatis vertikal:

```
"spark.kubernetes.driver.label.emr-containers.amazonaws.com/dynamic.sizing":"true",
"spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/
dynamic.sizing.signature":"YOUR_JOB_SIGNATURE"
```

Pada kode di atas, baris pertama memungkinkan kemampuan autoscaling vertikal. Baris berikutnya adalah konfigurasi tanda tangan wajib yang memungkinkan Anda memilih tanda tangan untuk pekerjaan Anda.

Untuk informasi selengkapnya tentang konfigurasi ini dan nilai parameter yang dapat diterima, lihatMengkonfigurasi penskalaan otomatis vertikal untuk Amazon EMR di EKS. Secara default, pekerjaan Anda dikirimkan dalam mode Monitoring-Only Off dari penskalaan otomatis vertikal. Status pemantauan ini memungkinkan Anda menghitung dan melihat rekomendasi sumber daya tanpa melakukan penskalaan otomatis. Untuk informasi selengkapnya, lihat Mode penskalaan otomatis vertikal.

Contoh berikut menunjukkan cara menyelesaikan `start-job-run` perintah sampel dengan autoscaling vertikal:

```
aws emr-containers start-job-run \
--virtual-cluster-id $VIRTUAL_CLUSTER_ID \
--name $JOB_NAME \
--execution-role-arn $EMR_ROLE_ARN \
--release-label emr-6.10.0-latest \
--job-driver '{
  "sparkSubmitJobDriver": {
     "entryPoint": "local:///usr/lib/spark/examples/src/main/python/pi.py"
   }
 }' \
--configuration-overrides '{
    "applicationConfiguration": [{
        "classification": "spark-defaults",
        "properties": {
          "spark.kubernetes.driver.label.emr-containers.amazonaws.com/dynamic.sizing":
 "true",
          "spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/
dynamic.sizing.signature": "test-signature"
        }
    }]
  }'
```

Memverifikasi fungsionalitas penskalaan otomatis vertikal

Untuk memverifikasi bahwa penskalaan otomatis vertikal berfungsi dengan benar untuk pekerjaan yang dikirimkan, gunakan kubectl untuk mendapatkan sumber daya `verticalpodautoscaler` kustom dan melihat rekomendasi penskalaan Anda. Misalnya, permintaan perintah berikut untuk rekomendasi pada pekerjaan contoh dari Mengirimkan pekerjaan Spark dengan autoscaling vertikal bagian:

```
kubectl get verticalpodautoscalers --all-namespaces \
```

```
-l=emr-containers.amazonaws.com/dynamic.sizing.signature=test-signature
```

Output dari kueri ini harus menyerupai yang berikut:

```
NAME                                                              MODE    CPU         MEM
 PROVIDED   AGE
ds-jceyefkxnhrvdzw6djum3naf2abm6o63a6dvjkkedqtkhlrf25eq-vpa   Off    3304504865  True
         87m
```

Jika output Anda tidak terlihat serupa atau berisi kode kesalahan, lihat langkah-langkah Memecahkan masalah Amazon EMR pada penskalaan otomatis vertikal EKS untuk membantu menyelesaikan masalah.

Mengkonfigurasi penskalaan otomatis vertikal untuk Amazon EMR di EKS

Anda dapat mengonfigurasi penskalaan otomatis vertikal saat mengirimkan pekerjaan Amazon EMR Spark melalui API. StartJobRun Atur parameter konfigurasi terkait penskalaan otomatis pada pod driver Spark seperti yang ditunjukkan pada contoh di. Mengirimkan pekerjaan Spark dengan autoscaling vertikal

Amazon EMR pada operator penskalaan otomatis vertikal EKS mendengarkan pod driver yang memiliki autoscaling, kemudian mengatur integrasi dengan Kubernetes Vertical Pod Autoscaler (VPA) dengan pengaturan pada pod driver. Ini memfasilitasi pelacakan sumber daya dan penskalaan otomatis pod pelaksana Spark.

Bagian berikut menjelaskan parameter yang dapat Anda gunakan saat mengonfigurasi penskalaan otomatis vertikal untuk kluster Amazon EKS Anda.

> **Note**
>
> Konfigurasikan parameter sakelar fitur sebagai label, dan konfigurasikan parameter yang tersisa sebagai anotasi pada pod driver Spark. Parameter penskalaan otomatis milik `emr-containers.amazonaws.com/` domain dan memiliki awalan. `dynamic.sizing`

Parameter yang diperlukan

Anda harus menyertakan dua parameter berikut pada driver pekerjaan Spark saat Anda mengirimkan pekerjaan Anda:

Kunci	Deskripsi	Nilai yang diterima	Nilai default	Tipe	Parameter percikan 1
`dynamic.sizing`	Fitur toggle	`true`, `false`	tidak diatur	label	`spark.kubernetes.driver.label.emr-containers.amazonaws.com/dynamic.sizing`
`dynamic.sizing.signature`	Tanda tangan Job	tali	tidak diatur	anotasi	`spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.signature`

[1] Gunakan parameter ini sebagai `SparkSubmitParameter` atau `ConfigurationOverride` di `StartJobRun` API.

- **`dynamic.sizing`**— Anda dapat menghidupkan dan mematikan penskalaan otomatis vertikal dengan label. `dynamic.sizing` Untuk mengaktifkan penskalaan otomatis vertikal, atur `dynamic.sizing` ke `true` pada pod driver Spark. Jika Anda menghilangkan label ini atau mengaturnya ke nilai apa pun selain`true`, penskalaan otomatis vertikal tidak aktif.
- **`dynamic.sizing.signature`**— Atur tanda tangan pekerjaan dengan `dynamic.sizing.signature` anotasi pada pod driver. Penskalaan otomatis vertikal mengumpulkan data penggunaan sumber daya Anda di berbagai pekerjaan Amazon EMR Spark untuk mendapatkan rekomendasi sumber daya. Anda memberikan pengenal unik untuk mengikat pekerjaan bersama-sama.

> **Note**
>
> Jika pekerjaan Anda berulang pada interval tetap seperti harian atau mingguan, maka tanda tangan pekerjaan Anda harus tetap sama untuk setiap contoh pekerjaan baru. Ini memastikan bahwa penskalaan otomatis vertikal dapat menghitung dan mengumpulkan rekomendasi di berbagai pekerjaan.

[1] Gunakan parameter ini sebagai `SparkSubmitParameter` atau `ConfigurationOverride` di `StartJobRun` API.

Parameter opsional

Penskalaan otomatis vertikal juga mendukung parameter opsional berikut. Tetapkan mereka sebagai anotasi pada pod driver.

Kunci	Deskripsi	Nilai yang diterima	Nilai default	Tipe	Parameter percikan 1
dynamic.sizing.mode	Mode penskalaan otomatis vertikal	`Off`, `Initial`, `Auto`	`Off`	anotasi	`spark.kubernetes.driver.label.emr-containers.amazonaws.com/dynamic.sizing.mode`
dynamic.sizing.scale.memory	Mengaktifkan penskalaan memori	*`true`*, *`false`*	`true`	anotasi	`spark.kubernetes.driver.label.emr-containers.amazonaws.com/dyna`

Kunci	Deskripsi	Nilai yang diterima	Nilai default	Tipe	Parameter percikan 1
					`mic.sizing.scale.memory`
`dynamic.sizing.scale.cpu`	Aktifkan atau nonaktifkan penskalaan CPU	*`true`*, *`false`*	`false`	anotasi	`spark.kubernetes.driver.label.emr-containers.amazonaws.com/dynamic.sizing.scale.cpu`
`dynamic.sizing.scale.memory.min`	Batas minimum untuk penskalaan memori	string, kuantitas sumber daya K8s mis: `1G`	tidak diatur	anotasi	`spark.kubernetes.driver.label.emr-containers.amazonaws.com/dynamic.sizing.scale.memory.min`

Kunci	Deskripsi	Nilai yang diterima	Nilai default	Tipe	Parameter percikan 1
dynamic.sizing.scale.memory.max	Batas maksimum untuk penskalaan memori	string, kuantitas sumber daya K8s mis: 4G	tidak diatur	anotasi	spark.kubernetes.driver.label.emr-containers.amazonaws.com/dynamic.sizing.scale.memory.max
dynamic.sizing.scale.cpu.min	Batas minimum untuk penskalaan CPU	string, kuantitas sumber daya K8s mis: 1	tidak diatur	anotasi	spark.kubernetes.driver.label.emr-containers.amazonaws.com/dynamic.sizing.scale.cpu.min
dynamic.sizing.scale.cpu.max	Batas maksimum untuk penskalaan CPU	string, kuantitas sumber daya K8s mis: 2	tidak diatur	anotasi	spark.kubernetes.driver.label.emr-containers.amazonaws.com/dynamic.sizing.scale.cpu.max

Mode penskalaan otomatis vertikal

`mode`Parameter memetakan ke berbagai mode penskalaan otomatis yang didukung VPA. Gunakan `dynamic.sizing.mode` anotasi pada pod driver untuk mengatur mode. Nilai-nilai berikut didukung untuk parameter ini:

- Off — Mode dry-run di mana Anda dapat memantau rekomendasi, tetapi penskalaan otomatis tidak dilakukan. Ini adalah mode default untuk penskalaan otomatis vertikal. Dalam mode ini, sumber daya autoscaler pod vertikal terkait menghitung rekomendasi, dan Anda dapat memantau rekomendasi melalui alat seperti kubectl, Prometheus, dan Grafana.
- Awal — Dalam mode ini, VPA menskalakan otomatis sumber daya saat pekerjaan dimulai jika rekomendasi tersedia berdasarkan riwayat pekerjaan, seperti dalam kasus pekerjaan berulang.
- Otomatis — Dalam mode ini, VPA mengusir pod pelaksana Spark, dan menskalakannya secara otomatis dengan pengaturan sumber daya yang disarankan saat pod driver Spark memulai ulang. Terkadang, VPA mengusir pod pelaksana Spark yang sedang berjalan, sehingga dapat menghasilkan latensi tambahan saat mencoba ulang eksekutor yang terputus.

Penskalaan sumber daya

Saat Anda mengatur penskalaan otomatis vertikal, Anda dapat memilih apakah akan menskalakan sumber daya CPU dan memori. Atur `dynamic.sizing.scale.cpu` dan `dynamic.sizing.scale.memory` anotasi ke `true` atau`false`. Secara default, penskalaan CPU diatur ke`false`, dan penskalaan memori diatur ke. `true`

Minimum dan maksimum sumber daya (Batas)

Secara opsional, Anda juga dapat menetapkan batasan pada CPU dan sumber daya memori. Pilih nilai minimum dan maksimum untuk sumber daya ini dengan `dynamic.sizing.[memory/cpu].[min/max]` anotasi saat Anda mengaktifkan penskalaan otomatis. Secara default, sumber daya tidak memiliki batasan. Tetapkan anotasi sebagai nilai string yang mewakili kuantitas sumber daya Kubernetes. Misalnya, atur `dynamic.sizing.memory.max` `4G` untuk mewakili 4 GB.

Memantau penskalaan otomatis vertikal untuk Amazon EMR di EKS

Anda dapat menggunakan alat baris perintah kubectl Kubernetes untuk membuat daftar rekomendasi terkait penskalaan otomatis vertikal yang aktif di klaster Anda. Anda juga dapat melihat tanda tangan pekerjaan yang dilacak, dan membersihkan sumber daya yang tidak dibutuhkan yang terkait dengan tanda tangan.

Buat daftar rekomendasi penskalaan otomatis vertikal untuk klaster Anda

Gunakan kubectl untuk mendapatkan `verticalpodautoscaler` sumber daya, dan lihat status dan rekomendasi saat ini. Contoh kueri berikut mengembalikan semua sumber daya aktif di klaster Amazon EKS Anda.

```
kubectl get verticalpodautoscalers \
-o custom-columns="NAME:.metadata.name,"\
"SIGNATURE:.metadata.labels.emr-containers\.amazonaws\.com/dynamic\.sizing
\.signature,"\
"MODE:.spec.updatePolicy.updateMode,"\
"MEM:.status.recommendation.containerRecommendations[0].target.memory" \
--all-namespaces
```

Output dari kueri ini menyerupai yang berikut:

```
NAME                  SIGNATURE              MODE      MEM
ds-example-id-1-vpa   job-signature-1        Off       none
ds-example-id-2-vpa   job-signature-2        Initial   12936384283
```

Kueri dan hapus rekomendasi penskalaan otomatis vertikal untuk klaster Anda

Saat Anda menghapus resource job-run penskalaan otomatis vertikal Amazon EMR, sumber daya ini secara otomatis menghapus objek VPA terkait yang melacak dan menyimpan rekomendasi.

Contoh berikut menggunakan kubectl untuk membersihkan rekomendasi untuk pekerjaan yang diidentifikasi dengan tanda tangan:

```
kubectl delete jobrun -n emr -l=emr-containers\.amazonaws\.com/dynamic\.sizing
\.signature=integ-test
jobrun.dynamicsizing.emr.services.k8s.aws "ds-job-signature" deleted
```

Jika Anda tidak mengetahui tanda tangan pekerjaan tertentu, atau ingin membersihkan semua sumber daya di klaster, Anda dapat menggunakan `--all` atau `--all-namespaces` dalam perintah Anda alih-alih ID pekerjaan unik, seperti yang ditunjukkan pada contoh berikut:

```
kubectl delete jobruns --all --all-namespaces
jobrun.dynamicsizing.emr.services.k8s.aws "ds-example-id" deleted
```

Copot pemasangan EMR Amazon pada operator penskalaan otomatis vertikal EKS

Jika Anda ingin menghapus operator penskalaan otomatis vertikal dari kluster Amazon EKS Anda, gunakan `cleanup` perintah dengan Operator SDK CLI seperti yang ditunjukkan pada contoh berikut. Ini juga menghapus dependensi upstream yang diinstal dengan operator, seperti Vertical Pod Autoscaler.

```
operator-sdk cleanup emr-dynamic-sizing
```

Jika ada pekerjaan yang berjalan di cluster saat Anda menghapus operator, pekerjaan tersebut terus berjalan tanpa penskalaan otomatis vertikal. Jika Anda mengirimkan pekerjaan di klaster setelah menghapus operator, Amazon EMR di EKS akan mengabaikan parameter terkait penskalaan otomatis vertikal yang mungkin telah Anda tentukan selama konfigurasi.

Menjalankan beban kerja interaktif di Amazon EMR EKS

Endpoint interaktif adalah gateway yang menghubungkan Amazon EMR Studio ke EMR Amazon EKS sehingga Anda dapat menjalankan beban kerja interaktif. Anda dapat menggunakan endpoint interaktif dengan EMR Studio untuk menjalankan analisis interaktif dengan kumpulan data di penyimpanan data seperti Amazon S3 dan Amazon DynamoDB.

Kasus penggunaan

- Buat ETL skrip dengan IDE pengalaman EMR Studio. IDEMenyerap data lokal dan menyimpannya di Amazon S3 setelah transformasi untuk analisis selanjutnya.
- Gunakan buku catatan untuk menjelajahi kumpulan data dan melatih model pembelajaran mesin untuk mendeteksi anomali dalam kumpulan data.
- Buat skrip yang menghasilkan laporan harian untuk aplikasi analitik seperti dasbor bisnis.

Topik

Ikhtisar titik akhir interaktif

Endpoint interaktif menyediakan kemampuan bagi klien interaktif seperti Amazon EMR Studio untuk terhubung ke Amazon EMR di EKS cluster untuk menjalankan beban kerja interaktif. Endpoint interaktif didukung oleh Jupyter Enterprise Gateway yang menyediakan kemampuan manajemen siklus hidup kernel jarak jauh yang dibutuhkan klien interaktif. Kernel adalah proses khusus bahasa yang berinteraksi dengan klien Amazon EMR Studio berbasis Jupiter untuk menjalankan beban kerja interaktif.

Endpoint interaktif mendukung kernel berikut:

- Python 3
- PySpark di Kubernetes
- Apache Spark dengan Scala

> **Note**
>
> Amazon EMR pada EKS harga berlaku untuk endpoint interaktif dan kernel. Untuk informasi selengkapnya, lihat EMRhalaman EKS harga Amazon.

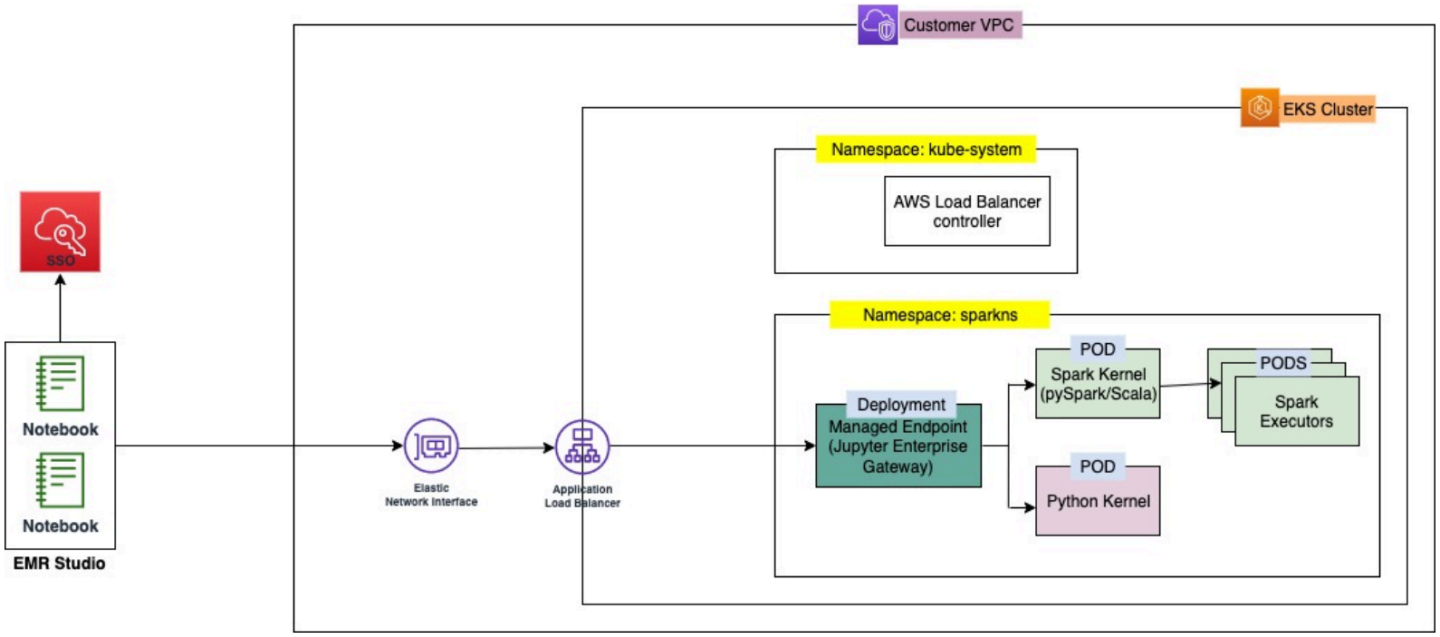
Entitas berikut diperlukan agar EMR Studio dapat terhubung dengan Amazon EMR diEKS.

- Amazon EMR di klaster EKS virtual — Cluster virtual adalah namespace Kubernetes tempat Anda mendaftarkan Amazon. EMR Amazon EMR menggunakan klaster virtual untuk menjalankan pekerjaan dan meng-host titik akhir. Anda dapat mendukung beberapa cluster virtual dengan cluster fisik yang sama. Namun, setiap cluster virtual memetakan ke satu namespace di cluster AmazonEKS. Cluster virtual tidak membuat sumber daya aktif apa pun yang berkontribusi pada tagihan Anda atau yang memerlukan manajemen siklus hidup di luar layanan.
- Amazon EMR pada endpoint EKS interaktif — Endpoint interaktif adalah HTTPS titik akhir tempat pengguna EMR Studio dapat menghubungkan ruang kerja. Anda hanya dapat mengakses HTTPS titik akhir dari EMR Studio, dan Anda membuatnya di subnet pribadi Amazon Virtual Private Cloud (AmazonVPC) untuk klaster Amazon EKS Anda.

  Kernel Python, PySpark, dan Spark Scala menggunakan izin yang ditentukan di EMR Amazon Anda EKS pada peran eksekusi pekerjaan untuk memanggil lainnya. AWS layanan Semua kernel dan pengguna yang terhubung ke endpoint interaktif menggunakan peran yang Anda tentukan saat membuat endpoint. Kami menyarankan Anda membuat endpoint terpisah untuk pengguna yang berbeda, dan bahwa pengguna memiliki peran AWS Identity and Access Management (IAM) yang berbeda.
- AWS Pengontrol Application Load Balancer — Pengontrol AWS Application Load Balancer mengelola Elastic Load Balancing untuk klaster Amazon Kubernetes. EKS Controller menyediakan Application Load Balancer (ALB) saat Anda membuat resource Kubernetes Ingress. Sebuah ALB mengekspos layanan Kubernetes, seperti endpoint interaktif, di luar EKS klaster Amazon tetapi dalam Amazon yang sama. VPC Ketika Anda membuat endpoint interaktif, sumber daya Ingress juga digunakan yang mengekspos endpoint interaktif melalui untuk klien interaktif ALB untuk

terhubung ke. Anda hanya perlu menginstal satu AWS Application Load Balancer controller untuk setiap cluster AmazonEKS.

Diagram berikut menggambarkan arsitektur endpoint interaktif di Amazon EMR pada. EKS EKSCluster Amazon terdiri dari komputasi untuk menjalankan beban kerja analitik, dan titik akhir interaktif. Pengontrol Application Load Balancer berjalan di `kube-system` namespace; beban kerja dan endpoint interaktif berjalan di namespace yang Anda tentukan saat Anda membuat cluster virtual. Saat Anda membuat endpoint interaktif, Amazon EMR on EKS control plane akan membuat penyebaran endpoint interaktif di klaster Amazon. EKS Selain itu, instance dari ingress penyeimbang beban aplikasi dibuat oleh pengontrol penyeimbang AWS beban. Penyeimbang beban aplikasi menyediakan antarmuka eksternal untuk klien seperti EMR Studio untuk terhubung ke EMR cluster Amazon dan menjalankan beban kerja interaktif.



Prasyarat untuk membuat titik akhir interaktif di Amazon EMR EKS

Bagian ini menjelaskan prasyarat untuk menyiapkan titik akhir interaktif yang dapat digunakan EMR Studio untuk terhubung ke Amazon EMR di EKS klaster dan menjalankan beban kerja interaktif.

AWS CLI

Ikuti langkah-langkah Instal AWS CLI untuk menginstal versi terbaru dari AWS Command Line Interface (AWS CLI).

Instalasi eksctl

Ikuti langkah-langkah Instal `eksctl` untuk menginstal versi terbaru eksctl. Jika Anda menggunakan Kubernetes versi 1.22 atau yang lebih baru untuk EKS klaster Amazon Anda, gunakan versi eksctl yang lebih besar dari 0.117.0.

EKSCluster Amazon

Buat EKS cluster Amazon. Daftarkan cluster sebagai cluster virtual dengan EMR Amazon aktifEKS. Berikut ini adalah persyaratan dan pertimbangan untuk cluster ini:

- Cluster harus berada di Amazon Virtual Private Cloud (VPC) yang sama dengan EMR Studio Anda.
- Cluster harus memiliki setidaknya satu subnet pribadi untuk mengaktifkan endpoint interaktif, untuk menautkan repositori berbasis Git, dan untuk meluncurkan Application Load Balancer dalam mode pribadi.
- Harus ada setidaknya satu subnet pribadi yang sama antara EMR Studio Anda dan EKS klaster Amazon yang Anda gunakan untuk mendaftarkan cluster virtual Anda. Ini memastikan bahwa endpoint interaktif Anda muncul sebagai opsi di ruang kerja Studio Anda, dan mengaktifkan konektivitas dari Studio ke Application Load Balancer.

  Ada dua metode yang dapat Anda pilih untuk menghubungkan Studio dan EKS klaster Amazon Anda:

  - Buat EKS klaster Amazon dan kaitkan dengan subnet milik EMR Studio Anda.
  - Atau, buat EMR Studio dan tentukan subnet pribadi untuk EKS klaster Amazon Anda.
- Amazon EKS dioptimalkan ARM Amazon Linux tidak AMIs didukung untuk Amazon EMR pada endpoint EKS interaktif.
- Endpoint interaktif bekerja dengan EKS klaster Amazon yang menggunakan versi Kubernetes hingga 1,29.
- Hanya grup node EKS terkelola Amazon yang didukung.

Berikan akses Cluster untuk Amazon EMR di EKS

Gunakan langkah-langkah di Grant Cluster Access for EMR Amazon EKS untuk memberi EMR Amazon EKS akses ke namespace tertentu di klaster Anda.

Aktifkan IRSA di EKS cluster Amazon

Untuk mengaktifkan IAM peran untuk Akun Layanan (IRSA) di EKS klaster Amazon, ikuti langkah-langkah di Aktifkan IAM Peran untuk Akun Layanan (IRSA).

Buat peran eksekusi IAM pekerjaan

Anda harus membuat IAM peran untuk menjalankan beban kerja di Amazon EMR pada titik akhir EKS interaktif. Kami menyebut IAM peran ini sebagai peran pelaksanaan pekerjaan dalam dokumentasi ini. IAMPeran ini akan ditetapkan ke wadah endpoint interaktif dan container eksekusi aktual yang dibuat saat Anda mengirimkan pekerjaan dengan EMR Studio. Anda memerlukan Amazon Resource Name (ARN) dari peran eksekusi pekerjaan Anda untuk EMR AmazonEKS. Ada dua langkah yang diperlukan untuk ini:

- Buat IAM peran untuk eksekusi pekerjaan.
- Perbarui kebijakan kepercayaan dari peran eksekusi pekerjaan.

Berikan pengguna akses ke Amazon EMR di EKS

IAMEntitas (pengguna atau peran) yang membuat permintaan untuk membuat titik akhir interaktif juga harus memiliki Amazon EC2 dan `emr-containers` izin berikut. Ikuti langkah-langkah yang dijelaskan Berikan pengguna akses ke Amazon EMR di EKS untuk memberikan izin ini yang memungkinkan EMR Amazon EKS untuk membuat, mengelola, dan menghapus grup keamanan yang membatasi lalu lintas masuk ke penyeimbang beban titik akhir interaktif Anda.

`emr-containers`Izin berikut memungkinkan pengguna untuk melakukan operasi endpoint interaktif dasar:

```
"ec2:CreateSecurityGroup",
"ec2:DeleteSecurityGroup",
"ec2:AuthorizeSecurityGroupEgress",
"ec2:AuthorizeSecurityGroupIngress",
"ec2:RevokeSecurityGroupEgress",
"ec2:RevokeSecurityGroupIngress"

"emr-containers:CreateManagedEndpoint",
"emr-containers:ListManagedEndpoints",
"emr-containers:DescribeManagedEndpoint",
"emr-containers:DeleteManagedEndpoint"
```

Daftarkan EKS cluster Amazon dengan Amazon EMR

Siapkan cluster virtual dan petakan ke namespace di EKS cluster Amazon tempat Anda ingin menjalankan pekerjaan. Untuk cluster AWS Fargate-only, gunakan namespace yang sama untuk Amazon EMR di EKS klaster virtual dan profil Fargate.

Untuk informasi tentang cara menyiapkan Amazon EMR di klaster EKS virtual, lihatDaftarkan EKS cluster Amazon dengan Amazon EMR.

Menerapkan AWS Load Balancer Controller ke klaster Amazon EKS

AWS Application Load Balancer diperlukan untuk klaster Amazon EKS Anda. Anda hanya perlu menyiapkan satu pengontrol Application Load Balancer per cluster AmazonEKS. Untuk informasi tentang cara menyiapkan pengontrol AWS Application Load Balancer, lihat Menginstal add-on Load AWS Balancer Controller di Panduan Pengguna Amazon. EKS

Membuat endpoint interaktif untuk klaster virtual Anda

Halaman ini menjelaskan cara membuat endpoint interaktif menggunakan AWS Command Line Interface (AWS CLI).

Buat endpoint interaktif dengan perintah **`create-managed-endpoint`**

Tentukan parameter dalam `create-managed-endpoint` perintah sebagai berikut. EMRAmazon EKS mendukung pembuatan titik akhir interaktif dengan Amazon EMR merilis 6.7.0 dan yang lebih tinggi.

```
aws emr-containers create-managed-endpoint \
--type JUPYTER_ENTERPRISE_GATEWAY \
--virtual-cluster-id 1234567890abcdef0xxxxxxx \
--name example-endpoint-name \
--execution-role-arn arn:aws:iam::444455556666:role/JobExecutionRole \
--release-label emr-6.9.0-latest \
--configuration-overrides '{
    "applicationConfiguration": [{
        "classification": "spark-defaults",
        "properties": {
            "spark.driver.memory": "2G"
        }
    }],
    "monitoringConfiguration": {
```

```
        "cloudWatchMonitoringConfiguration": {
            "logGroupName": "log_group_name",
            "logStreamNamePrefix": "log_stream_prefix"
        },
        "persistentAppUI": "ENABLED",
        "s3MonitoringConfiguration": {
            "logUri": "s3://my_s3_log_location"
        }
    }
}'
```

Untuk informasi selengkapnya, lihat Parameter untuk membuat endpoint interaktif.

Buat endpoint interaktif dengan parameter tertentu dalam file JSON

1. Buat `create-managed-endpoint-request.json` file dan tentukan parameter yang diperlukan untuk titik akhir Anda, seperti yang ditunjukkan pada JSON file berikut:

```
{
    "name": "MY_TEST_ENDPOINT",
    "virtualClusterId": "MY_CLUSTER_ID",
    "type": "JUPYTER_ENTERPRISE_GATEWAY",
    "releaseLabel": "emr-6.9.0-latest",
    "executionRoleArn": "arn:aws:iam::444455556666:role/JobExecutionRole",
    "configurationOverrides":
    {
        "applicationConfiguration":
        [
            {
                "classification": "spark-defaults",
                "properties":
                {
                    "spark.driver.memory": "8G"
                }
            }
        ],
        "monitoringConfiguration":
        {
            "persistentAppUI": "ENABLED",
            "cloudWatchMonitoringConfiguration":
            {
                "logGroupName": "my_log_group",
                "logStreamNamePrefix": "log_stream_prefix"
```

```
            },
            "s3MonitoringConfiguration":
            {
                "logUri": "s3://my_s3_log_location"
            }
        }
    }
}
```

2. Gunakan `create-managed-endpoint` perintah dengan path ke `create-managed-endpoint-request.json` file yang disimpan secara lokal atau di Amazon S3.

```
aws emr-containers create-managed-endpoint \
--cli-input-json  file://./create-managed-endpoint-request.json --region AWS-Region
```

Output dari membuat endpoint interaktif

Anda akan melihat output berikut di terminal. Outputnya mencakup nama dan pengenal titik akhir interaktif baru Anda:

```
{
    "id": "1234567890abcdef0",
    "name": "example-endpoint-name",
    "arn": "arn:aws:emr-containers:us-west-2:111122223333:/
virtualclusters/444455556666/endpoints/444455556666",
    "virtualClusterId": "111122223333xxxxxxxx"
}
```

Running `aws emr-containers create-managed-endpoint` membuat sertifikat yang ditandatangani sendiri yang memungkinkan HTTPS komunikasi antara EMR Studio dan server endpoint interaktif.

Jika Anda menjalankan `create-managed-endpoint` dan belum menyelesaikan prasyarat, EMR Amazon mengembalikan pesan kesalahan dengan tindakan yang harus Anda ambil untuk melanjutkan.

Parameter untuk membuat endpoint interaktif

Topik

- Parameter yang diperlukan untuk titik akhir interaktif

Parameter yang diperlukan untuk titik akhir interaktif

Anda harus menentukan parameter berikut saat membuat endpoint interaktif:

`--type`

Gunakan `JUPYTER_ENTERPRISE_GATEWAY`. Ini adalah satu-satunya jenis yang didukung.

`--virtual-cluster-id`

Pengidentifikasi cluster virtual tempat Anda terdaftar EMR di EKS Amazon.

`--name`

Nama deskriptif untuk endpoint interaktif yang membantu pengguna EMR Studio memilihnya dari daftar dropdown.

`--execution-role-arn`

Nama Sumber Daya Amazon (ARN) dari peran eksekusi IAM pekerjaan Anda untuk EMR Amazon EKS yang dibuat sebagai bagian dari prasyarat.

`--release-label`

Label rilis EMR rilis Amazon yang akan digunakan untuk titik akhir. Misalnya, `emr-6.9.0-latest`. EMRAmazon EKS mendukung endpoint interaktif dengan Amazon EMR merilis 6.7.0 dan yang lebih tinggi.

Parameter opsional untuk titik akhir interaktif

Secara opsional, Anda juga dapat menentukan parameter berikut saat membuat endpoint interaktif:

`--configuration-overrides`

Untuk mengganti konfigurasi default untuk aplikasi, berikan objek coonfiguration. 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 satu JSON objek. Klasifikasi konfigurasi yang tersedia bervariasi menurut EMR Amazon saat EKS rilis. Untuk daftar klasifikasi konfigurasi yang tersedia untuk setiap rilis EMR AmazonEKS, lihatAmazon EMR pada EKS rilis. Selain klasifikasi konfigurasi yang terdaftar untuk setiap rilis, titik akhir interaktif membawa klasifikasi tambahan. `jeg-config` Untuk informasi selengkapnya, lihat Opsi konfigurasi Jupyter Enterprise Gateway (JEG).

Mengkonfigurasi pengaturan untuk titik akhir interaktif

Lowongan kerja Monitoring Spark

Agar Anda dapat memantau dan memecahkan masalah kegagalan, konfigurasikan titik akhir interaktif Anda sehingga pekerjaan yang dimulai dengan titik akhir dapat mengirim informasi log ke Amazon S3, Amazon Log, atau keduanya. CloudWatch Bagian berikut menjelaskan cara mengirim log aplikasi Spark ke Amazon S3 untuk pekerjaan Spark yang Anda luncurkan dengan EMR Amazon EKS pada titik akhir interaktif.

Konfigurasikan IAM kebijakan untuk log Amazon S3

Sebelum kernel Anda dapat mengirim data log ke Amazon S3, kebijakan izin untuk peran eksekusi pekerjaan harus menyertakan izin berikut. Ganti *DOC-EXAMPLE-BUCKET-LOGGING* dengan nama ember logging Anda.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET-LOGGING",
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET-LOGGING/*",
            ]
        }
    ]
}
```

> **Note**
>
> Amazon EMR on juga EKS dapat membuat bucket S3. Jika bucket S3 tidak tersedia, sertakan `s3:CreateBucket` izin tersebut dalam IAM kebijakan.

Setelah Anda memberikan izin pada peran eksekusi yang diperlukan untuk mengirim log ke bucket S3, data log Anda akan dikirim ke lokasi Amazon S3 berikut. Ini terjadi ketika `s3MonitoringConfiguration` diteruskan di `monitoringConfiguration` bagian `create-managed-endpoint` permintaan.

- Log driver — `logUri/virtual-cluster-id/endpoints/endpoint-id/containers/spark-application-id/spark-application-id-driver/(stderr.gz/stdout.gz)`
- Log pelaksana — `logUri/virtual-cluster-id/endpoints/endpoint-id/containers/spark-application-id/executor-pod-name-exec-<Number>/(stderr.gz/stdout.gz)`

> **Note**
>
> Amazon EMR on EKS tidak mengunggah log titik akhir ke bucket S3 Anda.

Menentukan template pod kustom dengan endpoint interaktif

Anda dapat membuat endpoint interaktif di mana Anda menentukan template pod kustom untuk driver dan pelaksana. Template Pod adalah spesifikasi yang menentukan cara menjalankan setiap pod. Anda dapat menggunakan file template pod untuk menentukan konfigurasi driver atau pod pelaksana yang tidak didukung oleh konfigurasi Spark. Template Pod saat ini didukung di Amazon EMR rilis 6.3.0 dan yang lebih baru.

Untuk informasi selengkapnya tentang templat pod, lihat Menggunakan templat pod di Amazon EMR on EKS Development Guide.

Contoh berikut menunjukkan cara membuat endpoint interaktif dengan template pod:

```
aws emr-containers create-managed-endpoint \
    --type JUPYTER_ENTERPRISE_GATEWAY \
    --virtual-cluster-id virtual-cluster-id \
```

```
    --name example-endpoint-name \
    --execution-role-arn arn:aws:iam::aws-account-id:role/EKSClusterRole \
    --release-label emr-6.9.0-latest \
    --configuration-overrides '{
        "applicationConfiguration": [
        {
            "classification": "spark-defaults",
            "properties": {
                "spark.kubernetes.driver.podTemplateFile": "path/to/driver/
template.yaml",
                "spark.kubernetes.executor.podTemplateFile": "path/to/executor/
template.yaml"
            }
        }]
    }'
```

Menerapkan JEG pod ke grup node

JEG(Jupyter Enterprise Gateway) penempatan pod adalah fitur yang memungkinkan Anda untuk menerapkan endpoint interaktif pada grup node tertentu. Dengan fitur ini, Anda dapat mengonfigurasi pengaturan seperti `instance type` untuk titik akhir interaktif.

Mengaitkan JEG pod ke grup node terkelola

Properti konfigurasi berikut memungkinkan Anda menentukan nama grup node terkelola di EKS klaster Amazon tempat JEG pod akan di-deploy.

```
//payload
--configuration-overrides '{
      "applicationConfiguration": [
            {
                "classification": "endpoint-configuration",
                "properties": {
                    "managed-nodegroup-name": NodeGroupName
                }
            }
        ]
    }'
```

Sebuah grup node harus memiliki label Kubernetes yang `for-use-with-emr-containers-managed-endpoint-ng=`*NodeGroupName* melekat pada semua node yang merupakan bagian dari

grup node. Untuk membuat daftar semua node dari grup node yang memiliki tag ini, gunakan perintah berikut:

```
kubectl get nodes --show-labels | grep for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName
```

Jika output dari perintah di atas tidak mengembalikan node yang merupakan bagian dari grup node terkelola Anda, maka tidak ada node dalam grup node yang memiliki label `for-use-with-emr-containers-managed-endpoint-ng=`*`NodeGroupName`* Kubernetes terpasang. Dalam hal ini, ikuti langkah-langkah di bawah ini untuk melampirkan label tersebut ke node di grup node Anda.

1. Gunakan perintah berikut untuk menambahkan label `for-use-with-emr-containers-managed-endpoint-ng=`*`NodeGroupName`* Kubernetes ke semua node dalam grup node terkelola: *`NodeGroupName`*

   ```
   kubectl label nodes --selector eks:nodegroup-name=NodeGroupName for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName
   ```

2. Verifikasi bahwa node diberi label dengan benar menggunakan perintah berikut:

   ```
   kubectl get nodes --show-labels | grep for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName
   ```

Grup node terkelola harus dikaitkan dengan grup keamanan EKS klaster Amazon, yang biasanya terjadi jika Anda membuat cluster dan grup node terkelola menggunakan`eksctl`. Anda dapat memverifikasi ini di AWS konsol menggunakan langkah-langkah berikut.

1. Buka cluster Anda di EKS konsol Amazon.
2. Buka tab jaringan klaster Anda dan catat grup keamanan cluster.
3. Buka tab komputasi klaster Anda dan klik pada nama grup node terkelola.
4. Di bawah tab Detail grup node terkelola, verifikasi bahwa grup keamanan klaster yang Anda catat sebelumnya terdaftar di bawah Grup keamanan.

Jika grup node terkelola tidak dilampirkan ke grup keamanan EKS klaster Amazon, Anda harus melampirkan `for-use-with-emr-containers-managed-endpoint-sg=`*`ClusterName`*`/`*`NodeGroupName`* tag ke grup keamanan grup node. Gunakan langkah-langkah di bawah ini untuk melampirkan tag ini.

1. Buka EC2 konsol Amazon dan klik grup keamanan di panel navigasi kiri.
2. Pilih grup keamanan grup node terkelola Anda dengan mengklik kotak centang.
3. Di bawah tab Tag, tambahkan tag `for-use-with-emr-containers-managed-endpoint-sg=`*`ClusterName`*`/`*`NodeGroupName`* menggunakan tombol Kelola tag.

Mengaitkan JEG pod ke grup node yang dikelola sendiri

Properti konfigurasi berikut memungkinkan Anda menentukan nama grup node yang dikelola sendiri atau tidak dikelola di EKS klaster Amazon tempat JEG pod akan di-deploy.

```
//payload
--configuration-overrides '{
      "applicationConfiguration": [
            {
                "classification": "endpoint-configuration",
                "properties": {
                    "self-managed-nodegroup-name": NodeGroupName
                }
            }
        ]
    }'
```

Grup node harus memiliki label `for-use-with-emr-containers-managed-endpoint-ng=`*`NodeGroupName`* Kubernetes yang melekat pada semua node yang merupakan bagian dari grup node. Untuk membuat daftar semua node dari grup node yang memiliki tag ini, gunakan perintah berikut:

```
kubectl get nodes --show-labels | grep for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName
```

Jika output dari perintah di atas tidak mengembalikan node yang merupakan bagian dari grup node yang dikelola sendiri, maka tidak ada node di grup node yang memiliki label `for-use-with-emr-containers-managed-endpoint-ng=`*`NodeGroupName`* Kubernetes yang terpasang. Dalam hal ini, ikuti langkah-langkah di bawah ini untuk melampirkan label tersebut ke node di grup node Anda.

1. Jika Anda membuat grup node yang dikelola sendiri menggunakan`eksctl`, maka gunakan perintah berikut untuk menambahkan label `for-use-with-emr-containers-managed-endpoint-ng=`*`NodeGroupName`* Kubernetes ke semua node dalam grup node yang dikelola sendiri sekaligus. *`NodeGroupName`*

```
kubectl label nodes --selector alpha.eksctl.io/nodegroup-name=NodeGroupName for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName
```

Jika Anda tidak menggunakannya `eksctl` untuk membuat grup node yang dikelola sendiri, maka Anda perlu mengganti pemilih pada perintah di atas ke label Kubernetes yang berbeda yang dilampirkan ke semua node dari grup node.

2. Gunakan perintah berikut untuk memverifikasi bahwa node diberi label dengan benar:

```
kubectl get nodes --show-labels | grep for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName
```

Grup keamanan untuk grup node yang dikelola sendiri harus memiliki `for-use-with-emr-containers-managed-endpoint-sg=`*`ClusterName`*`/`*`NodeGroupName` tag yang dilampirkan. Gunakan langkah-langkah berikut untuk melampirkan tag ke grup keamanan dari AWS Management Console.

1. Arahkan ke EC2 konsol Amazon. Pilih Grup keamanan di panel navigasi kiri.
2. Pilih kotak centang di samping grup keamanan untuk grup node yang dikelola sendiri.
3. Di bawah tab Tag, gunakan tombol Kelola tag untuk menambahkan tag`for-use-with-emr-containers-managed-endpoint-sg=`*`ClusterName`*`/`*`NodeGroupName`*. Ganti *`ClusterName`* dan *`NodeGroupName`* dengan nilai yang sesuai.

Mengaitkan JEG pod ke grup node terkelola dengan instance On-Demand

Anda juga dapat menentukan label tambahan, yang dikenal sebagai pemilih label Kubernetes, untuk menentukan batasan atau batasan tambahan untuk menjalankan titik akhir interaktif pada node atau grup node tertentu. Contoh berikut menunjukkan cara menggunakan EC2 instans Amazon On-Demand untuk sebuah JEG pod.

```
--configuration-overrides '{
      "applicationConfiguration": [
            {
                "classification": "endpoint-configuration",
                "properties": {
                    "managed-nodegroup-name": NodeGroupName,
                    "node-labels": "eks.amazonaws.com/capacityType:ON_DEMAND"
```

```
                }
            }
        ]
    }'
```

> **Note**
>
> Anda hanya dapat menggunakan `node-labels` properti dengan `self-managed-nodegroup-name` properti `managed-nodegroup-name` atau properti.

Opsi konfigurasi Jupyter Enterprise Gateway (JEG)

EMRAmazon EKS menggunakan Jupyter Enterprise Gateway (JEG) untuk mengaktifkan titik akhir interaktif. Anda dapat menyetel nilai berikut untuk JEG konfigurasi allow-listed saat Anda membuat endpoint.

- **`RemoteMappingKernelManager.cull_idle_timeout`**— Timeout dalam hitungan detik (integer), setelah itu kernel dianggap idle dan siap untuk dimusnahkan. Nilai `0` atau lebih rendah menonaktifkan pemusnahan. Batas waktu yang singkat dapat mengakibatkan kernel dimusnahkan untuk pengguna dengan koneksi jaringan yang buruk.
- **`RemoteMappingKernelManager.cull_interval`**— Interval dalam detik (integer) untuk memeriksa kernel idle yang melebihi nilai batas waktu pemusnahan.

Memodifikasi PySpark parameter sesi

Dimulai dengan Amazon EMR pada EKS rilis 6.9.0, di Amazon EMR Studio Anda dapat menyesuaikan konfigurasi Spark yang terkait dengan PySpark sesi dengan menjalankan perintah `%%configure` ajaib di sel notebook. EMR

Contoh berikut menunjukkan payload sampel yang dapat Anda gunakan untuk memodifikasi memori, core, dan properti lainnya untuk driver dan eksekutor Spark. Untuk `conf` pengaturan, Anda dapat mengonfigurasi konfigurasi Spark apa pun yang disebutkan dalam dokumentasi konfigurasi Apache Spark.

```
%%configure -f
{
  "driverMemory": "16G",
```

```
  "driverCores" 4,
  "executorMemory" : "32G"
  "executorCores": 2,
  "conf": {
     "spark.dynamicAllocation.maxExecutors" : 10,
     "spark.dynamicAllocation.minExecutors": 1
  }
}
```

Contoh berikut menunjukkan contoh payload yang dapat Anda gunakan untuk menambahkan file,pyFiles, dan jar dependensi ke runtime Spark.

```
%%configure -f
{
  "files": "s3://test-bucket-emr-eks/sample_file.txt",
  "pyFiles": : "path-to-python-files",
  "jars" : "path-to-jars
}
```

Gambar kernel kustom dengan endpoint interaktif

Untuk memastikan bahwa Anda memiliki dependensi yang benar untuk aplikasi saat menjalankan beban kerja interaktif dari Amazon EMR Studio, Anda dapat menyesuaikan gambar Docker untuk titik akhir interaktif dan menjalankan image kernel dasar yang disesuaikan. Untuk membuat endpoint interaktif dan menghubungkannya dengan image Docker kustom, lakukan langkah-langkah berikut.

> **Note**
>
> Anda hanya dapat mengganti gambar dasar. Anda tidak dapat menambahkan jenis gambar kernel baru.

1. Buat dan publikasikan gambar Docker yang disesuaikan. Gambar dasar berisi runtime Spark dan kernel notebook yang berjalan dengannya. Untuk membuat gambar, Anda dapat mengikuti langkah 1 hingga 4 inCara menyesuaikan gambar Docker. Pada langkah 1, gambar dasar URI dalam file Docker Anda harus digunakan `notebook-spark` sebagai pengganti. `spark`

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

Untuk informasi selengkapnya tentang cara memilih Wilayah AWS dan menampung tag gambar, lihatCara memilih gambar dasar URI.

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

   a. Buat JSON file `custom-image-managed-endpoint.json` dengan konten berikut. Contoh ini menggunakan EMR rilis Amazon 6.9.0.

      Example

```
{
    "name": "endpoint-name",
    "virtualClusterId": "virtual-cluster-id",
    "type": "JUPYTER_ENTERPRISE_GATEWAY",
    "releaseLabel": "emr-6.9.0-latest",
    "executionRoleArn": "execution-role-arn",
    "configurationOverrides": {
        "applicationConfiguration": [
            {
                "classification": "jupyter-kernel-overrides",
                "configurations": [
                    {
                        "classification": "python3",
                        "properties": {
                            "container-image": "123456789012.dkr.ecr.us-
west-2.amazonaws.com/custom-notebook-python:latest"
                        }
                    },
                    {
                        "classification": "spark-python-kubernetes",
                        "properties": {
                            "container-image": "123456789012.dkr.ecr.us-
west-2.amazonaws.com/custom-notebook-spark:latest"
                        }
                    }
                ]
            }
        ]
    }
}
```

b. Buat endpoint interaktif dengan konfigurasi yang ditentukan dalam JSON file seperti yang ditunjukkan pada contoh berikut. Untuk informasi selengkapnya, lihat Buat endpoint interaktif dengan perintah `create-managed-endpoint`.

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

3. Connect ke endpoint interaktif melalui EMR Studio. Untuk informasi selengkapnya dan langkah-langkah yang harus diselesaikan, lihat Menghubungkan dari Studio di Amazon EMR di EKS bagian dokumen AWS Workshop Studio.

Memantau titik akhir interaktif

Dengan Amazon EMR pada EKS versi 6.10 dan yang lebih baru, titik akhir interaktif memancarkan metrik CloudWatch Amazon untuk memantau dan memecahkan masalah operasi siklus hidup kernel. Metrik dipicu oleh klien interaktif, seperti EMR Studio atau notebook Jupyter yang dihosting sendiri. Setiap operasi yang didukung oleh endpoint interaktif memiliki metrik yang terkait dengannya. Operasi dimodelkan sebagai dimensi untuk setiap metrik, seperti yang ditunjukkan pada tabel di bawah ini. Metrik yang dipancarkan oleh titik akhir interaktif terlihat di bawah namespace khusus, di akun Anda. EMRContainers

Metrik	Deskripsi	Unit
RequestCount	Jumlah kumulatif permintaan operasi yang diproses oleh endpoint interaktif.	Hitung
RequestLatency	Waktu dari ketika permintaan tiba di titik akhir interaktif dan respons dikirim oleh titik akhir interaktif.	Milidetik
4 XXError	Dipancarkan ketika permintaan untuk operasi menghasilkan kesalahan 4xx selama pemrosesan.	Hitung

Metrik	Deskripsi	Unit
5 XXError	Dipancarkan saat permintaan operasi menghasilkan kesalahan sisi server 5Xxx.	Hitung
KernelLaunchSuccess	Hanya berlaku untuk CreateKernel operasi. Ini menunjukkan jumlah kumulatif peluncuran kernel yang berhasil hingga dan termasuk permintaan ini.	Hitung
KernelLaunchFailure	Hanya berlaku untuk CreateKernel operasi. Ini menunjukkan jumlah kumulatif kegagalan peluncuran kernel hingga dan termasuk permintaan ini.	Hitung

Setiap metrik endpoint interaktif memiliki dimensi berikut yang melekat padanya:

- **`ManagedEndpointId`**— Pengenal untuk titik akhir interaktif
- **`OperationName`**— Operasi yang dipicu oleh klien interaktif

Nilai yang mungkin untuk **`OperationName`**dimensi ditunjukkan pada tabel berikut:

`operationName`	Deskripsi operasi
`CreateKernel`	Minta endpoint interaktif memulai kernel.
`ListKernels`	Minta agar titik akhir interaktif mencantumkan kernel yang sebelumnya telah dimulai menggunakan token sesi yang sama.

operationName	Deskripsi operasi
`GetKernel`	Minta agar endpoint interaktif mendapatkan detail tentang kernel tertentu yang telah dimulai sebelumnya.
`ConnectKernel`	Minta endpoint interaktif membangun konektivi tas antara klien notebook dan kernel.
`ConfigureKernel`	Publikasikan `%%configure magic request` pada kernel pyspark.
`ListKernelSpecs`	Minta agar titik akhir interaktif mencantumkan spesifikasi kernel yang tersedia.
`GetKernelSpec`	Minta agar endpoint interaktif mendapatk an spesifikasi kernel dari kernel yang telah diluncurkan sebelumnya.
`GetKernelSpecResource`	Minta agar endpoint interaktif mendapatkan sumber daya spesifik yang terkait dengan spesifikasi kernel yang telah diluncurkan sebelumnya.

Contoh

Untuk mengakses jumlah kernel yang diluncurkan untuk titik akhir interaktif pada hari tertentu:

1. Pilih namespace kustom: `EMRContainers`
2. Pilih Anda`ManagedEndpointId`, `OperationName - CreateKernel`
3. `RequestCount`metrik dengan statistik `SUM` dan periode `1 day` akan memberikan semua permintaan peluncuran kernel yang dibuat dalam 24 jam terakhir.
4. KernelLaunchSuccess metrik dengan statistik `SUM` dan periode `1 day` akan memberikan semua permintaan peluncuran kernel yang berhasil dibuat dalam 24 jam terakhir.

Untuk mengakses jumlah kegagalan kernel untuk endpoint interaktif pada hari tertentu:

1. Pilih namespace kustom: EMRContainers
2. Pilih Anda`ManagedEndpointId`, `OperationName - CreateKernel`
3. `KernelLaunchFailure`metrik dengan statistik `SUM` dan periode `1 day` akan memberikan semua permintaan peluncuran kernel gagal yang dibuat dalam 24 jam terakhir. Anda juga dapat memilih `5XXError` metrik `4XXError` dan untuk mengetahui jenis kegagalan peluncuran kernel yang terjadi.

Menggunakan notebook Jupyter yang dihosting sendiri

Anda dapat meng-host dan mengelola Jupyter atau JupyterLab notebook di EC2 instans Amazon atau di EKS klaster Amazon Anda sendiri sebagai notebook Jupyter yang dihosting sendiri. Anda kemudian dapat menjalankan beban kerja interaktif dengan notebook Jupyter yang dihosting sendiri. Bagian berikut berjalan melalui proses untuk menyiapkan dan menerapkan notebook Jupyter yang dihosting sendiri di cluster Amazon. EKS

Membuat notebook Jupyter yang dihosting sendiri di sebuah cluster EKS

- Membuat grup keamanan
- Buat Amazon EMR di endpoint EKS interaktif
- Ambil server gateway dari titik URL akhir interaktif Anda
- Ambil token autentikasi untuk terhubung ke titik akhir interaktif
- Contoh: Menyebarkan buku catatan JupyterLab
- Menghapus buku catatan Jupyter yang dihosting sendiri

Membuat grup keamanan

Sebelum Anda dapat membuat endpoint interaktif dan menjalankan Jupyter atau JupyterLab notebook yang dihosting sendiri, Anda harus membuat grup keamanan untuk mengontrol lalu lintas antara buku catatan Anda dan titik akhir interaktif. Untuk menggunakan EC2 konsol Amazon atau Amazon EC2 SDK untuk membuat grup keamanan, lihat langkah-langkah dalam Membuat grup keamanan di Panduan EC2 Pengguna Amazon. Anda harus membuat grup keamanan di VPC tempat Anda ingin menyebarkan server notebook Anda.

Untuk mengikuti contoh dalam panduan ini, gunakan yang VPC sama dengan EKS cluster Amazon Anda. Jika Anda ingin meng-host notebook Anda di VPC yang berbeda dari EKS cluster VPC untuk Amazon Anda, Anda mungkin perlu membuat koneksi peering di antara keduanyaVPCs. Untuk langkah-langkah untuk membuat koneksi peering di antara keduanyaVPCs, lihat Membuat koneksi VPC peering di Panduan VPC Memulai Amazon.

Anda memerlukan ID untuk grup keamanan untuk membuat Amazon EMR pada titik akhir EKS interaktif di langkah berikutnya.

Buat Amazon EMR di endpoint EKS interaktif

Setelah Anda membuat grup keamanan untuk buku catatan Anda, gunakan langkah-langkah yang disediakan Membuat endpoint interaktif untuk klaster virtual Anda untuk membuat titik akhir interaktif. Anda harus memberikan ID grup keamanan yang Anda buat untuk buku catatan AndaMembuat grup keamanan.

Masukkan ID keamanan di tempat *your-notebook-security-group-id* dalam pengaturan penggantian konfigurasi berikut:

```
--configuration-overrides '{
    "applicationConfiguration": [
        {
            "classification": "endpoint-configuration",
            "properties": {
                "notebook-security-group-id": "your-notebook-security-group-id"
            }
        }
    ],
    "monitoringConfiguration": {
    ...'
```

Ambil server gateway dari titik URL akhir interaktif Anda

Setelah Anda membuat endpoint interaktif, ambil server gateway URL dengan `describe-managed-endpoint` perintah di. AWS CLI Anda memerlukan ini URL untuk menghubungkan notebook Anda ke titik akhir. Server gateway URL adalah titik akhir pribadi.

```
aws emr-containers describe-managed-endpoint \
--region region \
--virtual-cluster-id virtualClusterId \
```

```
--id endpointId
```

Awalnya, titik akhir Anda ada di CREATINGnegara bagian. Setelah beberapa menit, ia beralih ke ACTIVEnegara. Ketika titik akhir ACTIVE, itu siap digunakan.

Perhatikan `serverUrl` atribut yang dikembalikan `aws emr-containers describe-managed-endpoint` perintah dari endpoint aktif. Anda memerlukan ini URL untuk menghubungkan notebook Anda ke titik akhir saat Anda menerapkan Jupyter atau notebook yang dihosting sendiri. JupyterLab

Ambil token autentikasi untuk terhubung ke titik akhir interaktif

Untuk terhubung ke titik akhir interaktif dari Jupyter atau JupyterLab notebook, Anda harus membuat token sesi dengan. `GetManagedEndpointSessionCredentials` API Token bertindak sebagai bukti otentikasi untuk terhubung ke server endpoint interaktif.

Perintah berikut dijelaskan secara lebih rinci dengan contoh output di bawah ini.

```
aws emr-containers get-managed-endpoint-session-credentials \
--endpoint-identifier endpointArn \
--virtual-cluster-identifier virtualClusterArn \
--execution-role-arn executionRoleArn \
--credential-type "TOKEN" \
--duration-in-seconds durationInSeconds \
--region region
```

endpointArn

Titik ARN akhir Anda. Anda dapat menemukan hasil `describe-managed-endpoint` panggilan. ARN

virtualClusterArn

ARNDari cluster virtual.

executionRoleArn

ARNPeran eksekusi.

durationInSeconds

Durasi dalam detik dimana token valid. Durasi default adalah 15 menit (`900`), dan maksimum adalah 12 jam (`43200`).

region

Wilayah yang sama dengan titik akhir Anda.

Output Anda harus menyerupai contoh berikut. Catat *session-token* nilai yang akan Anda gunakan saat menerapkan Jupyter atau notebook yang dihosting sendiri. JupyterLab

```
{
    "id": "credentialsId",
    "credentials": {
        "token": "session-token"
    },
    "expiresAt": "2022-07-05T17:49:38Z"
}
```

Contoh: Menyebarkan buku catatan JupyterLab

Setelah Anda menyelesaikan langkah-langkah di atas, Anda dapat mencoba prosedur contoh ini untuk menyebarkan JupyterLab notebook ke dalam EKS klaster Amazon dengan endpoint interaktif Anda.

1. Buat namespace untuk menjalankan server notebook.
2. Buat file secara lokal,`notebook.yaml`, dengan konten berikut. Isi file dijelaskan di bawah ini.

```
apiVersion: v1
kind: Pod
metadata:
  name: jupyter-notebook
  namespace: namespace
spec:
  containers:
  - name: minimal-notebook
    image: jupyter/all-spark-notebook:lab-3.1.4 # open source image
    ports:
    - containerPort: 8888
    command: ["start-notebook.sh"]
    args: ["--LabApp.token=''"]
    env:
    - name: JUPYTER_ENABLE_LAB
      value: "yes"
    - name: KERNEL_LAUNCH_TIMEOUT
```

```
      value: "400"
    - name: JUPYTER_GATEWAY_URL
      value: "serverUrl"
    - name: JUPYTER_GATEWAY_VALIDATE_CERT
      value: "false"
    - name: JUPYTER_GATEWAY_AUTH_TOKEN
      value: "session-token"
```

Jika Anda menerapkan notebook Jupyter ke cluster khusus Fargate, beri label pada pod Jupyter dengan label seperti yang ditunjukkan pada contoh berikut: `role`

```
...
metadata:
  name: jupyter-notebook
  namespace: default
  labels:
    role: example-role-name-label
spec:
            ...
```

namespace

Namespace Kubernetes yang digunakan notebook.

serverUrl

`serverUrl`Atribut yang dikembalikan `describe-managed-endpoint` perintahAmbil server gateway dari titik URL akhir interaktif Anda .

session-token

`session-token`Atribut yang dikembalikan `get-managed-endpoint-session-credentials` perintahAmbil token autentikasi untuk terhubung ke titik akhir interaktif.

KERNEL_LAUNCH_TIMEOUT

Jumlah waktu dalam detik titik akhir interaktif menunggu kernel muncul. RUNNING Pastikan waktu yang cukup untuk menyelesaikan peluncuran kernel dengan mengatur batas waktu peluncuran kernel ke nilai yang sesuai (maksimum 400 detik).

KERNEL_EXTRA_SPARK_OPTS

Secara opsional, Anda dapat meneruskan konfigurasi Spark tambahan untuk kernel Spark. Tetapkan variabel lingkungan ini dengan nilai-nilai sebagai properti konfigurasi Spark seperti yang ditunjukkan pada contoh berikut:

```
- name: KERNEL_EXTRA_SPARK_OPTS
  value: "--conf spark.driver.cores=2
          --conf spark.driver.memory=2G
          --conf spark.executor.instances=2
          --conf spark.executor.cores=2
          --conf spark.executor.memory=2G
          --conf spark.dynamicAllocation.enabled=true
          --conf spark.dynamicAllocation.shuffleTracking.enabled=true
          --conf spark.dynamicAllocation.minExecutors=1
          --conf spark.dynamicAllocation.maxExecutors=5
          --conf spark.dynamicAllocation.initialExecutors=1
          "
```

3. Menerapkan spesifikasi pod ke cluster Amazon EKS Anda:

```
kubectl apply -f notebook.yaml -n namespace
```

Ini akan memulai JupyterLab notebook minimal yang terhubung ke Amazon Anda EMR pada titik akhir EKS interaktif. Tunggu sampai pod itu RUNNING. Anda dapat memeriksa statusnya dengan perintah berikut:

```
kubectl get pod jupyter-notebook -n namespace
```

Ketika pod siap, `get pod` perintah mengembalikan output yang mirip dengan ini:

```
NAME              READY   STATUS    RESTARTS   AGE
jupyter-notebook  1/1     Running   0          46s
```

4. Lampirkan grup keamanan notebook ke node tempat notebook dijadwalkan.

   a. Pertama, identifikasi node tempat `jupyter-notebook` pod dijadwalkan dengan `describe` pod perintah.

   ```
   kubectl describe pod jupyter-notebook -n namespace
   ```

b. Buka EKS konsol Amazon di https://console.aws.amazon.com/eks/rumah#/cluster.

c. Arahkan ke tab Compute untuk EKS klaster Amazon Anda dan pilih node yang diidentifikasi oleh `describe pod` perintah. Pilih ID instance untuk node.

d. Dari menu Tindakan, pilih Keamanan > Ubah grup keamanan untuk melampirkan grup keamanan yang Anda buatMembuat grup keamanan.

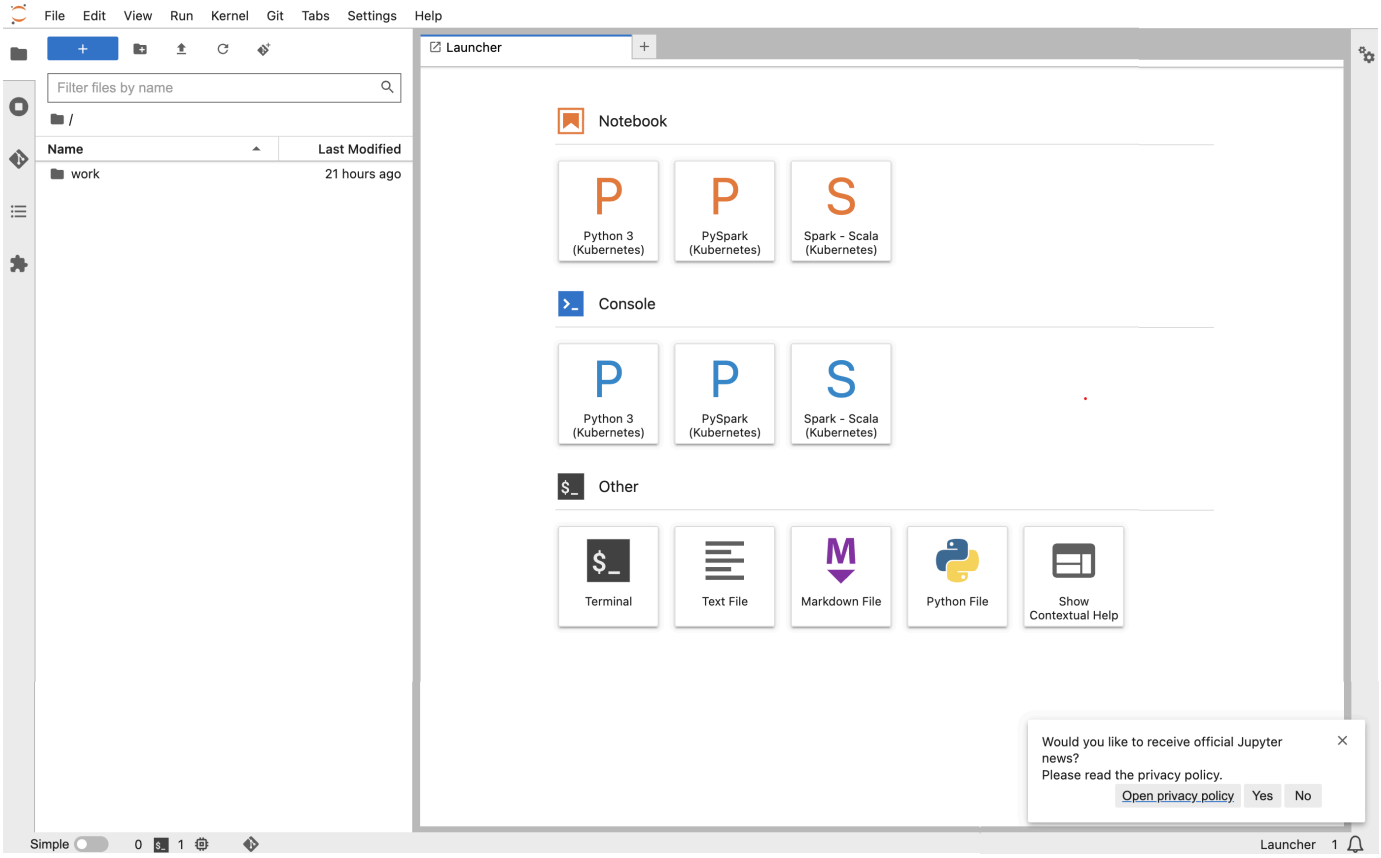
e. Jika Anda menerapkan pod notebook Jupyter AWS Fargate, buat a `SecurityGroupPolicy`to apply ke pod notebook Jupyter dengan label peran:

```
cat >my-security-group-policy.yaml <<EOF
apiVersion: vpcresources.k8s.aws/v1beta1
kind: SecurityGroupPolicy
metadata:
  name: example-security-group-policy-name
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: example-role-name-label
  securityGroups:
    groupIds:
      - your-notebook-security-group-id
EOF
```

5. Sekarang, port-forward sehingga Anda dapat mengakses antarmuka secara lokal: JupyterLab

```
kubectl port-forward jupyter-notebook 8888:8888 -n namespace
```

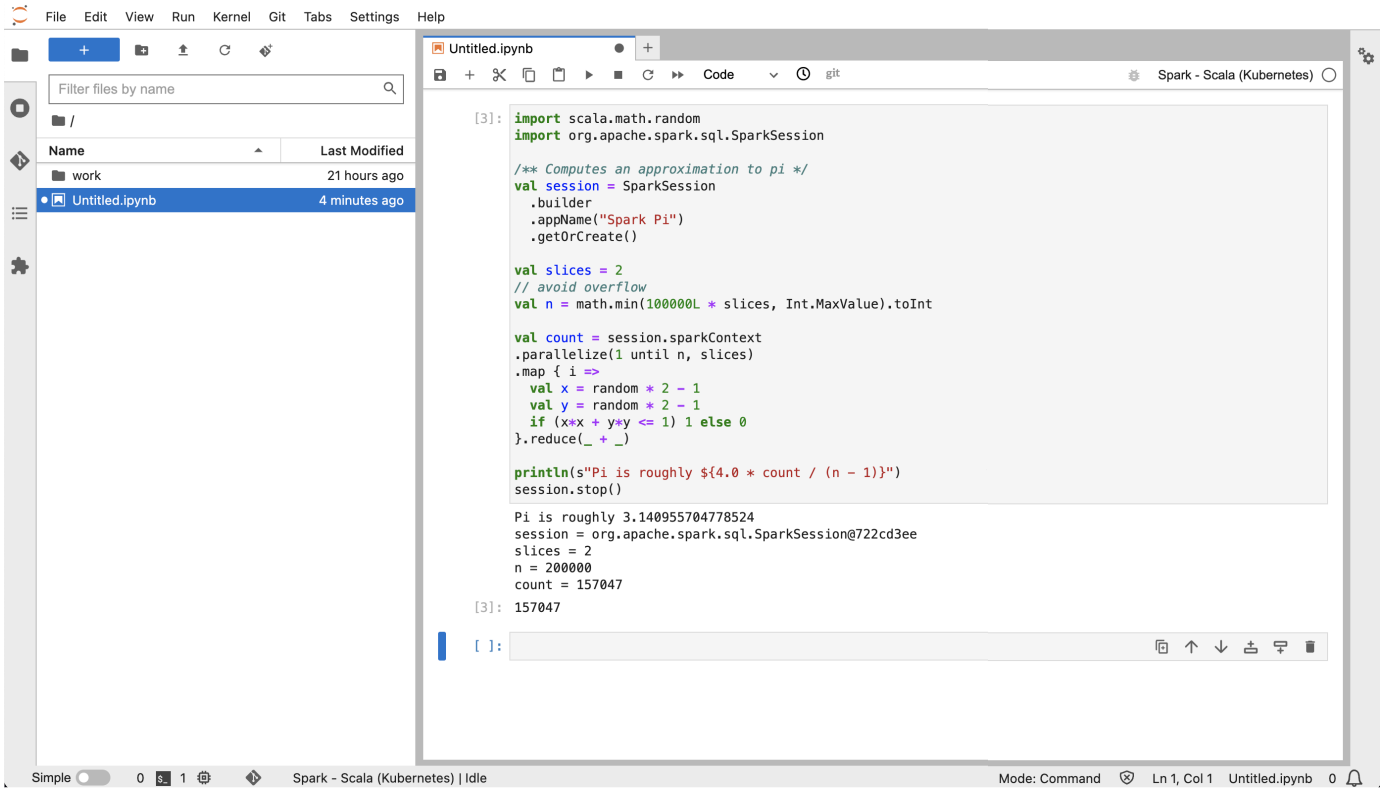
Setelah berjalan, navigasikan ke browser lokal Anda dan kunjungi `localhost:8888` untuk melihat JupyterLab antarmuka:



6. Dari JupyterLab, buat notebook Scala baru. Berikut adalah contoh cuplikan kode yang dapat Anda jalankan untuk memperkirakan nilai Pi:

```
import scala.math.random
import org.apache.spark.sql.SparkSession

/** Computes an approximation to pi */
val session = SparkSession
  .builder
  .appName("Spark Pi")
  .getOrCreate()

val slices = 2
// avoid overflow
val n = math.min(100000L * slices, Int.MaxValue).toInt

val count = session.sparkContext
.parallelize(1 until n, slices)
.map { i =>
  val x = random * 2 - 1
  val y = random * 2 - 1
```

```
  if (x*x + y*y <= 1) 1 else 0
}.reduce(_ + _)

println(s"Pi is roughly ${4.0 * count / (n - 1)}")
session.stop()
```



Menghapus buku catatan Jupyter yang dihosting sendiri

Saat Anda siap untuk menghapus buku catatan yang dihosting sendiri, Anda juga dapat menghapus titik akhir interaktif dan grup keamanan juga. Lakukan tindakan dengan urutan sebagai berikut:

1. Gunakan perintah berikut untuk menghapus `jupyter-notebook` pod:

   ```
   kubectl delete pod jupyter-notebook -n namespace
   ```

2. Kemudian, hapus endpoint interaktif Anda dengan `delete-managed-endpoint` perintah. Untuk langkah-langkah menghapus titik akhir interaktif, lihatHapus titik akhir interaktif. Awalnya, titik akhir Anda akan berada di TERMINATINGnegara bagian. Setelah semua sumber daya dibersihkan, ia beralih ke TERMINATEDnegara bagian.

3. Jika Anda tidak berencana untuk menggunakan grup keamanan notebook yang Anda buat Membuat grup keamanan untuk penerapan notebook Jupyter lainnya, Anda dapat menghapusnya. Lihat Menghapus grup keamanan di Panduan EC2 Pengguna Amazon untuk informasi selengkapnya.

Operasi lain pada titik akhir interaktif

Topik ini mencakup operasi yang didukung pada titik akhir interaktif selain `create-managed-endpoint`.

Ambil detail titik akhir interaktif

Setelah Anda membuat endpoint interaktif, Anda dapat mengambil detailnya menggunakan perintah. `describe-managed-endpoint` AWS CLI Masukkan nilai Anda sendiri untuk *`managed-endpoint-id`*, *`virtual-cluster-id`*, dan *`region`*:

```
aws emr-containers describe-managed-endpoint --id managed-endpoint-id \
 --virtual-cluster-id virtual-cluster-id --region region
```

Outputnya terlihat mirip dengan berikut ini, dengan titik akhir yang ditentukan, sepertiARN, ID, dan nama.

```
{
   "id": "as3ys2xxxxxxx",
   "name": "endpoint-name",
    "arn": "arn:aws:emr-containers:us-east-1:1828xxxxxxxx:/virtualclusters/
lbhl6kwwyoxxxxxxxxxxxxxxx/endpoints/as3ysxxxxxxxx",
    "virtualClusterId": "lbhl6kwwyoxxxxxxxxxxxxxxx",
    "type": "JUPYTER_ENTERPRISE_GATEWAY",
    "state": "ACTIVE",
    "releaseLabel": "emr-6.9.0-latest",
   "executionRoleArn": "arn:aws:iam::1828xxxxxxxx:role/RoleName",
    "certificateAuthority": {
        "certificateArn": "arn:aws:acm:us-east-1:1828xxxxxxxx:certificate/zzzzzzzz-
e59b-4ed0-aaaa-bbbbbbbbbbbb",
        "certificateData": "certificate-data"
    },
    "configurationOverrides": {
        "applicationConfiguration": [
            {
```

```
                "classification": "spark-defaults",
                "properties": {
                    "spark.driver.memory": "8G"
                }
            }
        ],
        "monitoringConfiguration": {
            "persistentAppUI": "ENABLED",
            "cloudWatchMonitoringConfiguration": {
                "logGroupName": "log-group-name",
                "logStreamNamePrefix": "log-stream-name-prefix"
            },
            "s3MonitoringConfiguration": {
                "logUri": "s3-bucket-name"
            }
        }
    },
   "serverUrl": "https://internal-k8s-namespace-ingressa-aaaaaaaaaa-
zzzzzzzzzz.us-east-1.elb.amazonaws.com:18888 (https://internal-k8s-nspluto-
ingressa-51e860abbd-1620715833.us-east-1.elb.amazonaws.com:18888/)",
    "createdAt": "2022-09-19T12:37:49+00:00",
    "securityGroup": "sg-aaaaaaaaaaaaaa",
    "subnetIds": [
        "subnet-11111111111",
        "subnet-22222222222",
        "subnet-33333333333"
    ],
    "stateDetails": "Endpoint created successfully. It took 3 Minutes 15 Seconds",
    "tags": {}
 }
```

Buat daftar semua titik akhir interaktif yang terkait dengan cluster virtual

Gunakan `list-managed-endpoints` AWS CLI perintah untuk mengambil daftar semua endpoint interaktif yang terkait dengan cluster virtual tertentu. Ganti `virtual-cluster-id` dengan ID cluster virtual Anda.

```
aws emr-containers list-managed-endpoints --virtual-cluster-id virtual-cluster-id
```

Output dari `list-managed-endpoint` perintah ditunjukkan di bawah ini:

```
{
```

```
    "endpoints": [{
        "id": "as3ys2xxxxxxx",
        "name": "endpoint-name",
        "arn": "arn:aws:emr-containers:us-east-1:1828xxxxxxxx:/virtualclusters/
lbhl6kwwyoxxxxxxxxxxxxxx/endpoints/as3ysxxxxxxxx",
        "virtualClusterId": "lbhl6kwwyoxxxxxxxxxxxxxx",
        "type": "JUPYTER_ENTERPRISE_GATEWAY",
        "state": "ACTIVE",
        "releaseLabel": "emr-6.9.0-latest",
        "executionRoleArn": "arn:aws:iam::1828xxxxxxxx:role/RoleName",
        "certificateAuthority": {
            "certificateArn": "arn:aws:acm:us-east-1:1828xxxxxxxx:certificate/zzzzzzzz-
e59b-4ed0-aaaa-bbbbbbbbbbbb",
            "certificateData": "certificate-data"
        },
        "configurationOverrides": {
            "applicationConfiguration": [{
                "classification": "spark-defaults",
                "properties": {
                    "spark.driver.memory": "8G"
                }
            }],
            "monitoringConfiguration": {
                "persistentAppUI": "ENABLED",
                "cloudWatchMonitoringConfiguration": {
                    "logGroupName": "log-group-name",
                    "logStreamNamePrefix": "log-stream-name-prefix"
                },
                "s3MonitoringConfiguration": {
                    "logUri": "s3-bucket-name"
                }
            }
        },
        "serverUrl": "https://internal-k8s-namespace-ingressa-aaaaaaaaaa-
zzzzzzzzzz.us-east-1.elb.amazonaws.com:18888 (https://internal-k8s-nspluto-
ingressa-51e860abbd-1620715833.us-east-1.elb.amazonaws.com:18888/)",
        "createdAt": "2022-09-19T12:37:49+00:00",
        "securityGroup": "sg-aaaaaaaaaaaaa",
        "subnetIds": [
            "subnet-11111111111",
            "subnet-22222222222",
            "subnet-33333333333"
        ],
        "stateDetails": "Endpoint created successfully. It took 3 Minutes 15 Seconds",
```

```
        "tags": {}
    }]
}
```

Hapus titik akhir interaktif

Untuk menghapus titik akhir interaktif yang terkait dengan Amazon EMR di klaster EKS virtual, gunakan `delete-managed-endpoint` AWS CLI perintah. Saat Anda menghapus titik akhir interaktif, EMR Amazon akan EKS menghapus grup keamanan default yang dibuat untuk titik akhir tersebut.

Tentukan nilai untuk parameter berikut ke perintah:

- --id: Pengenal titik akhir interaktif yang ingin Anda hapus.
- -- virtual-cluster-id — Pengidentifikasi cluster virtual yang terkait dengan titik akhir interaktif yang ingin Anda hapus. Ini adalah ID cluster virtual yang sama yang ditentukan ketika endpoint interaktif dibuat.

```
aws emr-containers delete-managed-endpoint --id managed-endpoint-id --virtual-cluster-
id virtual-cluster-id
```

Perintah mengembalikan output yang mirip dengan berikut ini untuk mengonfirmasi bahwa Anda menghapus titik akhir interaktif:

```
{
    "id":"8gai4l4exxxxx",
    "virtualClusterId":"0b0qvauoy3ch1nqodxxxxxxxx"
}
```

Mengunggah data ke Amazon S3 Express One Zone dengan Amazon di EMR EKS

Dengan Amazon EMR merilis 7.2.0 dan yang lebih tinggi, Anda dapat menggunakan EMR Amazon EKS dengan kelas penyimpanan Amazon S3 Express One Zone untuk meningkatkan kinerja saat menjalankan pekerjaan dan beban kerja. S3 Express One Zone adalah kelas penyimpanan Amazon S3 zona tunggal berkinerja tinggi yang memberikan akses data milidetik satu digit yang konsisten untuk sebagian besar aplikasi yang sensitif terhadap latensi. Pada saat rilis, S3 Express One Zone memberikan latensi terendah dan penyimpanan objek cloud kinerja tertinggi di Amazon S3.

Prasyarat

Sebelum Anda dapat menggunakan S3 Express One Zone dengan EMR Amazon aktifEKS, Anda harus memiliki prasyarat berikut:

- Selesai menyiapkan Amazon EMR di EKS.
- Setelah Anda mengatur EMR AmazonEKS, buat cluster virtual.

Memulai dengan S3 Express One Zone

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

1. Tambahkan `CreateSession` izin ke peran eksekusi pekerjaan Anda. Ketika S3 Express One Zone awalnya melakukan tindakan sepertiGET,LIST, atau PUT pada objek S3, kelas penyimpanan memanggil `CreateSession` atas nama Anda. Berikut ini adalah contoh cara memberikan `CreateSession` izin.

```
{
  "Version":"2012-10-17",
  "Statement": [
        {
            "Effect": "Allow",
            "Resource": "arn:aws:s3express:<AWS_REGION>:<ACCOUNT_ID>:bucket/DOC-
EXAMPLE-BUCKET",
            "Action": [
                "s3express:CreateSession"
            ]
```

```
        }
    ]
 }
```

2. Anda harus menggunakan konektor Apache Hadoop S3A untuk mengakses bucket S3 Express, jadi ubah Amazon S3 Anda untuk menggunakan skema untuk menggunakan konektor. URIs `s3a` Jika mereka tidak menggunakan skema, Anda dapat mengubah implementasi sistem file yang Anda gunakan untuk `s3` dan skema. `s3n`

   Untuk mengubah `s3` skema, tentukan konfigurasi cluster berikut:

```
[
  {
    "Classification": "core-site",
    "Properties": {
      "fs.s3.impl": "org.apache.hadoop.fs.s3a.S3AFileSystem",
      "fs.AbstractFileSystem.s3.impl": "org.apache.hadoop.fs.s3a.S3A"
    }
  }
]
```

   Untuk mengubah skema s3n, tentukan konfigurasi cluster berikut:

```
[
  {
    "Classification": "core-site",
    "Properties": {
      "fs.s3n.impl": "org.apache.hadoop.fs.s3a.S3AFileSystem",
      "fs.AbstractFileSystem.s3n.impl": "org.apache.hadoop.fs.s3a.S3A"
    }
  }
]
```

3. Dalam konfigurasi spark-submit Anda, gunakan penyedia kredensi identitas web.

```
"spark.hadoop.fs.s3a.aws.credentials.provider=com.amazonaws.auth.WebIdentityTokenCredential
```

Pemantauan tugas

Topik

Pantau pekerjaan dengan Amazon CloudWatch Events

Amazon EMR di EKS menyiarkan peristiwa ketika keadaan tugas berjalan berubah. Setiap peristiwa memberikan informasi, seperti tanggal dan waktu ketika peristiwa terjadi, bersama dengan detail lebih lanjut tentang peristiwa, seperti ID klaster virtual dan ID tugas berjalan yang terpengaruh.

Anda dapat menggunakan peristiwa untuk melacak aktivitas dan kesehatan tugas yang Anda jalankan pada klaster virtual. Anda juga dapat menggunakan Amazon CloudWatch Events untuk menentukan tindakan yang akan diambil saat menjalankan pekerjaan menghasilkan peristiwa yang cocok dengan pola yang Anda tentukan. Peristiwa berguna untuk memantau kejadian tertentu selama siklus hidup dari tugas berjalan. Misalnya, Anda dapat memantau saat status tugas berjalan berubah dari `submitted` ke `running`. Untuk informasi selengkapnya tentang CloudWatch Acara, lihat Panduan EventBridge Pengguna Amazon.

Tabel berikut mencantumkan peristiwa Amazon EMR di EKS, bersama dengan status atau perubahan status yang ditunjukkan peristiwa, tingkat kepelikan peristiwa, dan pesan peristiwa. Setiap peristiwa direpresentasikan sebagai objek JSON yang dikirim secara otomatis ke alur kejadian. Objek JSON mencakup detail lebih lanjut tentang peristiwa tersebut. Objek JSON sangat penting ketika Anda mengatur aturan untuk pemrosesan acara menggunakan CloudWatch Acara karena aturan berusaha untuk mencocokkan pola dalam objek JSON. Untuk informasi selengkapnya, lihat pola EventBridge acara Amazon dan EMR Amazon di Acara EKS di EventBridge Panduan Pengguna Amazon.

Peristiwa perubahan status tugas berjalan

Status	Kepelikan	Pesan
DIKIRIMKAN	INFO	Job Run *JobRunId*(*JobRunName*) berhasil dikirimkan ke cluster virtual *VirtualCl usterId* di *Time* UTC.
BERJALAN	INFO	Job Run *JobRunId*(*JobRunName*) di klaster virtual *VirtualClusterId* mulai berjalan pada *Waktu*.
DISELESAIKAN	INFO	Job Run *jobRunId*(*JobRunName*) di klaster virtual *VirtualClusterId* selesai pada *Waktu*. Tugas Berjalan mulai berjalan pada *Waktu* dan memakan waktu *Num* menit untuk diselesaikan.
DIBATALKAN	PERINGATAN	Permintaan pembatalan telah berhasil untuk Job Run *JobRunId*(*JobRunName*) di klaster virtual *VirtualClusterId* saat *Time* dan Job Run sekarang dibatalkan.
Failed	ERROR	Job Run *JobRunId*(*JobRunName*) di klaster virtual *VirtualClusterId* gagal pada *Waktu*.

Otomatiskan Amazon EMR di EKS dengan Acara CloudWatch

Anda dapat menggunakan Amazon CloudWatch Events untuk mengotomatiskan AWS layanan Anda guna merespons peristiwa sistem seperti masalah ketersediaan aplikasi atau perubahan sumber daya. Acara dari AWS layanan dikirim ke CloudWatch Acara dalam waktu dekat. Anda dapat menuliskan aturan sederhana untuk menunjukkan peristiwa mana yang sesuai kepentingan Anda, dan tindakan otomatis yang diambil ketika suatu peristiwa sesuai dengan suatu aturan. Tindakan yang dapat dipicu secara otomatis meliputi hal berikut:

- Memanggil fungsi AWS Lambda
- Meminta Perintah Amazon EC2 Run

- Mengirim peristiwa ke Amazon Kinesis Data Streams
- Mengaktifkan mesin AWS Step Functions negara
- Memberitahu topik Amazon Simple Notification Service (SNS) atau antrian Amazon Simple Queue Service (SQS)

Beberapa contoh penggunaan CloudWatch Acara dengan Amazon EMR di EKS meliputi yang berikut:

- Mengaktifkan fungsi Lambda ketika tugas berjalan berhasil
- Memberi tahu topik Amazon SNS saat pekerjaan berjalan gagal

CloudWatch Acara untuk "`detail-type:`" "`EMR Job Run State Change`" dihasilkan oleh Amazon EMR di EKS untuk`SUBMITTED`,, `RUNNING``CANCELLED`, `FAILED` dan perubahan `COMPLETED` status.

Contoh: Mengatur aturan yang memanggil Lambda

Gunakan langkah-langkah berikut untuk menyiapkan aturan CloudWatch Acara yang memanggil Lambda ketika ada acara “EMR Job Run State Change”.

```
aws events put-rule \
--name cwe-test \
--event-pattern '{"detail-type": ["EMR Job Run State Change"]}'
```

Tambahkan fungsi Lambda yang Anda miliki sebagai target baru dan berikan izin CloudWatch Acara untuk menjalankan fungsi Lambda sebagai berikut. Ganti *123456789012* ID akun Anda.

```
aws events put-targets \
--rule cwe-test \
--targets Id=1,Arn=arn:aws:lambda:us-east-1:123456789012:function:MyFunction
```

```
aws lambda add-permission \
--function-name MyFunction \
--statement-id MyId \
--action 'lambda:InvokeFunction' \
--principal events.amazonaws.com
```

> **Note**
>
> Anda tidak dapat menulis program yang tergantung pada urutan keberadaan atau notifikasi peristiwa, karena program tersebut mungkin tidak berurutan atau hilang. Kejadian dipancarkan atas dasar upaya terbaik.

Pantau pod driver job dengan kebijakan coba lagi menggunakan Amazon CloudWatch Events

Dengan menggunakan CloudWatch peristiwa, Anda dapat memantau pod driver yang telah dibuat dalam pekerjaan yang memiliki kebijakan coba ulang. Untuk informasi selengkapnya, lihat Memantau pekerjaan dengan kebijakan coba lagi dalam panduan ini.

Mengelola klaster virtual

Sebuah klaster virtual adalah namespace Kubernetes tempat Amazon EMR terdaftar. Anda dapat menciptakan, menggambarkan, membuat daftar, dan menghapus klaster virtual. Mereka tidak mengonsumsi sumber daya tambahan apa pun dalam sistem Anda. Sebuah klaster virtual tunggal memetakan ke satu namespace Kubernetes. Mengingat hubungan ini, Anda dapat memodelkan klaster virtual dengan cara yang sama Anda memodelkan namespace Kubernetes untuk memenuhi persyaratan Anda. Lihat kemungkinan kasus penggunaan di dokumentasi Gambaran Umum Konsep Kubernetes.

Untuk mendaftar Amazon EMR dengan namespace Kubernetes pada klaster Amazon EKS, Anda perlu nama klaster EKS dan namespace yang telah diatur untuk menjalankan beban kerja Anda. Klaster terdaftar tersebut di Amazon EMR disebut klaster virtual karena mereka tidak mengelola komputasi atau penyimpanan fisik tetapi menunjuk ke namespace Kubernetes di mana beban kerja Anda dijadwalkan.

> **Note**
>
> Sebelum membuat klaster virtual, Anda harus terlebih dahulu menyelesaikan langkah 1-8 di Menyiapkan Amazon EMR di EKS.

Topik

Membuat klaster virtual

Jalankan perintah berikut untuk membuat klaster virtual dengan mendaftarkan Amazon EMR dengan namespace pada klaster EKS. Ganti *virtual_cluster_name* dengan nama yang Anda berikan untuk klaster virtual Anda. Ganti *eks_cluster_name* dengan nama klaster EKS. Ganti *namespace_name* dengan namespace yang ingin Anda daftarkan dengan Amazon EMR.

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

Atau, Anda dapat membuat file JSON yang mencakup parameter yang diperlukan untuk klaster virtual, seperti yang ditunjukkan contoh berikut.

```
{
    "name": "virtual_cluster_name",
    "containerProvider": {
        "type": "EKS",
        "id": "eks_cluster_name",
        "info": {
            "eksInfo": {
                "namespace": "namespace_name"
            }
        }
    }
}
```

Kemudian jalankan perintah `create-virtual-cluster` berikut dengan jalur ke file JSON.

```
aws emr-containers create-virtual-cluster \
--cli-input-json file://./create-virtual-cluster-request.json
```

> **Note**
>
> Untuk memvalidasi keberhasilan pembuatan klaster virtual, lihat status klaster virtual dengan menjalankan perintah `list-virtual-clusters` atau dengan masuk ke halaman Klaster virtual di konsol Amazon EMR.

Daftar klaster virtual

Jalankan perintah berikut untuk menampilkan status klaster virtual.

```
aws emr-containers list-virtual-clusters
```

Gambarkan klaster virtual

Jalankan perintah berikut untuk mendapatkan detail lebih lanjut tentang klaster virtual, seperti namespace, status, dan tanggal terdaftar. Ganti *123456* dengan ID klaster virtual Anda.

```
aws emr-containers describe-virtual-cluster --id 123456
```

Menghapus klaster virtual

Jalankan perintah berikut untuk menghapus klaster virtual. Ganti *123456* dengan ID klaster virtual Anda.

```
aws emr-containers delete-virtual-cluster --id 123456
```

Status klaster virtual

Tabel berikut menjelaskan empat kemungkinan status klaster virtual.

State	Deskripsi
RUNNING	Klaster virtual berada dalam status RUNNING.
TERMINATING	Penghentian klaster virtual yang diminta sedang berlangsung.
TERMINATED	Penghentian yang diminta selesai.
ARRESTED	Penghentian yang diminta gagal karena izin yang tidak mencukupi.

Tutorial untuk Amazon EMR di EKS

Bagian ini menjelaskan kasus penggunaan umum ketika Anda bekerja dengan Amazon EMR pada EKS aplikasi.

Topik

Menggunakan Delta Lake dengan Amazon EMR di EKS

Untuk menggunakan Delta Lake dengan Amazon EMR pada aplikasi EKS

1. Saat Anda memulai pekerjaan untuk mengirimkan pekerjaan Spark dalam konfigurasi aplikasi, sertakan file JAR Delta Lake:

```
--job-driver '{"sparkSubmitJobDriver" : {
      "sparkSubmitParameters" : "--jars local:///usr/share/aws/delta/lib/delta-
core.jar,local:///usr/share/aws/delta/lib/delta-storage.jar,local:///usr/share/aws/
delta/lib/delta-storage-s3-dynamodb.jar"}}'
```

> **Note**
>
> Amazon EMR merilis 7.0.0 dan yang lebih tinggi menggunakan Delta Lake 3.0, yang berganti nama menjadi. `delta-core.jar` `delta-spark.jar` Jika Anda menggunakan Amazon EMR rilis 7.0.0 atau lebih tinggi, pastikan untuk menggunakan nama file yang benar, seperti dalam contoh berikut:

```
--jars local:///usr/share/aws/delta/lib/delta-spark.jar
```

2. Sertakan konfigurasi tambahan Delta Lake dan gunakan AWS Glue Data Catalog sebagai metastore Anda.

```
--configuration-overrides '{
        "applicationConfiguration": [
        {
          "classification" : "spark-defaults",
          "properties" : {
            "spark.sql.extensions" : "io.delta.sql.DeltaSparkSessionExtension",

 "spark.sql.catalog.spark_catalog":"org.apache.spark.sql.delta.catalog.DeltaCatalog",
"spark.hadoop.hive.metastore.client.factory.class":"com.amazonaws.glue.catalog.metastore.AW
           }
        }]}'
```

Menggunakan Apache Iceberg EMR di EKS

Menggunakan Apache Iceberg EMR di EKS

1. Saat Anda memulai pekerjaan untuk mengirimkan pekerjaan Spark dalam konfigurasi aplikasi, sertakan file JAR runtime Iceberg spark:

```
--job-driver '{"sparkSubmitJobDriver" : {"sparkSubmitParameters" : "--jars
 local:///usr/share/aws/iceberg/lib/iceberg-spark3-runtime.jar"}}'
```

2. Sertakan konfigurasi tambahan Iceberg:

```
--configuration-overrides '{
    "applicationConfiguration": [
    "classification" : "spark-defaults",
    "properties" : {
        "spark.sql.catalog.dev.warehouse" : "s3://DOC-EXAMPLE-BUCKET/EXAMPLE-
PREFIX/ ",
        "spark.sql.extensions ":"
 org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions ",
        "spark.sql.catalog.dev" : "org.apache.iceberg.spark.SparkCatalog",
```

```
        "spark.sql.catalog.dev.catalog-impl" :
 "org.apache.iceberg.aws.glue.GlueCatalog",
        "spark.sql.catalog.dev.io-impl": "org.apache.iceberg.aws.s3.S3FileIO"
        }
    ]
}'
```

Untuk mempelajari lebih lanjut tentang versi rilis Apache Iceberg dari EMR, lihat riwayat rilis Iceberg.

Menggunakan PyFlink

Amazon EMR di EKS merilis 6.15.0 dan dukungan yang lebih tinggi. PyFlink Jika Anda sudah memiliki PyFlink skrip, Anda dapat melakukan salah satu hal berikut:

- Buat gambar kustom dengan PyFlink skrip Anda disertakan.
- Unggah skrip Anda ke lokasi Amazon S3

Jika Anda belum memiliki skrip, Anda dapat menggunakan contoh berikut untuk meluncurkan PyFlink pekerjaan. Contoh ini mengambil skrip dari S3. Jika Anda menggunakan gambar kustom dengan skrip yang sudah disertakan dalam gambar, Anda harus memperbarui jalur skrip ke lokasi tempat Anda menyimpan skrip Anda. Jika skrip berada di lokasi S3, Amazon EMR di EKS akan mengambil skrip dan menempatkannya di bawah `/opt/flink/usrlib/` direktori dalam wadah Flink.

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: python-example
spec:
  flinkVersion: v1_17
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "1"
  executionRoleArn: job-execution-role
  emrReleaseLabel: "emr-6.15.0-flink-latest"
  jobManager:
    highAvailabilityEnabled: false
    replicas: 1
    resource:
      memory: "2048m"
      cpu: 1
```

```
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
  job:
    jarURI: s3://S3 bucket with your script/pyflink-script.py
    entryClass: "org.apache.flink.client.python.PythonDriver"
    args: ["-py", "/opt/flink/usrlib/pyflink-script.py"]
    parallelism: 1
    upgradeMode: stateless
```

Menggunakan AWS Glue dengan Flink

Amazon EMR di EKS dengan Apache Flink merilis 6.15.0 dan dukungan yang lebih tinggi menggunakan Katalog Data AWS Glue sebagai penyimpanan metadata untuk streaming dan alur kerja SQL batch.

Anda harus terlebih dahulu membuat database AWS Glue bernama `default` yang berfungsi sebagai Katalog SQL Flink Anda. Katalog Flink ini menyimpan metadata seperti database, tabel, parisi, tampilan, fungsi, dan informasi lain yang diperlukan untuk mengakses data di sistem eksternal lainnya.

```
aws glue create-database \
    --database-input "{\"Name\":\"default\"}"
```

Untuk mengaktifkan dukungan AWS Glue, gunakan `FlinkDeployment` spesifikasi. Contoh spesifikasi ini menggunakan skrip Python untuk dengan cepat mengeluarkan beberapa pernyataan SQL Flink untuk berinteraksi dengan katalog Glue. AWS

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: python-example
spec:
  flinkVersion: v1_17
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "1"
    aws.glue.enabled: "true"
  executionRoleArn: job-execution-role-arn;
  emrReleaseLabel: "emr-6.15.0-flink-latest"
  jobManager:
```

```
    highAvailabilityEnabled: false
    replicas: 1
    resource:
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
  job:
    jarURI: s3://<S3_bucket_with_your_script/pyflink-glue-script.py
    entryClass: "org.apache.flink.client.python.PythonDriver"
    args: ["-py", "/opt/flink/usrlib/pyflink-glue-script.py"]
    parallelism: 1
    upgradeMode: stateless
```

Berikut ini adalah contoh dari apa PyFlink script Anda mungkin terlihat seperti.

```
import logging
import sys
from pyflink.datastream import StreamExecutionEnvironment
from pyflink.table import StreamTableEnvironment

def glue_demo():
    env = StreamExecutionEnvironment.get_execution_environment()
    t_env = StreamTableEnvironment.create(stream_execution_environment=env)
    t_env.execute_sql("""
          CREATE CATALOG glue_catalog WITH (
          'type' = 'hive',
          'default-database' = 'default',
          'hive-conf-dir' = '/glue/confs/hive/conf',
          'hadoop-conf-dir' = '/glue/confs/hadoop/conf'
          )
                      """)
    t_env.execute_sql("""
          USE CATALOG glue_catalog;
                      """)
    t_env.execute_sql("""
          DROP DATABASE IF EXISTS eks_flink_db CASCADE;
                      """)
    t_env.execute_sql("""
          CREATE DATABASE IF NOT EXISTS eks_flink_db WITH ('hive.database.location-
uri'= 's3a://S3-bucket-to-store-metadata/flink/flink-glue-for-hive/warehouse/');
```

```
                    """)
    t_env.execute_sql("""
          USE eks_flink_db;
                  """)
    t_env.execute_sql("""
          CREATE TABLE IF NOT EXISTS eksglueorders (
            order_number BIGINT,
            price        DECIMAL(32,2),
            buyer        RO first_name STRING, last_name STRING,
            order_time   TIMESTAMP(3)
          ) WITH (
            'connector' = 'datagen'
          );
                    """)
    t_env.execute_sql("""
          CREATE TABLE IF NOT EXISTS eksdestglueorders (
            order_number BIGINT,
            price        DECIMAL(32,2),
            buyer        ROW first_name STRING, last_name STRING,
            order_time   TIMESTAMP(3)
          ) WITH (
            'connector' = 'filesystem',
            'path' = 's3://S3-bucket-to-store-metadata/flink/flink-glue-for-hive/
warehouse/eksdestglueorders',
            'format' = 'json'
          );
                  """)
    t_env.execute_sql("""
          CREATE TABLE IF NOT EXISTS print_table (
            order_number BIGINT,
            price        DECIMAL(32,2),
            buyer        ROW first_name STRING, last_name STRING,
            order_time   TIMESTAMP(3)
          ) WITH (
            'connector' = 'print'
          );
                """)
    t_env.execute_sql("""
          EXECUTE STATEMENT SET
          BEGIN
          INSERT INTO eksdestglueorders SELECT * FROM  eksglueorders LIMIT 10;
          INSERT INTO print_table SELECT * FROM eksdestglueorders;
          END;
            """)
```

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

Menggunakan Apache Hudi dengan Apache Flink

Apache Hudi adalah kerangka kerja manajemen data sumber terbuka dengan operasi tingkat rekaman seperti menyisipkan, memperbarui, meningkatkan, dan menghapus yang dapat Anda gunakan untuk menyederhanakan manajemen data dan pengembangan pipa data. Ketika dikombinasikan dengan manajemen data yang efisien di Amazon S3, Hudi memungkinkan Anda menelan dan memperbarui data secara real time. Hudi mempertahankan metadata dari semua operasi yang Anda jalankan pada dataset, sehingga semua tindakan tetap atom dan konsisten.

Apache Hudi tersedia di Amazon EKS dengan Apache Flink dengan EMR EMR Amazon merilis 7.2.0 dan lebih tinggi. Lihat langkah-langkah berikut untuk mempelajari cara memulai dan mengirimkan pekerjaan Apache Hudi.

Kirim pekerjaan Apache Hudi

Lihat langkah-langkah berikut untuk mempelajari cara mengirimkan pekerjaan Apache Hudi.

1. Buat database AWS Glue bernama`default`.

   ```
   aws glue create-database --database-input "{\"Name\":\"default\"}"
   ```

2. Ikuti SQLContoh Operator Flink Kubernetes untuk membangun file. `flink-sql-runner.jar`
3. Buat SQL skrip Hudi seperti berikut ini.

   ```
   CREATE CATALOG hudi_glue_catalog WITH (
   'type' = 'hudi',
   'mode' = 'hms',
   'table.external' = 'true',
   'default-database' = 'default',
   'hive.conf.dir' = '/glue/confs/hive/conf/',
   'catalog.path' = 's3://<hudi-example-bucket>/FLINK_HUDI/warehouse/'
   );

   USE CATALOG hudi_glue_catalog;
   CREATE DATABASE IF NOT EXISTS hudi_db;
   ```

```
use hudi_db;

CREATE TABLE IF NOT EXISTS hudi-flink-example-table(
    uuid VARCHAR(20),
    name VARCHAR(10),
    age INT,
    ts TIMESTAMP(3),
    `partition` VARCHAR(20)
)
PARTITIONED BY (`partition`)
WITH (
  'connector' = 'hudi',
  'path' = 's3://<hudi-example-bucket>/hudi-flink-example-table',
  'hive_sync.enable' = 'true',
  'hive_sync.mode' = 'glue',
  'hive_sync.table' = 'hudi-flink-example-table',
  'hive_sync.db' = 'hudi_db',
  'compaction.delta_commits' = '1',
  'hive_sync.partition_fields' = 'partition',
  'hive_sync.partition_extractor_class' =
 'org.apache.hudi.hive.MultiPartKeysValueExtractor',
  'table.type' = 'COPY_ON_WRITE'
);

EXECUTE STATEMENT SET
BEGIN

INSERT INTO hudi-flink-example-table VALUES
    ('id1','Alex',23,TIMESTAMP '1970-01-01 00:00:01','par1'),
    ('id2','Stephen',33,TIMESTAMP '1970-01-01 00:00:02','par1'),
    ('id3','Julian',53,TIMESTAMP '1970-01-01 00:00:03','par2'),
    ('id4','Fabian',31,TIMESTAMP '1970-01-01 00:00:04','par2'),
    ('id5','Sophia',18,TIMESTAMP '1970-01-01 00:00:05','par3'),
    ('id6','Emma',20,TIMESTAMP '1970-01-01 00:00:06','par3'),
    ('id7','Bob',44,TIMESTAMP '1970-01-01 00:00:07','par4'),
    ('id8','Han',56,TIMESTAMP '1970-01-01 00:00:08','par4');

END;
```

4. Unggah SQL skrip Hudi Anda dan `flink-sql-runner.jar` file ke lokasi S3.
5. Dalam `FlinkDeployments` YAML file Anda, atur `hudi.enabled` ke`true`.

```
spec:
```

```
  flinkConfiguration:
    hudi.enabled: "true"
```

6. Buat YAML file untuk menjalankan konfigurasi Anda. File contoh ini diberi namahudi-write.yaml.

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: hudi-write-example
spec:
  flinkVersion: v1_18
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
    hudi.enabled: "true"
  executionRoleArn: "<JobExecutionRole>"
  emrReleaseLabel: "emr-7.2.0-flink-latest"
  jobManager:
    highAvailabilityEnabled: false
    replicas: 1
    resource:
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
  job:
    jarURI: local:///opt/flink/usrlib/flink-sql-runner.jar
    args: ["/opt/flink/scripts/hudi-write.sql"]
    parallelism: 1
    upgradeMode: stateless
  podTemplate:
    spec:
      initContainers:
        - name: flink-sql-script-download
          args:
            - s3
            - cp
            - s3://<s3_location>/hudi-write.sql
            - /flink-scripts
          image: amazon/aws-cli:latest
          imagePullPolicy: Always
```

```
          resources: {}
          terminationMessagePath: /dev/termination-log
          terminationMessagePolicy: File
          volumeMounts:
            - mountPath: /flink-scripts
              name: flink-scripts
        - name: flink-sql-runner-download
          args:
            - s3
            - cp
            - s3://<s3_location>/flink-sql-runner.jar
            - /flink-artifacts
          image: amazon/aws-cli:latest
          imagePullPolicy: Always
          resources: {}
          terminationMessagePath: /dev/termination-log
          terminationMessagePolicy: File
          volumeMounts:
            - mountPath: /flink-artifacts
              name: flink-artifact
      containers:
        - name: flink-main-container
          volumeMounts:
            - mountPath: /opt/flink/scripts
              name: flink-scripts
            - mountPath: /opt/flink/usrlib
              name: flink-artifact
      volumes:
        - emptyDir: {}
          name: flink-scripts
        - emptyDir: {}
          name: flink-artifact
```

7. Kirim pekerjaan Flink Hudi ke operator Flink Kubernetes.

```
kubectl apply -f hudi-write.yaml
```

Menggunakan RAPIDS Accelerator untuk Apache Spark dengan Amazon EMR di EKS

Dengan Amazon EMR di EKS, Anda dapat menjalankan pekerjaan untuk Nvidia RAPIDS Accelerator untuk Apache Spark. Tutorial ini mencakup cara menjalankan pekerjaan Spark menggunakan RAPIDS pada tipe instans unit pemrosesan grafis EC2 (GPU). Tutorial menggunakan versi berikut:

- Amazon EMR di EKS versi 6.9.0 dan versi yang lebih tinggi
- Apache Spark 3.x

Anda dapat mempercepat Spark dengan jenis instans GPU Amazon EC2 dengan menggunakan plugin Nvidia RAPIDS Accelerator for Apache Spark. Ketika Anda menggunakan teknologi ini bersama-sama, Anda mempercepat jaringan pipa ilmu data Anda tanpa harus membuat perubahan kode. Ini mengurangi waktu berjalan yang diperlukan untuk pemrosesan data dan pelatihan model. Dengan menyelesaikan lebih banyak dalam waktu yang lebih singkat, Anda menghabiskan lebih sedikit biaya infrastruktur.

Sebelum memulai, pastikan Anda memiliki sumber daya berikut.

- Amazon EMR pada klaster virtual EKS
- Klaster Amazon EKS dengan grup node yang diaktifkan GPU

Klaster virtual Amazon EKS adalah handel terdaftar untuk namespace Kubernetes pada klaster Amazon EKS, dan dikelola oleh Amazon EMR pada EKS. Handel ini memungkinkan Amazon EMR menggunakan namespace Kubernetes sebagai tujuan untuk menjalankan pekerjaan. Untuk informasi lebih lanjut tentang cara mengatur klaster virtual, lihat Menyiapkan Amazon EMR di EKS di panduan ini.

Anda harus mengonfigurasi klaster virtual Amazon EKS dengan grup node yang memiliki instans GPU. Anda harus mengkonfigurasi node dengan plugin perangkat Nvidia. Lihat grup node terkelola untuk mempelajari lebih lanjut.

Untuk mengonfigurasi klaster Amazon EKS Anda agar menambahkan grup node berkemampuan GPU, lakukan prosedur berikut:

Untuk menambahkan grup node yang diaktifkan GPU

1. Buat grup node berkemampuan GPU dengan perintah create-nodegroup berikut. Pastikan untuk mengganti parameter yang benar untuk klaster Amazon EKS Anda. Gunakan jenis instans yang mendukung Spark RAPIDS, seperti P4, P3, G5 atau G4dn.

```
aws eks create-nodegroup \
 --cluster-name EKS_CLUSTER_NAME \
 --nodegroup-name NODEGROUP_NAME \
 --scaling-config minSize=0,maxSize=5,desiredSize=2 CHOOSE_APPROPRIATELY \
 --ami-type AL2_x86_64_GPU \
 --node-role NODE_ROLE \
 --subnets SUBNETS_SPACE_DELIMITED  \
 --remote-access ec2SshKey= SSH_KEY \
 --instance-types GPU_INSTANCE_TYPE \
 --disk-size DISK_SIZE \
 --region AWS_REGION
```

2. Instal plugin perangkat Nvidia di klaster Anda untuk memancarkan jumlah GPU pada setiap node klaster Anda dan untuk menjalankan kontainer berkemampuan GPU di klaster Anda. Jalankan kode berikut untuk menginstal plugin:

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/v0.9.0/
nvidia-device-plugin.yml
```

3. Untuk memvalidasi berapa banyak GPU tersedia pada setiap node klaster Anda, jalankan perintah berikut ini:

```
kubectl get nodes  "-o=custom-
columns=NAME:.metadata.name,GPU:.status.allocatable.nvidia\.com/gpu"
```

Untuk menjalankan pekerjaan Spark RAPIDS

1. Kirim tugas Spark RAPIDS untuk Amazon EMR pada klaster EKS. Kode berikut menunjukkan contoh perintah untuk memulai pekerjaan. Pertama kali Anda menjalankan pekerjaan, mungkin perlu beberapa menit untuk men-download gambar dan cache pada node.

```
aws emr-containers start-job-run \
--virtual-cluster-id VIRTUAL_CLUSTER_ID \
--execution-role-arn JOB_EXECUTION_ROLE \
```

```
--release-label emr-6.9.0-spark-rapids-latest \
--job-driver '{"sparkSubmitJobDriver": {"entryPoint": "local:///usr/lib/
spark/examples/jars/spark-examples.jar","entryPointArguments":  ["10000"],
 "sparkSubmitParameters":"--class org.apache.spark.examples.SparkPi "}}' \
---configuration-overrides '{"applicationConfiguration": [{"classification":
 "spark-defaults","properties": {"spark.executor.instances":
 "2","spark.executor.memory": "2G"}}],"monitoringConfiguration":
 {"cloudWatchMonitoringConfiguration": {"logGroupName": "LOG_GROUP
 _NAME"},"s3MonitoringConfiguration": {"logUri": "LOG_GROUP_STREAM"}}}'
```

2. Untuk memvalidasi bahwa Akselerator Spark RAPIDS diaktifkan, periksa log driver Spark. Log ini disimpan di dalam CloudWatch atau di lokasi S3 yang Anda tentukan saat menjalankan `start-job-run` perintah. Contoh berikut umumnya menunjukkan apa garis log terlihat seperti:

```
22/11/15 00:12:44 INFO RapidsPluginUtils: RAPIDS Accelerator build:
 {version=22.08.0-amzn-0, user=release, url=, date=2022-11-03T03:32:45Z, revision=,
 cudf_version=22.08.0, branch=}
22/11/15 00:12:44 INFO RapidsPluginUtils: RAPIDS Accelerator JNI build:
 {version=22.08.0, user=, url=https://github.com/NVIDIA/spark-rapids-jni.git,
 date=2022-08-18T04:14:34Z, revision=a1b23cd_sample, branch=HEAD}
22/11/15 00:12:44 INFO RapidsPluginUtils: cudf build: {version=22.08.0,
 user=, url=https://github.com/rapidsai/cudf.git, date=2022-08-18T04:14:34Z,
 revision=a1b23ce_sample, branch=HEAD}
22/11/15 00:12:44 WARN RapidsPluginUtils: RAPIDS Accelerator 22.08.0-amzn-0 using
 cudf 22.08.0.
22/11/15 00:12:44 WARN RapidsPluginUtils:
 spark.rapids.sql.multiThreadedRead.numThreads is set to 20.
22/11/15 00:12:44 WARN RapidsPluginUtils: RAPIDS Accelerator is enabled, to disable
 GPU support set `spark.rapids.sql.enabled` to false.
22/11/15 00:12:44 WARN RapidsPluginUtils: spark.rapids.sql.explain is set to
 `NOT_ON_GPU`. Set it to 'NONE' to suppress the diagnostics logging about the query
 placement on the GPU.
```

3. Untuk melihat operasi yang akan dijalankan pada GPU, lakukan langkah-langkah berikut untuk mengaktifkan pencatatan ekstra. Catatan "`spark.rapids.sql.explain : ALL`" config.

```
aws emr-containers start-job-run \
--virtual-cluster-id VIRTUAL_CLUSTER_ID \
--execution-role-arn JOB_EXECUTION_ROLE \
--release-label emr-6.9.0-spark-rapids-latest \
--job-driver '{"sparkSubmitJobDriver": {"entryPoint": "local:///usr/lib/
spark/examples/jars/spark-examples.jar","entryPointArguments":  ["10000"],
 "sparkSubmitParameters":"--class org.apache.spark.examples.SparkPi "}}' \
```

```
---configuration-overrides '{"applicationConfiguration":
 [{"classification": "spark-defaults","properties":
 {"spark.rapids.sql.explain":"ALL","spark.executor.instances":
 "2","spark.executor.memory": "2G"}}],"monitoringConfiguration":
 {"cloudWatchMonitoringConfiguration": {"logGroupName":
 "LOG_GROUP_NAME"},"s3MonitoringConfiguration": {"logUri": "LOG_GROUP_STREAM"}}}'
```

Perintah sebelumnya adalah contoh pekerjaan yang menggunakan GPU. Outputnya akan terlihat seperti contoh di bawah ini. Lihat kunci ini untuk membantu memahami output:

- *- menandai operasi yang bekerja pada GPU
- !- menandai operasi yang tidak dapat berjalan pada GPU
- @- menandai operasi yang bekerja pada GPU, tetapi tidak akan bisa berjalan karena ada di dalam rencana yang tidak dapat dijalankan pada GPU

```
22/11/15 01:22:58 INFO GpuOverrides: Plan conversion to the GPU took 118.64 ms
22/11/15 01:22:58 INFO GpuOverrides: Plan conversion to the GPU took 4.20 ms
22/11/15 01:22:58 INFO GpuOverrides: GPU plan transition optimization took 8.37 ms
22/11/15 01:22:59 WARN GpuOverrides:
   *Exec <ProjectExec> will run on GPU
     *Expression <Alias> substring(cast(date#149 as string), 0, 7) AS month#310
will run on GPU
       *Expression <Substring> substring(cast(date#149 as string), 0, 7) will run
on GPU
         *Expression <Cast> cast(date#149 as string) will run on GPU
     *Exec <SortExec> will run on GPU
       *Expression <SortOrder> date#149 ASC NULLS FIRST will run on GPU
       *Exec <ShuffleExchangeExec> will run on GPU
         *Partitioning <RangePartitioning> will run on GPU
           *Expression <SortOrder> date#149 ASC NULLS FIRST will run on GPU
         *Exec <UnionExec> will run on GPU
           !Exec <ProjectExec> cannot run on GPU because not all expressions can
be replaced
             @Expression <AttributeReference> customerID#0 could run on GPU
             @Expression <Alias> Charge AS kind#126 could run on GPU
               @Expression <Literal> Charge could run on GPU
             @Expression <AttributeReference> value#129 could run on GPU
             @Expression <Alias> add_months(2022-11-15, cast(-(cast(_we0#142 as
bigint) + last_month#128L) as int)) AS date#149 could run on GPU
```

```
                ! <AddMonths> add_months(2022-11-15, cast(-
(cast(_we0#142 as bigint) + last_month#128L) as int)) cannot run
 on GPU because GPU does not currently support the operator class
 org.apache.spark.sql.catalyst.expressions.AddMonths
                  @Expression <Literal> 2022-11-15 could run on GPU
                  @Expression <Cast> cast(-(cast(_we0#142 as bigint) +
 last_month#128L) as int) could run on GPU
                    @Expression <UnaryMinus> -(cast(_we0#142 as bigint) +
 last_month#128L) could run on GPU
                      @Expression <Add> (cast(_we0#142 as bigint) +
 last_month#128L) could run on GPU
                        @Expression <Cast> cast(_we0#142 as bigint) could run on
 GPU
                          @Expression <AttributeReference> _we0#142 could run on
 GPU
                        @Expression <AttributeReference> last_month#128L could run
 on GPU
```

Menggunakan integrasi Amazon Redshift untuk Apache Spark di Amazon EMR di EKS

Dengan Amazon EMR rilis 6.9.0 dan yang lebih baru, setiap gambar rilis menyertakan konektor antara Apache Spark dan Amazon Redshift. Dengan cara ini, Anda dapat menggunakan Spark di Amazon EMR di EKS untuk memproses data yang disimpan di Amazon Redshift. Integrasi ini didasarkan pada konektor `spark-redshift` open-source. Untuk Amazon EMR di EKS, integrasi Amazon Redshift untuk Apache Spark disertakan sebagai integrasi asli.

Topik

- Meluncurkan aplikasi Spark menggunakan integrasi Amazon Redshift untuk Apache Spark
- Mengautentikasi dengan integrasi Amazon Redshift untuk Apache Spark
- Membaca dan menulis dari dan ke Amazon Redshift
- Pertimbangan dan batasan saat menggunakan konektor Spark

Meluncurkan aplikasi Spark menggunakan integrasi Amazon Redshift untuk Apache Spark

Untuk menggunakan integrasi, Anda harus meneruskan dependensi Spark Redshift yang diperlukan dengan pekerjaan Spark Anda. Anda harus menggunakan `--jars` untuk menyertakan pustaka terkait konektor Redshift. Untuk melihat lokasi file lain yang didukung oleh `--jars` opsi, lihat bagian Advanced Dependency Management dari dokumentasi Apache Spark.

- `spark-redshift.jar`
- `spark-avro.jar`
- `RedshiftJDBC.jar`
- `minimal-json.jar`

Untuk meluncurkan aplikasi Spark dengan integrasi Amazon Redshift untuk Apache Spark di Amazon EMR pada rilis EKS 6.9.0 atau yang lebih baru, gunakan perintah contoh berikut. Perhatikan bahwa jalur yang tercantum dengan `--conf spark.jars` opsi adalah jalur default untuk file JAR.

```
aws emr-containers start-job-run \

--virtual-cluster-id cluster_id \
--execution-role-arn arn \
--release-label emr-6.9.0-latest\
--job-driver '{
    "sparkSubmitJobDriver": {
        "entryPoint": "s3://script_path",
            "sparkSubmitParameters":
            "--conf spark.kubernetes.file.upload.path=s3://upload_path
             --conf spark.jars=
                /usr/share/aws/redshift/jdbc/RedshiftJDBC.jar,
                /usr/share/aws/redshift/spark-redshift/lib/spark-redshift.jar,
                /usr/share/aws/redshift/spark-redshift/lib/spark-avro.jar,
                /usr/share/aws/redshift/spark-redshift/lib/minimal-json.jar"
                            }
            }'
```

Mengautentikasi dengan integrasi Amazon Redshift untuk Apache Spark

Gunakan AWS Secrets Manager untuk mengambil kredensil dan terhubung ke Amazon Redshift

Anda dapat menyimpan kredensil di Secrets Manager untuk mengautentikasi dengan aman ke Amazon Redshift. Anda dapat meminta pekerjaan Spark memanggil `GetSecretValue` API untuk mengambil kredensialnya:

```
from pyspark.sql import SQLContextimport boto3

sc = # existing SparkContext
sql_context = SQLContext(sc)

secretsmanager_client = boto3.client('secretsmanager',
 region_name=os.getenv('AWS_REGION'))
secret_manager_response = secretsmanager_client.get_secret_value(
    SecretId='string',
    VersionId='string',
    VersionStage='string'
)
username = # get username from secret_manager_response
password = # get password from secret_manager_response
url = "jdbc:redshift://redshifthost:5439/database?user=" + username + "&password="
 + password

# Access to Redshift cluster using Spark
```

Gunakan otentikasi berbasis IAM dengan Amazon EMR pada peran eksekusi pekerjaan EKS

Dimulai dengan Amazon EMR pada rilis EKS 6.9.0, driver Amazon Redshift JDBC versi 2.1 atau lebih tinggi dikemas ke lingkungan. Dengan driver JDBC 2.1 dan yang lebih tinggi, Anda dapat menentukan URL JDBC dan tidak menyertakan nama pengguna dan kata sandi mentah. Sebagai gantinya, Anda dapat menentukan `jdbc:redshift:iam://` skema. Ini memerintahkan driver JDBC untuk menggunakan EMR Amazon Anda pada peran eksekusi pekerjaan EKS untuk mengambil kredensil secara otomatis.

Lihat Mengonfigurasi sambungan JDBC atau ODBC untuk menggunakan kredenal IAM di Panduan Manajemen Amazon Redshift untuk informasi selengkapnya.

Contoh URL berikut menggunakan `jdbc:redshift:iam://` skema.

```
jdbc:redshift:iam://examplecluster.abc123xyz789.us-west-2.redshift.amazonaws.com:5439/
dev
```

Izin berikut diperlukan untuk peran eksekusi pekerjaan Anda saat memenuhi ketentuan yang disediakan.

Izin	Kondisi bila diperlukan untuk peran pelaksanaan pekerjaan
`redshift:GetClusterCredentials`	Diperlukan untuk driver JDBC untuk mengambil kredensional dari Amazon Redshift
`redshift:DescribeCluster`	Diperlukan jika Anda menentukan klaster Amazon Redshift dan Wilayah AWS di URL JDBC, bukan titik akhir
`redshift-serverless:GetCredentials`	Diperlukan untuk driver JDBC untuk mengambil kredensional dari Amazon Redshift Serverless
`redshift-serverless:GetWorkgroup`	Diperlukan jika Anda menggunakan Amazon Redshift Tanpa Server dan Anda menentukan URL dalam hal nama grup kerja dan Wilayah

Kebijakan peran eksekusi pekerjaan Anda harus memiliki izin berikut.

```
{
            "Effect": "Allow",
            "Action": [
                "redshift:GetClusterCredentials",
                "redshift:DescribeCluster",
                "redshift-serverless:GetCredentials",
                "redshift-serverless:GetWorkgroup"
            ],
            "Resource": [

 "arn:aws:redshift:AWS_REGION:ACCOUNT_ID:dbname:CLUSTER_NAME/DATABASE_NAME",
```

```
            "arn:aws:redshift:AWS_REGION:ACCOUNT_ID:dbuser:DATABASE_NAME/USER_NAME"
        ]
    }
```

Otentikasi ke Amazon Redshift dengan driver JDBC

Tetapkan nama pengguna dan kata sandi di dalam URL JDBC

Untuk mengautentikasi pekerjaan Spark ke cluster Amazon Redshift, Anda dapat menentukan nama dan kata sandi database Amazon Redshift di URL JDBC.

> **Note**
>
> Jika Anda meneruskan kredensi database di URL, siapa pun yang memiliki akses ke URL juga dapat mengakses kredensialnya. Metode ini umumnya tidak disarankan karena ini bukan opsi yang aman.

Jika keamanan tidak menjadi perhatian untuk aplikasi Anda, Anda dapat menggunakan format berikut untuk mengatur nama pengguna dan kata sandi di URL JDBC:

```
jdbc:redshift://redshifthost:5439/database?user=username&password=password
```

Membaca dan menulis dari dan ke Amazon Redshift

Contoh kode berikut digunakan PySpark untuk membaca dan menulis data sampel dari dan ke database Amazon Redshift dengan API sumber data dan dengan SparkSQL.

Data source API

Gunakan PySpark untuk membaca dan menulis data sampel dari dan ke database Amazon Redshift dengan API sumber data.

```
import boto3
from pyspark.sql import SQLContext

sc = # existing SparkContext
sql_context = SQLContext(sc)

url = "jdbc:redshift:iam://redshifthost:5439/database"
```

```
aws_iam_role_arn = "arn:aws:iam::accountID:role/roleName"

df = sql_context.read \
    .format("io.github.spark_redshift_community.spark.redshift") \
    .option("url", url) \
    .option("dbtable", "tableName") \
    .option("tempdir", "s3://path/for/temp/data") \
    .option("aws_iam_role", "aws_iam_role_arn") \
    .load()

df.write \
    .format("io.github.spark_redshift_community.spark.redshift") \
    .option("url", url) \
    .option("dbtable", "tableName_copy") \
    .option("tempdir", "s3://path/for/temp/data") \
    .option("aws_iam_role", "aws_iam_role_arn") \
    .mode("error") \
    .save()
```

SparkSQL

Gunakan PySpark untuk membaca dan menulis data sampel dari dan ke database Amazon Redshift menggunakan SparkSQL.

```
import boto3
import json
import sys
import os
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .enableHiveSupport() \
    .getOrCreate()

url = "jdbc:redshift:iam://redshifthost:5439/database"
aws_iam_role_arn = "arn:aws:iam::accountID:role/roleName"

bucket = "s3://path/for/temp/data"
tableName = "tableName" # Redshift table name

s = f"""CREATE TABLE IF NOT EXISTS {tableName} (country string, data string)
    USING io.github.spark_redshift_community.spark.redshift
```

```
    OPTIONS (dbtable '{tableName}', tempdir '{bucket}', url '{url}', aws_iam_role
 '{aws_iam_role_arn}' ); """

spark.sql(s)

columns = ["country" ,"data"]
data = [("test-country","test-data")]
df = spark.sparkContext.parallelize(data).toDF(columns)

# Insert data into table
df.write.insertInto(tableName, overwrite=False)
df = spark.sql(f"SELECT * FROM {tableName}")
df.show()
```

Pertimbangan dan batasan saat menggunakan konektor Spark

- Kami menyarankan Anda mengaktifkan SSL untuk koneksi JDBC dari Spark di Amazon EMR ke Amazon Redshift.
- Kami menyarankan Anda mengelola kredensil untuk cluster Amazon Redshift sebagai praktik AWS Secrets Manager terbaik. Lihat Menggunakan AWS Secrets Manager untuk mengambil kredensil untuk menghubungkan ke Amazon Redshift sebagai contoh.
- Kami menyarankan Anda meneruskan peran IAM dengan parameter `aws_iam_role` untuk parameter autentikasi Amazon Redshift.
- Parameter `tempformat` saat ini tidak mendukung format Parket.
- `tempdir`URI menunjuk ke lokasi Amazon S3. Direktori temp ini tidak dibersihkan secara otomatis dan karenanya dapat menambah biaya tambahan.
- Pertimbangkan rekomendasi berikut untuk Amazon Redshift:
  - Kami menyarankan Anda memblokir akses publik ke cluster Amazon Redshift.
  - Kami menyarankan Anda mengaktifkan pencatatan audit Amazon Redshift.
  - Sebaiknya aktifkan enkripsi saat istirahat Amazon Redshift.
- Pertimbangkan rekomendasi berikut untuk Amazon S3:
  - Kami merekomendasikan untuk memblokir akses publik ke bucket Amazon S3.
  - Kami menyarankan Anda menggunakan enkripsi sisi server Amazon S3 untuk mengenkripsi bucket S3 yang Anda gunakan.

- Sebaiknya gunakan kebijakan siklus hidup Amazon S3 untuk menentukan aturan retensi bucket S3.
- Amazon EMR selalu memverifikasi kode yang diimpor dari sumber terbuka ke dalam gambar. Demi keamanan, kami tidak mendukung kunci AWS akses encoding di `tempdir` URI sebagai metode otentikasi dari Spark ke Amazon S3.

Untuk informasi selengkapnya tentang penggunaan konektor dan parameter yang didukung, lihat sumber daya berikut:

- Integrasi Amazon Redshift untuk Apache Spark di Panduan Manajemen Amazon Redshift
- Repositori `spark-redshift` komunitas di Github

Menggunakan Volcano sebagai penjadwal khusus untuk Apache Spark di Amazon EMR di EKS

Dengan Amazon EMR di EKS, Anda dapat menggunakan operator Spark atau spark-submit untuk menjalankan pekerjaan Spark dengan penjadwal kustom Kubernetes. Tutorial ini mencakup cara menjalankan pekerjaan Spark dengan penjadwal Volcano pada antrian khusus.

Gambaran Umum

Volcano dapat membantu mengelola penjadwalan Spark dengan fungsi-fungsi lanjutan seperti penjadwalan antrian, penjadwalan fair-share, dan reservasi sumber daya. Untuk informasi lebih lanjut tentang manfaat Volcano, lihat Why Spark memilih Volcano sebagai penjadwal batch bawaan pada Kubernetes di blog CNCF The Linux Foundation.

Instal dan atur Volcano

1. Pilih salah satu perintah kubectl berikut untuk menginstal Volcano, tergantung pada kebutuhan arsitektur Anda:

```
# x86_64
kubectl apply -f https://raw.githubusercontent.com/volcano-sh/volcano/v1.5.1/
installer/volcano-development.yaml
# arm64:
kubectl apply -f https://raw.githubusercontent.com/volcano-sh/volcano/v1.5.1/
installer/volcano-development-arm64.yaml
```

2. Siapkan sampel antrian gunung berapi. Antrian adalah kumpulan. PodGroups Antrian mengadopsi FIFO dan merupakan dasar untuk divisi sumber daya.

```
cat << EOF > volcanoQ.yaml
apiVersion: scheduling.volcano.sh/v1beta1
kind: Queue
metadata:
  name: sparkqueue
spec:
  weight: 4
  reclaimable: false
  capability:
    cpu: 10
    memory: 20Gi
EOF

kubectl apply -f volcanoQ.yaml
```

3. Unggah PodGroup manifes sampel ke Amazon S3. PodGroupAdalah sekelompok polong dengan asosiasi yang kuat. Anda biasanya menggunakan penjadwalan PodGroup untuk batch. Kirimkan contoh berikut PodGroup ke antrean yang Anda tentukan di langkah sebelumnya.

```
cat << EOF > podGroup.yaml
apiVersion: scheduling.volcano.sh/v1beta1
kind: PodGroup
spec:
  # Set minMember to 1 to make a driver pod
  minMember: 1
  # Specify minResources to support resource reservation.
  # Consider the driver pod resource and executors pod resource.
  # The available resources should meet the minimum requirements of the Spark job
  # to avoid a situation where drivers are scheduled, but they can't schedule
  # sufficient executors to progress.
  minResources:
    cpu: "1"
    memory: "1Gi"
  # Specify the queue. This defines the resource queue that the job should be
 submitted to.
  queue: sparkqueue
EOF

aws s3 mv podGroup.yaml s3://bucket-name
```

Jalankan aplikasi Spark dengan penjadwal Volcano dengan operator Spark

1. Jika Anda belum melakukannya, selesaikan langkah-langkah di bagian berikut untuk menyiapkan:

   a. Instal dan atur Volcano

   b. Menyiapkan operator Spark untuk Amazon di EMR EKS

   c. Instal operator Spark

   Sertakan argumen berikut saat Anda menjalankan `helm install spark-operator-demo` perintah:

   ```
   --set batchScheduler.enable=true
   --set webhook.enable=true
   ```

2. Buat file `SparkApplication` definisi `spark-pi.yaml` dengan `batchScheduler` dikonfigurasi.

   ```
   apiVersion: "sparkoperator.k8s.io/v1beta2"
   kind: SparkApplication
   metadata:
     name: spark-pi
     namespace: spark-operator
   spec:
     type: Scala
     mode: cluster
     image: "895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.10.0:latest"
     imagePullPolicy: Always
     mainClass: org.apache.spark.examples.SparkPi
     mainApplicationFile: "local:///usr/lib/spark/examples/jars/spark-examples.jar"
     sparkVersion: "3.3.1"
     batchScheduler: "volcano"   #Note: You must specify the batch scheduler name as
    'volcano'
     restartPolicy:
       type: Never
     volumes:
       - name: "test-volume"
         hostPath:
           path: "/tmp"
           type: Directory
     driver:
   ```

```
    cores: 1
    coreLimit: "1200m"
    memory: "512m"
    labels:
      version: 3.3.1
    serviceAccount: emr-containers-sa-spark
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
  executor:
    cores: 1
    instances: 1
    memory: "512m"
    labels:
      version: 3.3.1
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
```

3. Kirimkan aplikasi Spark dengan perintah berikut. Ini juga menciptakan `SparkApplication` objek yang disebut`spark-pi`:

```
kubectl apply -f spark-pi.yaml
```

4. Periksa peristiwa untuk `SparkApplication` objek dengan perintah berikut:

```
kubectl describe pods spark-pi-driver --namespace spark-operator
```

Peristiwa pod pertama akan menunjukkan bahwa Volcano telah menjadwalkan Pod:

```
Type    Reason     Age   From               Message
----    ------     ----  ----               -------
Normal  Scheduled  23s   volcano            Successfully assigned default/spark-
pi-driver to integration-worker2
```

Jalankan aplikasi Spark dengan Volcano scheduler dengan **`spark-submit`**

1. Pertama, selesaikan langkah-langkah di Menyiapkan spark-submit untuk Amazon EMR di EKS bagian ini. Anda harus membangun `spark-submit` distribusi Anda dengan dukungan Volcano.

Untuk informasi selengkapnya, lihat bagian Build dari Using Volcano as Customized Scheduler for Spark on Kubernetes dalam dokumentasi Apache Spark.

2. Tetapkan nilai untuk variabel lingkungan berikut:

```
export SPARK_HOME=spark-home
export MASTER_URL=k8s://Amazon-EKS-cluster-endpoint
```

3. Kirimkan aplikasi Spark dengan perintah berikut:

```
$SPARK_HOME/bin/spark-submit \
 --class org.apache.spark.examples.SparkPi \
 --master $MASTER_URL \
 --conf spark.kubernetes.container.image=895885662937.dkr.ecr.us-
west-2.amazonaws.com/spark/emr-6.10.0:latest \
 --conf spark.kubernetes.authenticate.driver.serviceAccountName=spark \
 --deploy-mode cluster \
 --conf spark.kubernetes.namespace=spark-operator \
 --conf spark.kubernetes.scheduler.name=volcano \
 --conf spark.kubernetes.scheduler.volcano.podGroupTemplateFile=/path/to/podgroup-
template.yaml \
 --conf
 spark.kubernetes.driver.pod.featureSteps=org.apache.spark.deploy.k8s.features.VolcanoFeatu
 \
 --conf
 spark.kubernetes.executor.pod.featureSteps=org.apache.spark.deploy.k8s.features.VolcanoFea
 \
 local:///usr/lib/spark/examples/jars/spark-examples.jar 20
```

4. Periksa peristiwa untuk `SparkApplication` objek dengan perintah berikut:

```
kubectl describe pod spark-pi --namespace spark-operator
```

Peristiwa pod pertama akan menunjukkan bahwa Volcano telah menjadwalkan Pod:

```
Type    Reason     Age   From               Message
----    ------     ----  ----               -------
Normal  Scheduled  23s   volcano            Successfully assigned default/spark-
pi-driver to integration-worker2
```

Menggunakan YuniKorn sebagai penjadwal khusus untuk Apache Spark di Amazon EMR di EKS

Dengan Amazon EMR di EKS, Anda dapat menggunakan operator Spark atau spark-submit untuk menjalankan pekerjaan Spark dengan penjadwal kustom Kubernetes. Tutorial ini mencakup cara menjalankan pekerjaan Spark dengan YuniKorn penjadwal pada antrian kustom dan penjadwalan geng.

Gambaran Umum

Apache YuniKorn dapat membantu mengelola penjadwalan Spark dengan penjadwalan sadar aplikasi sehingga Anda dapat memiliki kontrol yang baik pada kuota dan prioritas sumber daya. Dengan penjadwalan geng, YuniKorn jadwalkan aplikasi hanya jika permintaan sumber daya minimal untuk aplikasi dapat dipenuhi. Untuk informasi lebih lanjut, lihat Apa itu penjadwalan geng di situs YuniKorn dokumentasi Apache.

Buat klaster Anda dan siapkan YuniKorn

Gunakan langkah-langkah berikut untuk menerapkan klaster Amazon EKS. Anda dapat mengubah Wilayah AWS (`region`) dan Availability Zones (`availabilityZones`).

1. Tentukan klaster Amazon EKS:

```
cat <<EOF >eks-cluster.yaml
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: emr-eks-cluster
  region: eu-west-1

vpc:
  clusterEndpoints:
    publicAccess: true
    privateAccess: true

iam:
  withOIDC: true
```

```
nodeGroups:
  - name: spark-jobs
    labels: { app: spark }
    instanceType: m5.xlarge
    desiredCapacity: 2
    minSize: 2
    maxSize: 3
    availabilityZones: ["eu-west-1a"]
EOF
```

2. Buat klaster:

```
eksctl create cluster -f eks-cluster.yaml
```

3. Buat namespace `spark-job` tempat Anda akan mengeksekusi pekerjaan Spark:

```
kubectl create namespace spark-job
```

4. Selanjutnya, buat role dan role binding Kubernetes. Ini diperlukan untuk akun layanan yang digunakan oleh pekerjaan Spark.

   a. Tentukan akun layanan, peran, dan pengikatan peran untuk pekerjaan Spark.

```
cat <<EOF >emr-job-execution-rbac.yaml
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: spark-sa
  namespace: spark-job
automountServiceAccountToken: false
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: spark-role
  namespace: spark-job
rules:
  - apiGroups: ["", "batch","extensions"]
    resources: ["configmaps","serviceaccounts","events","pods","pods/
exec","pods/log","pods/
portforward","secrets","services","persistentvolumeclaims"]
    verbs: ["create","delete","get","list","patch","update","watch"]
```

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: spark-sa-rb
  namespace: spark-job
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: spark-role
subjects:
  - kind: ServiceAccount
    name: spark-sa
    namespace: spark-job
EOF
```

b. Terapkan definisi role dan role binding Kubernetes dengan perintah berikut:

```
kubectl apply -f emr-job-execution-rbac.yaml
```

Instal dan atur YuniKorn

1. Gunakan perintah kubectl berikut untuk membuat namespace untuk men-deploy penjadwal `yunikorn` Yunikorn:

```
kubectl create namespace yunikorn
```

2. Untuk menginstal scheduler, jalankan perintah Helm berikut:

```
helm repo add yunikorn https://apache.github.io/yunikorn-release
```

```
helm repo update
```

```
helm install yunikorn yunikorn/yunikorn --namespace yunikorn
```

Jalankan aplikasi Spark dengan YuniKorn scheduler dengan operator Spark

1. Jika Anda belum melakukannya, selesaikan langkah-langkah di bagian berikut untuk menyiapkan:

   a. Buat klaster Anda dan siapkan YuniKorn

   b. Instal dan atur YuniKorn

   c. Menyiapkan operator Spark untuk Amazon di EMR EKS

   d. Instal operator Spark

   Sertakan argumen berikut saat Anda menjalankan `helm install spark-operator-demo` perintah:

   ```
   --set batchScheduler.enable=true
   --set webhook.enable=true
   ```

2. Buat file `SparkApplication` definisi`spark-pi.yaml`.

   Untuk digunakan YuniKorn sebagai penjadwal untuk pekerjaan Anda, Anda harus menambahkan anotasi dan label tertentu ke definisi aplikasi Anda. Anotasi dan label menentukan antrian untuk pekerjaan Anda dan strategi penjadwalan yang ingin Anda gunakan.

   Pada contoh berikut, anotasi `schedulingPolicyParameters` mengatur penjadwalan geng untuk aplikasi. Kemudian, contoh membuat grup tugas, atau “geng” tugas, untuk menentukan kapasitas minimum yang harus tersedia sebelum menjadwalkan Pod untuk memulai eksekusi pekerjaan. Dan akhirnya, itu menentukan dalam definisi kelompok tugas untuk menggunakan kelompok simpul dengan `"app": "spark"` label, seperti yang didefinisikan dalam Buat klaster Anda dan siapkan YuniKorn bagian.

   ```
   apiVersion: "sparkoperator.k8s.io/v1beta2"
   kind: SparkApplication
   metadata:
     name: spark-pi
     namespace: spark-job
   spec:
     type: Scala
     mode: cluster
     image: "895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.10.0:latest"
     imagePullPolicy: Always
     mainClass: org.apache.spark.examples.SparkPi
   ```

```
  mainApplicationFile: "local:///usr/lib/spark/examples/jars/spark-examples.jar"
  sparkVersion: "3.3.1"
  restartPolicy:
    type: Never
  volumes:
    - name: "test-volume"
      hostPath:
        path: "/tmp"
        type: Directory
  driver:
    cores: 1
    coreLimit: "1200m"
    memory: "512m"
    labels:
      version: 3.3.1
    annotations:
      yunikorn.apache.org/schedulingPolicyParameters: "placeholderTimeoutSeconds=30
gangSchedulingStyle=Hard"
      yunikorn.apache.org/task-group-name: "spark-driver"
      yunikorn.apache.org/task-groups: |-
        [{
            "name": "spark-driver",
            "minMember": 1,
            "minResource": {
              "cpu": "1200m",
              "memory": "1Gi"
            },
            "nodeSelector": {
              "app": "spark"
            }
          },
          {
            "name": "spark-executor",
            "minMember": 1,
            "minResource": {
              "cpu": "1200m",
              "memory": "1Gi"
            },
            "nodeSelector": {
              "app": "spark"
            }
        }]
    serviceAccount: spark-sa
    volumeMounts:
```

```
      - name: "test-volume"
        mountPath: "/tmp"
  executor:
    cores: 1
    instances: 1
    memory: "512m"
    labels:
      version: 3.3.1
    annotations:
      yunikorn.apache.org/task-group-name: "spark-executor"
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
```

3. Kirim aplikasi Spark dengan perintah berikut. Ini juga menciptakan `SparkApplication` objek yang disebut`spark-pi`:

```
kubectl apply -f spark-pi.yaml
```

4. Periksa peristiwa untuk `SparkApplication` objek dengan perintah berikut:

```
kubectl describe sparkapplication spark-pi --namespace spark-job
```

Peristiwa Pod pertama akan menunjukkan bahwa YuniKorn telah menjadwalkan Pod:

```
Type    Reason            Age   From                         Message
----    ------            ----  ----                         -------
Normal Scheduling        3m12s yunikorn   spark-operator/org-apache-spark-examples-
sparkpi-2a777a88b98b8a95-driver is queued and waiting for allocation
Normal GangScheduling    3m12s yunikorn   Pod belongs to the taskGroup spark-
driver, it will be scheduled as a gang member
Normal Scheduled         3m10s yunikorn   Successfully assigned spark
Normal PodBindSuccessful 3m10s yunikorn   Pod spark-operator/
Normal TaskCompleted     2m3s  yunikorn   Task spark-operator/
Normal Pulling           3m10s kubelet    Pulling
```

Jalankan aplikasi Spark dengan YuniKorn scheduler dengan `spark-submit`

1. Pertama, selesaikan langkah-langkah di Menyiapkan spark-submit untuk Amazon EMR di EKS bagian ini.
2. Tetapkan nilai untuk variabel lingkungan berikut:

```
export SPARK_HOME=spark-home
export MASTER_URL=k8s://Amazon-EKS-cluster-endpoint
```

3. Kirimkan aplikasi Spark dengan perintah berikut:

   Pada contoh berikut, anotasi `schedulingPolicyParameters` mengatur penjadwalan geng untuk aplikasi. Kemudian, contoh membuat grup tugas, atau “geng” tugas, untuk menentukan kapasitas minimum yang harus tersedia sebelum menjadwalkan Pod untuk memulai eksekusi pekerjaan. Dan akhirnya, itu menentukan dalam definisi kelompok tugas untuk menggunakan kelompok simpul dengan `"app": "spark"` label, seperti yang didefinisikan dalam Buat klaster Anda dan siapkan YuniKorn bagian.

```
$SPARK_HOME/bin/spark-submit \
 --class org.apache.spark.examples.SparkPi \
 --master $MASTER_URL \
 --conf spark.kubernetes.container.image=895885662937.dkr.ecr.us-
west-2.amazonaws.com/spark/emr-6.10.0:latest \
 --conf spark.kubernetes.authenticate.driver.serviceAccountName=spark-sa \
 --deploy-mode cluster \
 --conf spark.kubernetes.namespace=spark-job \
 --conf spark.kubernetes.scheduler.name=yunikorn \
 --conf spark.kubernetes.driver.annotation.yunikorn.apache.org/
schedulingPolicyParameters="placeholderTimeoutSeconds=30 gangSchedulingStyle=Hard"
 \
 --conf spark.kubernetes.driver.annotation.yunikorn.apache.org/task-group-
name="spark-driver" \
 --conf spark.kubernetes.executor.annotation.yunikorn.apache.org/task-group-
name="spark-executor" \
 --conf spark.kubernetes.driver.annotation.yunikorn.apache.org/task-groups='[{
            "name": "spark-driver",
            "minMember": 1,
            "minResource": {
              "cpu": "1200m",
              "memory": "1Gi"
```

```
            },
            "nodeSelector": {
              "app": "spark"
            }
          },
          {
            "name": "spark-executor",
            "minMember": 1,
            "minResource": {
              "cpu": "1200m",
              "memory": "1Gi"
            },
            "nodeSelector": {
              "app": "spark"
            }
        }]' \
 local:///usr/lib/spark/examples/jars/spark-examples.jar 20
```

4. Periksa peristiwa untuk `SparkApplication` objek dengan perintah berikut:

```
kubectl describe pod spark-driver-pod --namespace spark-job
```

Peristiwa Pod pertama akan menunjukkan bahwa YuniKorn telah menjadwalkan Pod:

```
Type    Reason           Age   From                          Message
----    ------           ----  ----                          -------
Normal Scheduling        3m12s yunikorn   spark-operator/org-apache-spark-examples-
sparkpi-2a777a88b98b8a95-driver is queued and waiting for allocation
Normal GangScheduling    3m12s yunikorn   Pod belongs to the taskGroup spark-
driver, it will be scheduled as a gang member
Normal Scheduled         3m10s yunikorn   Successfully assigned spark
Normal PodBindSuccessful 3m10s yunikorn   Pod spark-operator/
Normal TaskCompleted     2m3s  yunikorn   Task spark-operator/
Normal Pulling           3m10s kubelet    Pulling
```

Keamanan di Amazon EMR di EKS

Keamanan cloud di AWS merupakan prioritas tertinggi. Sebagai pelanggan AWS, Anda mendapatkan manfaat dari pusat data dan arsitektur jaringan yang dibangun untuk memenuhi persyaratan organisasi yang paling sensitif terhadap keamanan.

Keamanan adalah tanggung jawab bersama antara AWS dan Anda. Model tanggung jawab bersama menggambarkan ini sebagai keamanan dari cloud dan keamanan di dalam cloud:

- Keamanan cloud – AWS bertanggung jawab untuk melindungi infrastruktur yang menjalankan layanan–layanan AWS di dalam AWS Cloud. AWS juga memberikan Anda layanan yang dapat digunakan dengan aman. Auditor pihak ketiga menguji dan memverifikasi secara berkala efektivitas keamanan kami sebagai bagian dari AWSProgram Kepatuhan. Untuk mempelajari tentang program kepatuhan yang berlaku untuk Amazon EMR, lihat AWSLayanan dalam Lingkup berdasarkan AWS Layanan Program Kepatuhan dalam Lingkup oleh Program .
- Keamanan dalam cloud – Tanggung jawab Anda ditentukan oleh layanan AWS yang Anda gunakan. Anda juga bertanggung jawab atas faktor lain termasuk sensitivitas data Anda, persyaratan perusahaan Anda, serta hukum dan peraturan yang berlaku.

Dokumentasi ini membantu Anda memahami cara menerapkan model tanggung jawab bersama saat menggunakan Amazon EMR di EKS. Topik berikut menunjukkan kepada Anda cara mengonfigurasi Amazon EMR di EKS untuk memenuhi tujuan keamanan dan kepatuhan Anda. Anda juga akan mempelajari cara menggunakan layanan AWS lain yang dapat membantu Anda memantau dan mengamankan sumber daya Amazon EMR di EKS Anda.

Topik

- Analisis konfigurasi dan kerentanan
- Connect ke Amazon EMR di EKS Menggunakan VPC endpoints
- Atur akses lintas akun untuk Amazon EMR di EKS

Praktik terbaik keamanan Amazon EMR di EKS

Amazon EMR di EKS menyediakan sejumlah fitur keamanan untuk dipertimbangkan ketika Anda mengembangkan dan menerapkan kebijakan keamanan Anda sendiri. Praktik terbaik berikut adalah pedoman umum dan tidak mewakili solusi keamanan yang lengkap. Karena praktik terbaik ini mungkin tidak sesuai atau cukup untuk lingkungan Anda, anggap praktik terbaik tersebut sebagai pertimbangan yang membantu dan bukan sebagai rekomendasi.

> **Note**
>
> Untuk praktik terbaik keamanan AmazonPraktik terbaik keamanan Amazon EMR di EKS.

Terapkan prinsip hak istimewa paling rendah

Amazon EMR di EKS menyediakan kebijakan akses granular untuk aplikasi menggunakan IAM role, seperti peran eksekusi. Peran eksekusi ini dipetakan ke akun layanan Kubernetes melalui kebijakan kepercayaan IAM role. Amazon EMR di EKS menciptakan pod dalam namespace Amazon EKS terdaftar yang mengeksekusi kode aplikasi yang disediakan pengguna. Pod tugas yang menjalankan kode aplikasi mengasumsikan peran eksekusi ketika menghubungkan ke layanan AWS lain. Kami merekomendasikan bahwa peran eksekusi diberikan hanya seperangkat minimum hak istimewa yang diperlukan oleh tugas, seperti mencakup aplikasi Anda dan akses ke tujuan log. Kami juga merekomendasikan mengaudit tugas untuk izin secara teratur dan pada setiap perubahan pada kode aplikasi.

Daftar kontrol akses untuk titik akhir

Titik akhir terkelola dapat dibuat hanya untuk klaster EKS yang telah dikonfigurasi untuk menggunakan setidaknya satu subnet privat di VPC Anda. Konfigurasi ini membatasi akses ke penyeimbang beban yang dibuat oleh titik akhir terkelola sehingga mereka hanya dapat diakses dari VPC Anda. Untuk lebih meningkatkan keamanan, kami sarankan Anda mengonfigurasi grup keamanan dengan penyeimbang beban ini sehingga mereka dapat membatasi lalu lintas masuk ke serangkaian alamat IP yang dipilih.

Dapatkan pembaruan keamanan terbaru untuk gambar kustom

Untuk menggunakan gambar kustom dengan Amazon EMR di EKS, Anda dapat menginstal biner dan perpustakaan pada gambar. Anda bertanggung jawab untuk patch keamanan biner yang Anda tambahkan ke gambar. Gambar Amazon EMR di EKS secara teratur di-patch dengan patch keamanan terbaru. Untuk mendapatkan gambar terbaru, Anda harus membangun kembali gambar kustom setiap kali ada versi gambar dasar baru dari rilis Amazon EMR. Untuk informasi lebih lanjut, lihat Amazon EMR pada EKS rilis dan Cara memilih gambar dasar URI.

Batasi akses kredensial pod

Kubernetes mendukung beberapa metode penetapan kredensial ke pod. Penyediaan beberapa penyedia kredensial dapat meningkatkan kompleksitas model keamanan Anda. Amazon EMR di EKS telah mengadopsi penggunaan IAM role untuk akun layanan (IRSA) sebagai penyedia kredensial standar dalam namespace EKS terdaftar. Metode lain tidak didukung, termasuk kube2iam, kiam dan menggunakan profil instans EC2 dari instans yang berjalan di klaster.

Mengisolasi kode aplikasi yang tidak tepercaya

Amazon EMR di EKS tidak memeriksa integritas kode aplikasi yang dikirimkan oleh pengguna sistem. Jika Anda menjalankan klaster virtual multi-tenanted yang dikonfigurasi menggunakan beberapa peran eksekusi yang dapat digunakan untuk mengirimkan tugas oleh penyewa tidak tepercaya yang menjalankan kode arbitrer, ada risiko aplikasi berbahaya meningkatkan hak istimewanya. Dalam situasi ini, pertimbangkan untuk mengisolasi peran eksekusi dengan hak istimewa yang sama ke klaster virtual yang berbeda.

Izin kontrol akses berbasis peran (RBAC)

Administrator harus mengontrol ketat izin kontrol akses berbasis peran (RBAC) untuk namespace terkelola Amazon EMR di EKS. Minimal, izin berikut tidak boleh diberikan kepada pengirim tugas di namespace terkelola Amazon EMR di EKS.

- Izin Kubernetes RBAC untuk memodifikasi configmap - karena Amazon EMR di EKS menggunakan configmap Kubernetes untuk menghasilkan templat pod terkelola yang memiliki nama akun layanan terkelola. Atribut ini seharusnya tidak bermutasi.
- Izin Kubernetes RBAC untuk exec ke pod Amazon EMR di EKS - untuk menghindari memberikan akses ke templat pod terkelola yang memiliki nama SA terkelola. Atribut ini seharusnya tidak bermutasi. Izin ini juga dapat memberikan akses ke token JWT dipasang ke pod yang kemudian dapat digunakan untuk mengambil kredensial peran eksekusi.

- Izin Kubernetes RBAC untuk membuat pod untuk mencegah pengguna membuat pod menggunakan Kubernetes ServiceAccount yang dapat dipetakan ke IAM role dengan keistimewaan lebih banyak dari pengguna. AWS
- Izin Kubernetes RBAC untuk menyebarkan webhook bermutasi - untuk mencegah pengguna menggunakan webhook bermutasi untuk memutasikan nama Kubernetes untuk pod yang dibuat oleh Amazon EMR di EKS. ServiceAccount
- Izin RBAC Kubernetes untuk membaca rahasia Kubernetes - untuk mencegah pengguna membaca data rahasia yang tersimpan dalam rahasia ini.

Membatasi akses ke nodegroup IAM role atau kredensial profil instans

- Kami menyarankan agar Anda menetapkan izin AWS minimum untuk nodegroup IAM role. Ini membantu untuk menghindari eskalasi hak istimewa dengan kode yang dapat dijalankan menggunakan kredensial profil instans dari simpul pekerja EKS.
- Untuk memblokir akses sepenuhnya ke kredensial profil isntans untuk semua pod yang berjalan di namespace terkelola Amazon EMR di EKS, kami sarankan Anda menjalankan perintah `iptables` pada simpul EKS. Untuk informasi lebih lanjut, lihat Membatasi akses ke kredensial profil instans Amazon EC2. Namun, penting untuk mencakup dengan benar IAM role akun layanan Anda sehingga pod Anda memiliki semua izin yang diperlukan. Sebagai contoh, IAM role simpul diberikan izin untuk menarik gambar kontainer dari Amazon ECR. Jika pod tidak ditugaskan izin tersebut, pod tidak dapat menarik gambar kontainer dari Amazon ECR. Plugin VPC CNI juga perlu diperbarui. Untuk informasi lebih lanjut, lihat Panduan: Memperbarui plugin VPC CNI untuk menggunakan IAM role untuk akun layanan.

Perlindungan data

AWS model tanggung jawab bersama diterapkan untuk perlindungan data di Amazon EMR di EKS. Sebagaimana diuraikan dalam model ini, AWS bertanggung jawab untuk melindungi infrastruktur global yang menjalankan semua CloudAWS. Anda bertanggung jawab untuk memelihara kontrol terhadap konten Anda yang di-hosting pada infrastruktur ini. Konten ini meliputi konfigurasi keamanan dan tugas manajemen untuk berbagai layanan AWS yang Anda gunakan. Untuk informasi selengkapnya tentang privasi data, lihat FAQ Privasi Data . Untuk informasi tentang perlindungan data di Eropa, lihat postingan blog Model Tanggung Jawab Bersama dan GDPR AWS di Blog Keamanan AWS.

Untuk tujuan perlindungan data, kami menyarankan agar Anda melindungi kredensyalAWS akun dan mengatur akun individu denganAWS Identity and Access Management (IAM). Dengan cara seperti itu, 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 melakukan komunikasi dengan sumber daya AWS. Kami merekomendasikan TLS 1.2 atau versi yang lebih baru.
- Siapkan API dan log aktivitas pengguna dengan AWS CloudTrail.
- Gunakan solusi enkripsi AWS, bersama dengan semua kontrol keamanan default dalam layanan AWS.
- Gunakan layanan keamanan terkelola tingkat lanjut seperti Amazon Macie, yang membantu menemukan dan mengamankan data pribadi yang disimpan di Amazon S3.
- Gunakan opsi enkripsi Amazon EMR di EKS pada opsi enkripsi untuk mengenkripsi data at rest dan transit.
- Jika Anda memerlukan modul kriptografi tervalidasi FIPS 140-2 ketika mengakses AWS melalui antarmuka baris perintah atau API, gunakan titik akhir FIPS. Untuk informasi lebih lanjut tentang titik akhir FIPS yang tersedia, lihat Standar Pemrosesan Informasi Federal (FIPS) 140-2.

Kami sangat merekomendasikan agar Anda tidak memasukkan informasi identifikasi sensitif apapun, seperti nomor rekening pelanggan Anda, ke dalam kolom isian teks bebas seperti kolom Nama. Hal ini termasuk ketika Anda bekerja dengan Amazon EMR di EKS atau layanan AWS lainnya dengan menggunakan konsol, API, AWS CLI, atau SDK AWS. Data apa pun yang Anda masukkan ke dalam Amazon EMR di EKS atau layanan lain mungkin akan diambil untuk dimasukkan ke dalam log diagnostik. Saat Anda menyediakan URL ke peladen eksternal, jangan menyertakan informasi kredensial dalam URL untuk memvalidasi permintaan Anda ke peladen tersebut.

Enkripsi saat istirahat

Enkripsi data membantu mencegah pengguna yang tidak sah membaca data pada klaster 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 dan sertifikat dari penyedia kustom yang Anda suplai. Saat menggunakan AWS KMS sebagai penyedia kunci Anda, biaya berlaku untuk penyimpanan dan penggunaan kunci enkripsi. Untuk informasi lebih lanjut, lihat AWS KMS Harga.

Sebelum Anda menentukan opsi enkripsi, tentukan sistem manajemen kunci dan sertifikat yang ingin Anda gunakan. Kemudian buat kunci dan sertifikat untuk penyedia kustom yang Anda tentukan sebagai bagian dari pengaturan enkripsi.

Enkripsi saat istirahat untuk data EMRFS di Amazon S3

Enkripsi Amazon S3 bekerja dengan objek EMR File System (EMRFS) yang dibaca dari dan ditulis ke Amazon S3. Anda menentukan Amazon S3 server-side encryption (SSE) atau client-side encryption (CSE) sebagai Mode enkripsi default saat Anda mengaktifkan enkripsi saat istirahat. Secara opsional, Anda dapat menentukan metode enkripsi yang berbeda untuk setiap bucket menggunakan Per penimpaan enkripsi bucket. Keamanan Lapisan Pengangkutan (TLS) terlepas dari apakah enkripsi Amazon S3 diaktifkan, Keamanan Lapisan Pengangkutan (TLS) mengenkripsi objek EMRFS dalam transit antara simpul klaster EMR dan Amazon S3. Untuk informasi selengkapnya tentang enkripsi Amazon S3, lihat Melindungi Data Menggunakan Enkripsi di Panduan Developer Amazon Simple Storage Service.

> **Note**
>
> Saat Anda menggunakan AWS KMS, biaya berlaku untuk penyimpanan dan penggunaan kunci enkripsi. Untuk informasi lebih lanjut, lihat AWS KMS Harga.

Enkripsi sisi server Amazon S3

Saat Anda mengatur enkripsi sisi server Amazon S3, Amazon S3 akan mengenkripsi data pada tingkat objek saat menulis data ke disk dan mendekripsi data saat diakses. Untuk informasi selengkapnya, lihat Melindungi Data Menggunakan Enkripsi Sisi Server di Panduan Developer Amazon Simple Storage Service.

Anda dapat memilih antara dua sistem pengelolaan kunci yang berbeda ketika Anda menentukan SSE di Amazon EMR di EKS:

- SSE-S3 - Amazon S3 mengelola kunci untuk Anda.

- SSE-KMS - Anda menggunakan sebuahAWS KMS key untuk mengatur kebijakan yang cocok untuk Amazon EMR di EKS.

SSE dengan kunci yang disediakan pelanggan (SSE-C) tidak tersedia untuk digunakan dengan Amazon EMR di EKS.

Enkripsi di sisi klien Amazon S3

Dengan enkripsi sisi klien Amazon S3, enkripsi dan dekripsi Amazon S3 dilakukan di klien EMRFS pada klaster Anda. Objek dienkripsi sebelum diunggah ke Amazon S3 dan didekripsi setelah diunduh. Penyedia yang Anda tentukan menyediakan kunci enkripsi yang digunakan klien. Klien dapat menggunakan kunci yang disediakan melaluiAWS KMS (CSE-KMS) atau kelas Java khusus yang menyediakan kunci utama sisi klien (CSE-C). Spesifikasi enkripsi sedikit berbeda antara CSE-KMS dan CSE-C, tergantung pada penyedia yang ditentukan dan metadata objek yang didekripsi atau dienkripsi. Untuk informasi selengkapnya, tentang perbedaan ini, lihat Melindungi Data Menggunakan Enkripsi Di Sisi Klien di Panduan Developer Amazon Simple Storage Service.

> **Note**
>
> Amazon S3 CSE hanya memastikan bahwa data EMRFS yang dipertukarkan dengan Amazon S3 dienkripsi; tidak semua data pada volume instans klaster dienkripsi. Selain itu, karena Hue tidak menggunakan EMRFS, objek yang Hue S3 File Browser tulis ke Amazon S3 tidak dienkripsi.

Enkripsi disk lokal

Apache Spark mendukung mengenkripsi data sementara ditulis ke disk lokal. Ini mencakup file acak, spill acak, dan blok data yang disimpan pada disk untuk variabel baik caching maupun siaran. Ini tidak mencakup enkripsi data output yang dihasilkan oleh aplikasi dengan API seperti `saveAsHadoopFile` atau `saveAsTable`. Ini juga mungkin tidak mencakup file sementara yang dibuat secara eksplisit oleh pengguna. Untuk informasi selengkapnya, lihat Enkripsi Penyimpanan Lokal dalam dokumentasi Spark. Spark tidak mendukung data terenkripsi pada disk lokal, seperti data menengah yang ditulis ke disk lokal oleh proses eksekutor ketika data tidak cocok dalam memori. Data yang bertahan ke disk dicakup untuk waktu aktif tugas, dan kunci yang digunakan untuk mengenkripsi data dihasilkan secara dinamis oleh Spark untuk setiap tugas berjalan. Setelah pekerjaan Spark berakhir, tidak ada proses lain yang dapat mendekripsi data.

Untuk driver dan pod eksekutor, Anda mengenkripsi data at rest yang bertahan untuk volume terpasang. Ada tiga perbedaan pilihan penyimpanan asli AWS yang dapat Anda gunakan dengan Kubernetes: EBS, EFS, dan FSx for Lustre. Ketiganya menawarkan enkripsi at rest menggunakan kunci terkelola layanan atau kunci terkelola layanan atauAWS KMS key. Untuk informasi selengkapnya lihat Panduan Praktik Terbaik EKS. Dengan pendekatan ini, semua data yang disimpan ke volume terpasang dienkripsi.

Manajemen kunci

Anda dapat mengonfigurasi KMS untuk secara otomatis merotasi kunci KMS Anda. Ini merotasi kunci Anda setahun sekali sambil menyimpan kunci lama tanpa batas waktu sehingga data Anda masih dapat didekripsi. Untuk informasi tambahan, lihat MerotasiAWS KMS keys.

Enkripsi dalam transit

Beberapa mekanisme enkripsi diaktifkan dengan enkripsi dalam transit. Ini adalah fitur sumber daya terbuka, khusus aplikasi, dan dapat bervariasi dengan rilis Amazon EMR di EKS. Fitur enkripsi khusus aplikasi berikut dapat diaktifkan dengan Amazon EMR di EKS:

- Spark
  - Komunikasi RPC internal antara komponen Spark, seperti layanan transfer blok dan layanan shuffle eksternal, dienkripsi menggunakan cipher AES-256 di Amazon EMR versi 5.9.0 dan versi terbaru. Di rilis sebelumnya, komunikasi RPC internal dienkripsi menggunakan SASL dengan DIGEST-MD5 sebagai cipher.
  - Komunikasi protokol HTTP dengan antarmuka pengguna seperti Spark History Server dan server file HTTPS-enabled dienkripsi menggunakan konfigurasi SSL Spark. Untuk informasi lebih lanjut, lihat Konfigurasi SSL di dokumentasi Spark.

  Untuk informasi lebih lanjut, lihat Pengaturan keamanan Spark.
- Anda sebaiknya hanya mengizinkan koneksi terenkripsi melalui HTTPS (TLS) menggunakan SecureTransport kondisi aws: di kebijakan IAM bucket Amazon S3.
- Hasil kueri yang di-stream ke klien JDBC atau ODBC dienkripsi menggunakan TLS.

Manajemen Identitas dan Akses

AWS Identity and Access Management (IAM) adalah AWS layanan yang membantu administrator mengontrol akses ke AWS sumber daya dengan aman. IAMadministrator mengontrol siapa yang

dapat diautentikasi (masuk) dan diberi wewenang (memiliki izin) untuk menggunakan Amazon EMR pada sumber daya. EKS IAMadalah AWS layanan yang dapat Anda gunakan tanpa biaya tambahan.

Topik

Audiens

Cara Anda menggunakan AWS Identity and Access Management (IAM) berbeda, tergantung pada pekerjaan yang Anda lakukan EMR di AmazonEKS.

Pengguna layanan — Jika Anda menggunakan EKS layanan EMR Amazon untuk melakukan pekerjaan Anda, administrator Anda memberi Anda kredensi dan izin yang Anda butuhkan. Saat Anda menggunakan lebih banyak EKS fitur EMR Amazon untuk melakukan pekerjaan Anda, Anda mungkin memerlukan izin tambahan. Memahami cara akses dikelola dapat membantu Anda meminta izin yang tepat dari administrator Anda. Jika Anda tidak dapat mengakses fitur EMR di AmazonEKS, lihatMemecahkan masalah Amazon EMR tentang EKS identitas dan akses.

Administrator layanan - Jika Anda bertanggung jawab atas Amazon EMR pada EKS sumber daya di perusahaan Anda, Anda mungkin memiliki akses penuh ke EMR AmazonEKS. Tugas Anda adalah menentukan Amazon mana EMR pada EKS fitur dan sumber daya yang harus diakses pengguna layanan Anda. Anda kemudian harus mengirimkan permintaan ke IAM administrator Anda untuk mengubah izin pengguna layanan Anda. Tinjau informasi di halaman ini untuk memahami konsep dasarIAM. Untuk mempelajari lebih lanjut tentang bagaimana perusahaan Anda dapat menggunakan IAM Amazon EMR diEKS, lihatBagaimana EMR Amazon EKS bekerja dengan IAM.

IAMadministrator - Jika Anda seorang IAM administrator, Anda mungkin ingin mempelajari detail tentang cara menulis kebijakan untuk mengelola akses ke EMR AmazonEKS. Untuk melihat contoh Amazon EMR tentang kebijakan EKS berbasis identitas yang dapat Anda gunakan, lihat. IAM Contoh kebijakan berbasis identitas untuk Amazon di EMR EKS

Mengautentikasi dengan identitas

Otentikasi adalah cara Anda masuk AWS menggunakan kredensi identitas Anda. Anda harus diautentikasi (masuk ke AWS) sebagai Pengguna root akun AWS, sebagai IAM pengguna, atau dengan mengambil peranIAM.

Anda dapat masuk AWS sebagai identitas federasi dengan menggunakan kredensi yang disediakan melalui sumber identitas. AWS IAM Identity Center Pengguna (Pusat IAM Identitas), autentikasi masuk tunggal perusahaan Anda, dan kredensi Google atau Facebook Anda adalah contoh identitas federasi. Saat Anda masuk sebagai identitas federasi, administrator Anda sebelumnya menyiapkan federasi identitas menggunakan IAM peran. Ketika Anda mengakses AWS dengan menggunakan federasi, Anda secara tidak langsung mengambil peran.

Bergantung pada jenis pengguna Anda, Anda dapat masuk ke AWS Management Console atau portal AWS akses. Untuk informasi selengkapnya tentang masuk AWS, lihat Cara masuk ke Panduan AWS Sign-In Pengguna Anda Akun AWS.

Jika Anda mengakses AWS secara terprogram, AWS sediakan kit pengembangan perangkat lunak (SDK) dan antarmuka baris perintah (CLI) untuk menandatangani permintaan Anda secara kriptografis dengan menggunakan kredensil Anda. Jika Anda tidak menggunakan AWS alat, Anda harus menandatangani permintaan sendiri. Untuk informasi selengkapnya tentang menggunakan metode yang disarankan untuk menandatangani permintaan sendiri, lihat Menandatangani AWS API permintaan di Panduan IAM Pengguna.

Apa pun metode autentikasi yang digunakan, Anda mungkin diminta untuk menyediakan informasi keamanan tambahan. Misalnya, AWS merekomendasikan agar Anda menggunakan otentikasi multi-faktor (MFA) untuk meningkatkan keamanan akun Anda. Untuk mempelajari lebih lanjut, lihat Autentikasi multi-faktor di Panduan AWS IAM Identity Center Pengguna dan Menggunakan otentikasi multi-faktor (MFA) AWS di Panduan Pengguna. IAM

Akun AWS pengguna root

Saat Anda membuat Akun AWS, Anda mulai dengan satu identitas masuk yang memiliki akses lengkap ke semua AWS layanan dan sumber daya di akun. Identitas ini disebut pengguna Akun AWS root dan diakses dengan masuk dengan alamat email dan kata sandi yang Anda gunakan

untuk membuat akun. Kami sangat menyarankan agar Anda tidak menggunakan pengguna root untuk tugas sehari-hari. Lindungi kredensial pengguna root Anda dan gunakan kredensial tersebut untuk melakukan tugas yang hanya dapat dilakukan pengguna root. Untuk daftar lengkap tugas yang mengharuskan Anda masuk sebagai pengguna root, lihat Tugas yang memerlukan kredensi pengguna root di IAMPanduan Pengguna.

Identitas gabungan

Sebagai praktik terbaik, mewajibkan pengguna manusia, termasuk pengguna yang memerlukan akses administrator, untuk menggunakan federasi dengan penyedia identitas untuk mengakses AWS layanan dengan menggunakan kredensi sementara.

Identitas federasi adalah pengguna dari direktori pengguna perusahaan Anda, penyedia identitas web, direktori Pusat Identitas AWS Directory Service, atau pengguna mana pun yang mengakses AWS layanan dengan menggunakan kredensi yang disediakan melalui sumber identitas. Ketika identitas federasi mengakses Akun AWS, mereka mengambil peran, dan peran memberikan kredensi sementara.

Untuk manajemen akses terpusat, kami sarankan Anda menggunakan AWS IAM Identity Center. Anda dapat membuat pengguna dan grup di Pusat IAM Identitas, atau Anda dapat menghubungkan dan menyinkronkan ke sekumpulan pengguna dan grup di sumber identitas Anda sendiri untuk digunakan di semua aplikasi Akun AWS dan aplikasi Anda. Untuk informasi tentang Pusat IAM Identitas, lihat Apa itu Pusat IAM Identitas? dalam AWS IAM Identity Center User Guide.

Pengguna dan grup IAM

IAMPengguna adalah identitas dalam diri Anda Akun AWS yang memiliki izin khusus untuk satu orang atau aplikasi. Jika memungkinkan, sebaiknya mengandalkan kredensi sementara daripada membuat IAM pengguna yang memiliki kredensi jangka panjang seperti kata sandi dan kunci akses. Namun, jika Anda memiliki kasus penggunaan khusus yang memerlukan kredensi jangka panjang dengan IAM pengguna, kami sarankan Anda memutar kunci akses. Untuk informasi selengkapnya, lihat Memutar kunci akses secara teratur untuk kasus penggunaan yang memerlukan kredensi jangka panjang di IAMPanduan Pengguna.

IAMGrup adalah identitas yang menentukan kumpulan IAM pengguna. Anda tidak dapat masuk sebagai grup. Anda dapat menggunakan grup untuk menentukan izin bagi beberapa pengguna sekaligus. Grup mempermudah manajemen izin untuk sejumlah besar pengguna sekaligus. Misalnya, Anda dapat memiliki grup bernama IAMAdminsdan memberikan izin grup tersebut untuk mengelola sumber dayaIAM.

Pengguna berbeda dari peran. Pengguna secara unik terkait dengan satu orang atau aplikasi, tetapi peran dimaksudkan untuk dapat digunakan oleh siapa pun yang membutuhkannya. Pengguna memiliki kredensial jangka panjang permanen, tetapi peran memberikan kredensial sementara. Untuk mempelajari lebih lanjut, lihat Kapan membuat IAM pengguna (bukan peran) di Panduan IAM Pengguna.

IAMperan

IAMPeran adalah identitas dalam diri Anda Akun AWS yang memiliki izin khusus. Ini mirip dengan IAM pengguna, tetapi tidak terkait dengan orang tertentu. Anda dapat mengambil IAM peran sementara AWS Management Console dengan beralih peran. Anda dapat mengambil peran dengan memanggil AWS CLI atau AWS API operasi atau dengan menggunakan kustomURL. Untuk informasi selengkapnya tentang metode penggunaan peran, lihat Menggunakan IAM peran di Panduan IAM Pengguna.

IAMperan dengan kredensi sementara berguna dalam situasi berikut:

- Akses pengguna terfederasi – Untuk menetapkan izin ke identitas terfederasi, Anda membuat peran dan menentukan izin untuk peran tersebut. Ketika identitas terfederasi mengautentikasi, identitas tersebut terhubung dengan peran dan diberi izin yang ditentukan oleh peran. Untuk informasi tentang peran untuk federasi, lihat Membuat peran untuk Penyedia Identitas pihak ketiga di Panduan IAM Pengguna. Jika Anda menggunakan Pusat IAM Identitas, Anda mengonfigurasi set izin. Untuk mengontrol apa yang dapat diakses identitas Anda setelah diautentikasi, Pusat IAM Identitas menghubungkan izin yang disetel ke peran. IAM Untuk informasi tentang set izin, lihat Set izin dalam Panduan Pengguna AWS IAM Identity Center .
- Izin IAM pengguna sementara — IAM Pengguna atau peran dapat mengambil IAM peran untuk sementara mengambil izin yang berbeda untuk tugas tertentu.
- Akses lintas akun — Anda dapat menggunakan IAM peran untuk memungkinkan seseorang (prinsipal tepercaya) di akun lain mengakses sumber daya di akun Anda. Peran adalah cara utama untuk memberikan akses lintas akun. Namun, dengan beberapa AWS layanan, Anda dapat melampirkan kebijakan secara langsung ke sumber daya (alih-alih menggunakan peran sebagai proxy). Untuk mempelajari perbedaan antara peran dan kebijakan berbasis sumber daya untuk akses lintas akun, lihat Akses sumber daya lintas akun di IAM Panduan Pengguna. IAM
- Akses lintas layanan — Beberapa AWS layanan menggunakan fitur lain AWS layanan. Misalnya, saat Anda melakukan panggilan dalam suatu layanan, biasanya layanan tersebut menjalankan aplikasi di Amazon EC2 atau menyimpan objek di Amazon S3. Sebuah layanan mungkin

melakukannya menggunakan izin prinsipal yang memanggil, menggunakan peran layanan, atau peran terkait layanan.

- Sesi akses teruskan (FAS) — Saat Anda menggunakan IAM pengguna atau peran untuk melakukan tindakan AWS, Anda dianggap sebagai prinsipal. Ketika Anda menggunakan beberapa layanan, Anda mungkin melakukan sebuah tindakan yang kemudian menginisiasi tindakan lain di layanan yang berbeda. FASmenggunakan izin dari pemanggilan utama AWS layanan, dikombinasikan dengan permintaan AWS layanan untuk membuat permintaan ke layanan hilir. FASPermintaan hanya dibuat ketika layanan menerima permintaan yang memerlukan interaksi dengan orang lain AWS layanan atau sumber daya untuk menyelesaikannya. Dalam hal ini, Anda harus memiliki izin untuk melakukan kedua tindakan tersebut. Untuk detail kebijakan saat membuat FAS permintaan, lihat Meneruskan sesi akses.
- Peran layanan — Peran layanan adalah IAMperan yang diasumsikan layanan untuk melakukan tindakan atas nama Anda. IAMAdministrator dapat membuat, memodifikasi, dan menghapus peran layanan dari dalamIAM. Untuk informasi selengkapnya, lihat Membuat peran untuk mendelegasikan izin ke AWS layanan dalam IAMPanduan Pengguna.
- Peran terkait layanan — Peran terkait layanan adalah jenis peran layanan yang ditautkan ke peran layanan. AWS layanan Layanan tersebut dapat menjalankan peran untuk melakukan tindakan atas nama Anda. Peran terkait layanan muncul di Anda Akun AWS dan dimiliki oleh layanan. IAMAdministrator dapat melihat, tetapi tidak mengedit izin untuk peran terkait layanan.

- Aplikasi yang berjalan di Amazon EC2 — Anda dapat menggunakan IAM peran untuk mengelola kredenal sementara untuk aplikasi yang berjalan pada EC2 instance dan membuat AWS CLI atau AWS API meminta. Ini lebih baik untuk menyimpan kunci akses dalam EC2 instance. Untuk menetapkan AWS peran ke EC2 instance dan membuatnya tersedia untuk semua aplikasinya, Anda membuat profil instance yang dilampirkan ke instance. Profil instance berisi peran dan memungkinkan program yang berjalan pada EC2 instance untuk mendapatkan kredensi sementara. Untuk informasi selengkapnya, lihat Menggunakan IAM peran untuk memberikan izin ke aplikasi yang berjalan di EC2 instans Amazon di IAMPanduan Pengguna.

Untuk mempelajari apakah akan menggunakan IAM peran atau IAM pengguna, lihat Kapan membuat IAM peran (bukan pengguna) di Panduan IAM Pengguna.

Mengelola akses menggunakan kebijakan

Anda mengontrol akses AWS dengan membuat kebijakan dan melampirkannya ke AWS identitas atau sumber daya. Kebijakan adalah objek AWS yang, ketika dikaitkan dengan identitas atau sumber daya, menentukan izinnya. AWS mengevaluasi kebijakan ini ketika prinsipal (pengguna, pengguna

root, atau sesi peran) membuat permintaan. Izin dalam kebijakan menentukan apakah permintaan diizinkan atau ditolak. Sebagian besar kebijakan disimpan AWS sebagai JSON dokumen. Untuk informasi selengkapnya tentang struktur dan isi dokumen JSON kebijakan, lihat Ringkasan JSON kebijakan di Panduan IAM Pengguna.

Administrator dapat menggunakan AWS JSON kebijakan untuk menentukan siapa yang memiliki akses ke apa. Yaitu, principal dapat melakukan tindakan pada suatu sumber daya, dan dalam suatu syarat.

Secara default, pengguna dan peran tidak memiliki izin. Untuk memberikan izin kepada pengguna untuk melakukan tindakan pada sumber daya yang mereka butuhkan, IAM administrator dapat membuat IAM kebijakan. Administrator kemudian dapat menambahkan IAM kebijakan ke peran, dan pengguna dapat mengambil peran.

IAMkebijakan menentukan izin untuk tindakan terlepas dari metode yang Anda gunakan untuk melakukan operasi. Misalnya, anggaplah Anda memiliki kebijakan yang mengizinkan tindakan `iam:GetRole`. Pengguna dengan kebijakan itu bisa mendapatkan informasi peran dari AWS Management Console, AWS CLI, atau AWS API.

Kebijakan berbasis identitas

Kebijakan berbasis identitas adalah dokumen kebijakan JSON izin yang dapat Anda lampirkan ke identitas, seperti pengguna, grup IAM pengguna, atau peran. Kebijakan ini mengontrol jenis tindakan yang dapat dilakukan oleh pengguna dan peran, di sumber daya mana, dan berdasarkan kondisi seperti apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat Membuat IAM kebijakan di Panduan Pengguna. IAM

Kebijakan berbasis identitas dapat dikategorikan lebih lanjut sebagai kebijakan inline atau kebijakan yang dikelola. Kebijakan inline disematkan langsung ke satu pengguna, grup, atau peran. Kebijakan terkelola adalah kebijakan mandiri yang dapat Anda lampirkan ke beberapa pengguna, grup, dan peran dalam. Akun AWS Kebijakan AWS terkelola mencakup kebijakan terkelola dan kebijakan yang dikelola pelanggan. Untuk mempelajari cara memilih antara kebijakan terkelola atau kebijakan sebaris, lihat Memilih antara kebijakan terkelola dan kebijakan sebaris di IAMPanduan Pengguna.

Kebijakan berbasis sumber daya

Kebijakan berbasis sumber daya adalah dokumen JSON kebijakan yang Anda lampirkan ke sumber daya. Contoh kebijakan berbasis sumber daya adalah kebijakan kepercayaan IAM peran dan kebijakan bucket Amazon S3. Dalam layanan yang mendukung kebijakan berbasis sumber daya, administrator layanan dapat menggunakannya untuk mengontrol akses ke sumber daya tertentu.

Untuk sumber daya tempat kebijakan dilampirkan, kebijakan menentukan tindakan apa yang dapat dilakukan oleh prinsipal tertentu pada sumber daya tersebut dan dalam kondisi apa. Anda harus menentukan prinsipal dalam kebijakan berbasis sumber daya. Prinsipal dapat mencakup akun, pengguna, peran, pengguna federasi, atau. AWS layanan

Kebijakan berbasis sumber daya merupakan kebijakan inline yang terletak di layanan tersebut. Anda tidak dapat menggunakan kebijakan AWS terkelola IAM dalam kebijakan berbasis sumber daya.

Daftar kontrol akses (ACLs)

Access control lists (ACLs) mengontrol prinsipal mana (anggota akun, pengguna, atau peran) yang memiliki izin untuk mengakses sumber daya. ACLsmirip dengan kebijakan berbasis sumber daya, meskipun mereka tidak menggunakan format dokumen kebijakan. JSON

Amazon S3, AWS WAF, dan Amazon VPC adalah contoh layanan yang mendukung. ACLs Untuk mempelajari selengkapnyaACLs, lihat Ikhtisar daftar kontrol akses (ACL) di Panduan Pengembang Layanan Penyimpanan Sederhana Amazon.

Jenis-jenis kebijakan lain

AWS mendukung jenis kebijakan tambahan yang kurang umum. Jenis-jenis kebijakan ini dapat mengatur izin maksimum yang diberikan kepada Anda oleh jenis kebijakan yang lebih umum.

- Batas izin — Batas izin adalah fitur lanjutan tempat Anda menetapkan izin maksimum yang dapat diberikan oleh kebijakan berbasis identitas kepada entitas (pengguna atau peran). IAM IAM Anda dapat menetapkan batasan izin untuk suatu entitas. Izin yang dihasilkan adalah perpotongan antara kebijakan berbasis identitas milik entitas dan batasan izinnya. Kebijakan berbasis sumber daya yang menentukan pengguna atau peran dalam bidang `Principal` tidak dibatasi oleh batasan izin. Penolakan eksplisit dalam salah satu kebijakan ini akan menggantikan pemberian izin. Untuk informasi selengkapnya tentang batas izin, lihat Batas izin untuk IAM entitas di IAMPanduan Pengguna.
- Kebijakan kontrol layanan (SCPs) — SCPs adalah JSON kebijakan yang menentukan izin maksimum untuk organisasi atau unit organisasi (OU) di AWS Organizations. AWS Organizations adalah layanan untuk mengelompokkan dan mengelola secara terpusat beberapa Akun AWS yang dimiliki bisnis Anda. Jika Anda mengaktifkan semua fitur dalam suatu organisasi, maka Anda dapat menerapkan kebijakan kontrol layanan (SCPs) ke salah satu atau semua akun Anda. SCPMembatasi izin untuk entitas di akun anggota, termasuk masing-masing Pengguna root akun AWS. Untuk informasi selengkapnya tentang Organizations danSCPs, lihat Kebijakan kontrol layanan di Panduan AWS Organizations Pengguna.

- Kebijakan sesi – Kebijakan sesi adalah kebijakan lanjutan yang Anda berikan sebagai parameter ketika Anda membuat sesi sementara secara programatis untuk peran atau pengguna terfederasi. Izin sesi yang dihasilkan adalah perpotongan antara kebijakan berbasis identitas pengguna atau peran dan kebijakan sesi. Izin juga bisa datang dari kebijakan berbasis sumber daya. Penolakan secara tegas dalam salah satu kebijakan ini membatalkan izin. Untuk informasi selengkapnya, lihat Kebijakan sesi di Panduan IAM Pengguna.

Berbagai jenis kebijakan

Ketika beberapa jenis kebijakan berlaku pada suatu permintaan, izin yang dihasilkan lebih rumit untuk dipahami. Untuk mempelajari cara AWS menentukan apakah akan mengizinkan permintaan saat beberapa jenis kebijakan terlibat, lihat Logika evaluasi kebijakan di Panduan IAM Pengguna.

Bagaimana EMR Amazon EKS bekerja dengan IAM

Sebelum Anda menggunakan IAM untuk mengelola akses ke EMR AmazonEKS, pelajari IAM fitur apa saja yang tersedia untuk digunakan dengan EMR AmazonEKS.

IAMfitur yang dapat Anda gunakan dengan Amazon EMR di EKS

IAMfitur	Amazon EMR pada EKS dukungan
Kebijakan berbasis identitas	Ya
Kebijakan berbasis sumber daya	Tidak
Tindakan kebijakan	Ya
Sumber daya kebijakan	Ya
Kunci kondisi kebijakan	Ya
ACLs	Tidak
ABAC(tag dalam kebijakan)	Ya
Kredensial sementara	Ya
Izin prinsipal	Ya
Peran layanan	Tidak

IAMfitur	Amazon EMR pada EKS dukungan
Peran terkait layanan	Ya

Untuk mendapatkan tampilan tingkat tinggi tentang cara kerja EMR Amazon EKS dan AWS layanan lainnya dengan sebagian besar IAM fitur, lihat AWS layanan yang berfungsi IAM di Panduan IAM Pengguna.

Kebijakan berbasis identitas untuk Amazon di EMR EKS

Mendukung kebijakan berbasis identitas: Ya

Kebijakan berbasis identitas adalah dokumen kebijakan JSON izin yang dapat Anda lampirkan ke identitas, seperti pengguna, grup IAM pengguna, atau peran. Kebijakan ini mengontrol jenis tindakan yang dapat dilakukan oleh pengguna dan peran, di sumber daya mana, dan berdasarkan kondisi seperti apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat Membuat IAM kebijakan di Panduan Pengguna. IAM

Dengan kebijakan IAM berbasis identitas, Anda dapat menentukan tindakan dan sumber daya yang diizinkan atau ditolak serta kondisi di mana tindakan diizinkan atau ditolak. Anda tidak dapat menentukan secara spesifik prinsipal dalam sebuah kebijakan berbasis identitas karena prinsipal berlaku bagi pengguna atau peran yang melekat kepadanya. Untuk mempelajari semua elemen yang dapat Anda gunakan dalam JSON kebijakan, lihat referensi elemen IAM JSON kebijakan di Panduan IAM Pengguna.

Contoh kebijakan berbasis identitas untuk Amazon di EMR EKS

Untuk melihat contoh Amazon EMR tentang kebijakan EKS berbasis identitas, lihat. Contoh kebijakan berbasis identitas untuk Amazon di EMR EKS

Kebijakan berbasis sumber daya di Amazon pada EMR EKS

Mendukung kebijakan berbasis sumber daya: Tidak

Kebijakan berbasis sumber daya adalah dokumen JSON kebijakan yang Anda lampirkan ke sumber daya. Contoh kebijakan berbasis sumber daya adalah kebijakan kepercayaan IAM peran dan kebijakan bucket Amazon S3. Dalam layanan yang mendukung kebijakan berbasis sumber daya, administrator layanan dapat menggunakannya untuk mengontrol akses ke sumber daya tertentu. Untuk sumber daya tempat kebijakan dilampirkan, kebijakan menentukan tindakan apa yang dapat

dilakukan oleh prinsipal tertentu pada sumber daya tersebut dan dalam kondisi apa. Anda harus menentukan prinsipal dalam kebijakan berbasis sumber daya. Prinsipal dapat mencakup akun, pengguna, peran, pengguna federasi, atau. AWS layanan

Untuk mengaktifkan akses lintas akun, Anda dapat menentukan seluruh akun atau IAM entitas di akun lain sebagai prinsipal dalam kebijakan berbasis sumber daya. Menambahkan prinsipal akun silang ke kebijakan berbasis sumber daya hanya setengah dari membangun hubungan kepercayaan. Ketika prinsipal dan sumber daya berbeda Akun AWS, IAM administrator di akun tepercaya juga harus memberikan izin entitas utama (pengguna atau peran) untuk mengakses sumber daya. Mereka memberikan izin dengan melampirkan kebijakan berbasis identitas kepada entitas. Namun, jika kebijakan berbasis sumber daya memberikan akses ke prinsipal dalam akun yang sama, tidak diperlukan kebijakan berbasis identitas tambahan. Untuk informasi selengkapnya, lihat Akses sumber daya lintas akun IAM di Panduan IAM Pengguna.

Tindakan kebijakan untuk Amazon EMR di EKS

Mendukung tindakan kebijakan: Ya

Administrator dapat menggunakan AWS JSON kebijakan untuk menentukan siapa yang memiliki akses ke apa. Yaitu, principal dapat melakukan tindakan pada suatu sumber daya, dan dalam suatu syarat.

`Action`Elemen JSON kebijakan menjelaskan tindakan yang dapat Anda gunakan untuk mengizinkan atau menolak akses dalam kebijakan. Tindakan kebijakan biasanya memiliki nama yang sama dengan AWS API operasi terkait. Ada beberapa pengecualian, seperti tindakan khusus izin yang tidak memiliki operasi yang cocok. API Ada juga beberapa operasi yang memerlukan beberapa tindakan dalam suatu kebijakan. Tindakan tambahan ini disebut tindakan dependen.

Menyertakan tindakan dalam kebijakan untuk memberikan izin untuk melakukan operasi terkait.

Untuk melihat daftar Amazon EMR tentang EKS tindakan, lihat Tindakan, sumber daya, dan kunci kondisi untuk Amazon EMR EKS di Referensi Otorisasi Layanan.

Tindakan kebijakan EMR di Amazon saat EKS menggunakan awalan berikut sebelum tindakan:

```
emr-containers
```

Untuk menetapkan secara spesifik beberapa tindakan dalam satu pernyataan, pisahkan tindakan tersebut dengan koma.

```
"Action": [
      "emr-containers:action1",
      "emr-containers:action2"
         ]
```

Untuk melihat contoh Amazon EMR tentang kebijakan EKS berbasis identitas, lihat. Contoh kebijakan berbasis identitas untuk Amazon di EMR EKS

Sumber daya kebijakan untuk Amazon EMR di EKS

Mendukung sumber daya kebijakan: Ya

Administrator dapat menggunakan AWS JSON kebijakan untuk menentukan siapa yang memiliki akses ke apa. Yaitu, principal dapat melakukan tindakan pada suatu sumber daya, dan dalam suatu syarat.

Elemen `Resource` JSON kebijakan menentukan objek atau objek yang tindakan tersebut berlaku. Pernyataan harus menyertakan elemen `Resource` atau `NotResource`. Sebagai praktik terbaik, tentukan sumber daya menggunakan Amazon Resource Name (ARN). Anda dapat melakukan ini untuk tindakan yang mendukung jenis sumber daya tertentu, yang dikenal sebagai izin tingkat sumber daya.

Untuk tindakan yang tidak mendukung izin di tingkat sumber daya, misalnya operasi pencantuman, gunakan wildcard (*) untuk menunjukkan bahwa pernyataan tersebut berlaku untuk semua sumber daya.

```
"Resource": "*"
```

Untuk melihat daftar Amazon EMR tentang jenis EKS sumber daya dan jenisnyaARNs, lihat Sumber daya yang ditentukan oleh Amazon EMR EKS di Referensi Otorisasi Layanan. Untuk mempelajari tindakan mana yang dapat Anda tentukan ARN dari setiap sumber daya, lihat Tindakan, sumber daya, dan kunci kondisi untuk EMR Amazon EKS.

Untuk melihat contoh Amazon EMR tentang kebijakan EKS berbasis identitas, lihat. Contoh kebijakan berbasis identitas untuk Amazon di EMR EKS

Kunci kondisi kebijakan untuk Amazon EMR di EKS

Mendukung kunci kondisi kebijakan khusus layanan: Ya

Administrator dapat menggunakan AWS JSON kebijakan untuk menentukan siapa yang memiliki akses ke apa. Yaitu, di mana utama dapat melakukan tindakan pada sumber daya, dan dalam kondisi apa.

Elemen `Condition` (atau blok `Condition`) akan memungkinkan Anda menentukan kondisi yang menjadi dasar suatu pernyataan berlaku. Elemen `Condition` bersifat opsional. Anda dapat membuat ekspresi bersyarat yang menggunakan operator kondisi, misalnya sama dengan atau kurang dari, untuk mencocokkan kondisi dalam kebijakan dengan nilai-nilai yang diminta.

Jika Anda menentukan beberapa elemen `Condition` dalam sebuah pernyataan, atau beberapa kunci dalam elemen `Condition` tunggal, maka AWS akan mengevaluasinya menggunakan operasi `AND` logis. Jika Anda menentukan beberapa nilai untuk satu kunci kondisi, AWS mengevaluasi kondisi menggunakan `OR` operasi logis. Semua kondisi harus dipenuhi sebelum izin pernyataan diberikan.

Anda juga dapat menggunakan variabel placeholder saat menentukan kondisi. Misalnya, Anda dapat memberikan izin IAM pengguna untuk mengakses sumber daya hanya jika ditandai dengan nama IAM pengguna mereka. Untuk informasi selengkapnya, lihat elemen IAM kebijakan: variabel dan tag di Panduan IAM Pengguna.

AWS mendukung kunci kondisi global dan kunci kondisi khusus layanan. Untuk melihat semua kunci kondisi AWS global, lihat kunci konteks kondisi AWS global di Panduan IAM Pengguna.

Untuk melihat daftar Amazon EMR pada kunci EKS kondisi dan untuk mempelajari tindakan dan sumber daya yang dapat Anda gunakan kunci kondisi, lihat Tindakan, sumber daya, dan kunci kondisi untuk Amazon EMR EKS di Referensi Otorisasi Layanan.

Untuk melihat contoh Amazon EMR tentang kebijakan EKS berbasis identitas, lihat. Contoh kebijakan berbasis identitas untuk Amazon di EMR EKS

Daftar kontrol akses (ACLs) di Amazon EMR pada EKS

MendukungACLs: Tidak

Access control lists (ACLs) mengontrol prinsipal mana (anggota akun, pengguna, atau peran) yang memiliki izin untuk mengakses sumber daya. ACLsmirip dengan kebijakan berbasis sumber daya, meskipun mereka tidak menggunakan format dokumen kebijakan. JSON

Kontrol akses berbasis atribut (ABAC) dengan Amazon di EMR EKS

Mendukung ABAC (tag dalam kebijakan)	Ya

Attribute-based access control (ABAC) adalah strategi otorisasi yang mendefinisikan izin berdasarkan atribut. Dalam AWS, atribut ini disebut tag. Anda dapat melampirkan tag ke IAM entitas (pengguna atau peran) dan ke banyak AWS sumber daya. Menandai entitas dan sumber daya adalah langkah pertama dari. ABAC Kemudian Anda merancang ABAC kebijakan untuk mengizinkan operasi ketika tag prinsipal cocok dengan tag pada sumber daya yang mereka coba akses.

ABACmembantu dalam lingkungan yang berkembang pesat dan membantu dengan situasi di mana manajemen kebijakan menjadi rumit.

Untuk mengendalikan akses berdasarkan tag, berikan informasi tentang tag 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 lebih lanjut tentangABAC, lihat Apa ituABAC? dalam IAMUser Guide. Untuk melihat tutorial dengan langkah-langkah penyiapanABAC, lihat Menggunakan kontrol akses berbasis atribut (ABAC) di IAMPanduan Pengguna.

Menggunakan kredensi Sementara dengan Amazon di EMR EKS

Mendukung kredensi sementara: Ya

Beberapa AWS layanan tidak berfungsi saat Anda masuk menggunakan kredensi sementara. Untuk informasi tambahan, termasuk yang AWS layanan bekerja dengan kredensil sementara, lihat AWS layanan yang berfungsi IAM di IAMPanduan Pengguna.

Anda menggunakan kredensi sementara jika Anda masuk AWS Management Console menggunakan metode apa pun kecuali nama pengguna dan kata sandi. Misalnya, ketika Anda mengakses AWS menggunakan link sign-on (SSO) tunggal perusahaan Anda, proses tersebut secara otomatis membuat kredensi sementara. Anda juga akan secara otomatis membuat kredensial sementara ketika Anda masuk ke konsol sebagai seorang pengguna lalu beralih peran. Untuk informasi selengkapnya tentang beralih peran, lihat Beralih ke peran (konsol) di Panduan IAM Pengguna.

Anda dapat secara manual membuat kredensi sementara menggunakan atau. AWS CLI AWS API Anda kemudian dapat menggunakan kredensi sementara tersebut untuk mengakses. AWS AWS merekomendasikan agar Anda secara dinamis menghasilkan kredensi sementara alih-alih menggunakan kunci akses jangka panjang. Untuk informasi selengkapnya, lihat Kredensi keamanan sementara di. IAM

Izin utama lintas layanan untuk Amazon di EMR EKS

Mendukung sesi akses maju (FAS): Ya

Saat Anda menggunakan IAM pengguna atau peran untuk melakukan tindakan AWS, Anda dianggap sebagai prinsipal. Ketika Anda menggunakan beberapa layanan, Anda mungkin melakukan sebuah tindakan yang kemudian menginisiasi tindakan lain di layanan yang berbeda. FASmenggunakan izin dari pemanggilan utama AWS layanan, dikombinasikan dengan permintaan AWS layanan untuk membuat permintaan ke layanan hilir. FASPermintaan hanya dibuat ketika layanan menerima permintaan yang memerlukan interaksi dengan orang lain AWS layanan atau sumber daya untuk menyelesaikannya. Dalam hal ini, Anda harus memiliki izin untuk melakukan kedua tindakan tersebut. Untuk detail kebijakan saat membuat FAS permintaan, lihat Meneruskan sesi akses.

Peran layanan untuk Amazon EMR di EKS

Mendukung peran layanan	Tidak

Peran terkait layanan untuk Amazon di EMR EKS

Mendukung peran terkait layanan	Ya

Untuk detail tentang membuat atau mengelola peran terkait layanan, lihat AWS layanan yang berfungsi dengannya. IAM Cari layanan dalam tabel yang memiliki `Yes` di kolom Peran terkait layanan. Pilih tautan Ya untuk melihat dokumentasi peran terkait layanan untuk layanan tersebut.

Menggunakan peran terkait layanan untuk Amazon EMR di EKS

Amazon EMR di EKS menggunakan AWS Identity and Access Management (IAM) peran terkait layanan. Peran terkait layanan adalah jenis IAM role unik yang terhubung langsung ke Amazon EMR di EKS. Peran tertaut layanan ditentukan sebelumnya oleh Amazon EMR di EKS dan mencakup semua izin yang diperlukan oleh layanan untuk menghubungi layanan AWS lainnya atas nama Anda.

Peran tertaut layanan mempermudah pengaturan Amazon EMR di EKS karena Anda tidak perlu menambahkan izin yang diperlukan secara manual. Amazon EMR di EKS menentukan izin dari peran tertaut layanan, kecuali jika ditentukan lain, hanya Amazon EMR di EKS yang dapat mengambil perannya. Izin yang ditentukan mencakup kebijakan kepercayaan dan kebijakan izin, serta bahwa kebijakan izin tidak dapat dilampirkan ke entitas IAM lainnya.

Anda dapat menghapus peran yang terhubung dengan layanan hanya setelah menghapus sumber daya terkait terlebih dahulu. Ini melindungi sumber daya Amazon EMR di EKS Anda karena Anda tidak dapat secara tidak sengaja menghapus izin untuk mengakses sumber daya.

Untuk informasi tentang layanan lain yang mendukung peran yang terhubung dengan layanan, lihat Layanan AWS yang Berfungsi dengan IAM dan cari layanan yang memiliki Ya di kolom Peran yang Terhubung dengan Layanan. Pilih Ya dengan tautan untuk melihat dokumentasi peran terkait layanan untuk layanan tersebut.

Izin peran terkait layanan untuk Amazon EMR di EKS

Amazon EMR di EKS menggunakan peran terkait layanan yang bernama AWSServiceRoleForAmazonEMRContainers.

`AWSServiceRoleForAmazonEMRContainers` peran terkait layanan memercayakan layanan berikut untuk menjalankan peran tersebut:

- `emr-containers.amazonaws.com`

Kebijakan izin peran `AmazonEMRContainersServiceRolePolicy` memungkinkan Amazon EMR di EKS untuk menyelesaikan serangkaian tindakan pada sumber daya yang ditentukan, seperti yang ditunjukkan pernyataan kebijakan berikut.

> **Note**
>
> Isi kebijakan terkelola berubah, sehingga kebijakan yang ditampilkan di sini mungkin out-of-date. Lihat up-to-date kebijakan paling AmazonEMRContainersServiceRolePolicy diAWS Management Console.

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

```
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeCluster",
                "eks:ListNodeGroups",
                "eks:DescribeNodeGroup",
                "ec2:DescribeRouteTables",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "elasticloadbalancing:DescribeInstanceHealth",
                "elasticloadbalancing:DescribeLoadBalancers",
                "elasticloadbalancing:DescribeTargetGroups",
                "elasticloadbalancing:DescribeTargetHealth"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "acm:ImportCertificate",
                "acm:AddTagsToCertificate"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/emr-container:endpoint:managed-certificate": "true"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "acm:DeleteCertificate"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/emr-container:endpoint:managed-certificate":
 "true"
                }
            }
        }
    ]
```

```
}
```

Anda harus mengonfigurasi izin untuk mengizinkan entitas IAM (seperti pengguna, grup, atau peran) untuk membuat, mengedit, atau menghapus peran terkait layanan. Untuk informasi lebih lanjut, lihat Izin Peran yang Terhubung dengan Layanan di Panduan Pengguna IAM.

Membuat Peran Terkait Layanan untuk Amazon EMR di EKS

Anda tidak perlu membuat peran terkait layanan secara manual. Saat Anda membuat klaster virtual, Amazon EMR di EKS membuat peran terkait layanan bagi Anda.

Jika Anda menghapus peran terkait layanan ini, lalu ingin membuatnya lagi, Anda dapat menggunakan proses yang sama untuk membuat ulang peran tersebut di akun Anda. Saat Anda membuat klaster virtual, Amazon EMR di EKS membuat peran terkait layanan bagi Anda lagi.

Anda juga dapat menggunakan konsol IAM untuk membuat peran terkait layanan dengan kasus penggunaan Amazon EMR di EKS. Di AWS CLI atau API AWS, buat peran yang terhubung dengan layanan dengan nama layanan `emr-containers.amazonaws.com`. Untuk informasi lebih lanjut, lihat Membuat Peran yang Terhubung dengan Layanan di Panduan Pengguna IAM. Jika Anda menghapus peran yang terhubung dengan layanan ini, Anda dapat menggunakan proses yang sama untuk membuat ulang peran tersebut.

Mengedit peran terkait layanan untuk Amazon EMR di EKS

Amazon EMR di EKS tidak mengizinkan Anda untuk mengedit peran terkait layanan `AWSServiceRoleForAmazonEMRContainers`. Setelah membuat peran terkait layanan, Anda tidak dapat mengubah nama peran karena berbagai entitas mungkin mereferensikan peran tersebut. Namun, Anda dapat mengedit penjelasan peran menggunakan IAM. Untuk informasi lebih lanjut, lihat Mengedit Peran yang Terhubung dengan Layanan di Panduan Pengguna IAM.

Menghapus peran terkait layanan untuk Amazon EMR di EKS

Jika Anda tidak perlu lagi menggunakan fitur atau layanan yang memerlukan peran terkait layanan, kami menyarankan Anda menghapus peran tersebut. Dengan begitu, Anda tidak memiliki entitas yang tidak digunakan yang tidak dipantau atau dipelihara secara aktif. Tetapi, Anda harus membersihkan sumber daya peran yang terhubung dengan layanan sebelum menghapusnya secara manual.

> **Note**
>
> Jika layanan Amazon EMR di EKS menggunakan peran tersebut ketika Anda mencoba menghapus sumber daya, penghapusan mungkin gagal. Jika hal itu terjadi, tunggu beberapa menit dan coba lagi.

Untuk menghapus sumber daya Amazon EMR di EKS yang digunakan oleh **`AWSServiceRoleForAmazonEMRContainers`**

1. Buka konsol Amazon EMR.
2. Pilih klaster virtual.
3. Pada halaman `Virtual Cluster` pilih Hapus.
4. Ulangi prosedur ini untuk setiap klaster virtual lainnya di akun Anda.

Untuk menghapus peran terkait layanan secara manual menggunakan IAM

Gunakan konsol IAM, AWS CLI, atau AWS API untuk menghapus peran terkait layanan `AWSServiceRoleForAmazonEMRContainers`. Untuk informasi lebih lanjut, lihat Menghapus Peran Tertaut Layanan dalam Panduan Pengguna IAM.

Wilayah yang Didukung untuk Peran Terkait Layanan Amazon EMR di EKS

Amazon EMR di EKS memberikan dukungan dengan peran terkait layanan di semua Wilayah tempat layanan tersedia. Untuk informasi selengkapnya, lihat Amazon EMR pada titik akhir dan kuota layanan EKS.

Kebijakan terkelola untuk Amazon EMR di EKS

Lihat detail tentang pembaruan kebijakan AWS terkelola untuk EMR Amazon EKS sejak 1 Maret 2021.

Perubahan	Deskripsi	Tanggal
`AmazonEMRContainersServiceRolePolicy` - Menambahkan izin	Izin berikut ditambahkan ke kebijakan:`eks:ListNodeGroups` ,, `eks:DescribeNodeGr`	13 Maret 2023

Perubahan	Deskripsi	Tanggal
untuk mendeskripsikan dan mencantumkan EKS nodegroup Amazon, mendeskripsikan grup target penyeimbang beban, dan menjelask an kesehatan target penyeimbang beban.	`oup elasticloadbalanci ng:DescribeTargetG roups ,elasticloadbalanci ng:DescribeTargetHealth .`	
`AmazonEMRContainer sServiceRolePolicy` - Menambahkan izin untuk mengimpor dan menghapus sertifikat di AWS Certificate Manager.	Izin berikut ditambahkan ke kebijakan:`acm:ImportCertific ate ,acm:AddTagsToCerti ficate ,acm:DeleteCertific ate .`	Desember 3, 2021
EMRAmazon EKS mulai melacak perubahan	EMRAmazon EKS mulai melacak perubahan untuk kebijakan yang AWS dikelola.	1 Maret 2021

Menggunakan peran eksekusi tugas dengan Amazon EMR di EKS

Untuk menggunakan perintah `StartJobRun` untuk mengirimkan tugas berjalan di klaster EKS, Anda harus terlebih dahulu mengorientasi peran eksekusi tugas yang akan digunakan dengan klaster virtual. Untuk informasi selengkapnya, lihat Untuk membuat peran eksekusi tugas di Menyiapkan Amazon EMR di EKS. Anda juga dapat mengikuti petunjuk di bagian Buat Peran IAM untuk pelaksanaan pekerjaan di Amazon EMR di EKS Workshop.

Izin berikut harus disertakan dalam kebijakan kepercayaan untuk peran eksekusi tugas.

```
{
"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Principal": {
```

```
        "Federated": "arn:aws:iam::AWS_ACCOUNT_ID:oidc-provider/OIDC_PROVIDER"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringLike": {
          "OIDC_PROVIDER:sub": "system:serviceaccount:NAMESPACE:emr-containers-sa-*-
*-AWS_ACCOUNT_ID-BASE36_ENCODED_ROLE_NAME"
        }
      }
    }
  ]
}
```

Kebijakan kepercayaan pada contoh sebelumnya hanya memberikan izin ke akun layanan Kubernetes yang dikelola EMR Amazon EMR dengan nama yang cocok dengan polanya. `emr-containers-sa-*-*-`*AWS_ACCOUNT_ID*`-`*BASE36_ENCODED_ROLE_NAME* Akun layanan dengan pola ini akan dibuat secara otomatis pada pengiriman pekerjaan, dan dicakup ke namespace tempat Anda mengirimkan pekerjaan. Kebijakan kepercayaan ini memungkinkan akun layanan ini untuk mengasumsikan peran eksekusi dan mendapatkan kredensial sementara peran eksekusi. Akun layanan dari kluster Amazon EKS yang berbeda atau dari namespace yang berbeda dalam kluster EKS yang sama dibatasi untuk mengasumsikan peran eksekusi.

Anda dapat menjalankan perintah berikut untuk memperbarui kebijakan kepercayaan secara otomatis dalam format yang diberikan di atas.

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

Mengontrol akses ke peran eksekusi

Administrator untuk klaster Amazon EKS Anda dapat membuat EMR Amazon multi-tenant di kluster virtual EKS tempat administrator IAM dapat menambahkan beberapa peran eksekusi. Karena penyewa yang tidak tepercaya dapat menggunakan peran eksekusi ini untuk mengirimkan pekerjaan yang menjalankan kode arbitrer, Anda mungkin ingin membatasi penyewa tersebut sehingga mereka tidak dapat menjalankan kode yang mendapatkan izin yang ditetapkan ke satu atau beberapa peran eksekusi ini. Untuk membatasi kebijakan IAM yang dilampirkan pada identitas IAM, administrator IAM dapat menggunakan kunci kondisi Amazon Resource Name (ARN) opsional. `emr-`

`containers:ExecutionRoleArn` Kondisi ini menerima daftar ARN peran eksekusi yang memiliki izin ke klaster virtual, seperti yang ditunjukkan oleh kebijakan izin berikut.

```
  {
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "emr-containers:StartJobRun",
      "Resource": "arn:aws:emr-containers:REGION:AWS_ACCOUNT_ID:/
virtualclusters/VIRTUAL_CLUSTER_ID",
      "Condition": {
        "ArnEquals": {
          "emr-containers:ExecutionRoleArn": [
            "execution_role_arn_1",
            "execution_role_arn_2",
            ...
          ]
        }
      }
    }
  ]
}
```

Jika Anda ingin mengizinkan semua peran eksekusi yang dimulai dengan awalan tertentu, sepertiMyRole, Anda dapat mengganti operator kondisi `ArnEquals` dengan `ArnLike` operator, dan Anda dapat mengganti `execution_role_arn` nilai dalam kondisi dengan karakter wildcard`*`. Sebagai contoh, `arn:aws:iam::AWS_ACCOUNT_ID:role/MyRole*`. Semua kunci kondisi ARN lainnya juga didukung.

> **Note**
>
> Dengan Amazon EMR di EKS, Anda tidak dapat memberikan izin untuk peran eksekusi berdasarkan tag atau atribut. EMR Amazon di EKS tidak mendukung kontrol akses berbasis tag (TBAC) atau kontrol akses berbasis atribut (ABAC) untuk peran eksekusi.

Contoh kebijakan berbasis identitas untuk Amazon di EMR EKS

Secara default, pengguna dan peran tidak memiliki izin untuk membuat atau memodifikasi Amazon EMR pada EKS sumber daya. Mereka juga tidak dapat melakukan tugas dengan menggunakan AWS Management Console, AWS Command Line Interface (AWS CLI), atau AWS API. Untuk memberikan izin kepada pengguna untuk melakukan tindakan pada sumber daya yang mereka butuhkan, IAM administrator dapat membuat IAM kebijakan. Administrator kemudian dapat menambahkan IAM kebijakan ke peran, dan pengguna dapat mengambil peran.

Untuk mempelajari cara membuat kebijakan IAM berbasis identitas menggunakan contoh dokumen kebijakan ini, lihat Membuat JSON IAM kebijakan di Panduan Pengguna. IAM

Untuk detail tentang tindakan dan jenis sumber daya yang ditentukan oleh EMR AmazonEKS, termasuk format ARNs untuk setiap jenis sumber daya, lihat Kunci tindakan, sumber daya, dan kondisi untuk Amazon EMR EKS di Referensi Otorisasi Layanan.

Topik

- Praktik terbaik kebijakan
- Menggunakan Amazon EMR di EKS konsol
- Mengizinkan pengguna melihat izin mereka sendiri

Praktik terbaik kebijakan

Kebijakan berbasis identitas menentukan apakah seseorang dapat membuat, mengakses, atau menghapus Amazon EMR pada EKS sumber daya 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 terkelola AWS pelanggan yang spesifik untuk kasus penggunaan Anda. Untuk informasi selengkapnya, lihat kebijakan AWSAWS terkelola atau kebijakan terkelola untuk fungsi pekerjaan di Panduan IAM Pengguna.
- Menerapkan izin hak istimewa paling sedikit — Saat Anda menetapkan izin dengan IAM kebijakan, berikan hanya 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 penggunaan IAM untuk menerapkan izin, lihat Kebijakan dan izin IAM di IAM Panduan Pengguna.

- Gunakan ketentuan dalam IAM kebijakan untuk membatasi akses lebih lanjut — Anda dapat menambahkan kondisi ke kebijakan Anda untuk membatasi akses ke tindakan dan sumber daya. Misalnya, Anda dapat menulis kondisi kebijakan untuk menentukan bahwa semua permintaan harus dikirim menggunakanSSL. Anda juga dapat menggunakan ketentuan untuk memberikan akses ke tindakan layanan jika digunakan melalui yang spesifik AWS layanan, seperti AWS CloudFormation. Untuk informasi selengkapnya, lihat elemen IAM JSON kebijakan: Kondisi dalam Panduan IAM Pengguna.
- Gunakan IAM Access Analyzer untuk memvalidasi IAM kebijakan Anda guna memastikan izin yang aman dan fungsional — IAM Access Analyzer memvalidasi kebijakan baru dan yang sudah ada sehingga kebijakan mematuhi bahasa IAM kebijakan () JSON dan praktik terbaik. IAM IAMAccess 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 IAM Access Analyzer di IAMPanduan Pengguna.
- Memerlukan otentikasi multi-faktor (MFA) - Jika Anda memiliki skenario yang mengharuskan IAM pengguna atau pengguna root di Anda Akun AWS, aktifkan MFA untuk keamanan tambahan. Untuk meminta MFA kapan API operasi dipanggil, tambahkan MFA kondisi ke kebijakan Anda. Untuk informasi selengkapnya, lihat Mengonfigurasi API akses MFA yang dilindungi di IAMPanduan Pengguna.

Untuk informasi selengkapnya tentang praktik terbaik diIAM, lihat Praktik terbaik keamanan IAM di Panduan IAM Pengguna.

Menggunakan Amazon EMR di EKS konsol

Untuk mengakses Amazon EMR di EKS konsol, Anda harus memiliki set izin minimum. Izin ini harus memungkinkan Anda untuk membuat daftar dan melihat detail tentang Amazon EMR pada EKS sumber daya di Anda Akun AWS. Jika Anda membuat kebijakan berbasis identitas yang lebih ketat daripada izin minimum yang diperlukan, konsol tidak akan berfungsi sebagaimana mestinya untuk entitas (pengguna atau peran) dengan kebijakan tersebut.

Anda tidak perlu mengizinkan izin konsol minimum untuk pengguna yang melakukan panggilan hanya ke AWS CLI atau. AWS API Sebagai gantinya, izinkan akses hanya ke tindakan yang cocok dengan API operasi yang mereka coba lakukan.

Untuk memastikan bahwa pengguna dan peran masih dapat menggunakan Amazon EMR di EKS konsol, lampirkan juga kebijakan EMR Amazon EKS `ConsoleAccess` atau `ReadOnly` AWS terkelola ke entitas. Untuk informasi selengkapnya, lihat Menambahkan izin ke pengguna di Panduan IAM Pengguna.

Mengizinkan pengguna melihat izin mereka sendiri

Contoh ini menunjukkan cara Anda membuat kebijakan yang memungkinkan IAM pengguna melihat kebijakan sebaris dan terkelola yang dilampirkan pada identitas pengguna mereka. Kebijakan ini mencakup izin untuk menyelesaikan tindakan ini di konsol atau secara terprogram menggunakan atau. AWS CLI AWS API

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
```

```
    ]
}
```

Kebijakan untuk kendali akses berbasis tanda

Anda dapat menggunakan kondisi di kebijakan berbasis identitas Anda untuk mengontrol akses ke klaster virtual dan tugas berjalan berbasis tanda. Untuk informasi lebih lanjut tentang penandaan, lihat Menandai Sumber Daya Amazon EMR di EKS Anda.

Contoh berikut menunjukkan skenario dan cara yang berbeda untuk menggunakan operator kondisi dengan Amazon EMR pada kunci EKS kondisi. Pernyataan IAM kebijakan ini dimaksudkan untuk tujuan demonstrasi saja dan tidak boleh digunakan dalam lingkungan produksi. Ada beberapa cara untuk menggabungkan pernyataan kebijakan untuk memberikan dan menolak izin sesuai dengan kebutuhan Anda. Untuk informasi selengkapnya tentang IAM kebijakan perencanaan dan pengujian, lihat Panduan IAM pengguna.

> **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 Amazon EMR di klaster virtual. EKS

Izinkan tindakan hanya pada sumber daya dengan nilai tanda tertentu

Dalam contoh kebijakan berikut, operator StringEquals kondisi mencoba mencocokkan dev dengan nilai untuk departemen tag. Jika departemen tanda belum ditambahkan ke klaster virtual, atau tidak mengandung dev nilai, kebijakan tersebut tidak berlaku, dan tindakan tersebut tidak diizinkan oleh kebijakan ini. Jika tidak ada pernyataan kebijakan lain mengizinkan tindakan, pengguna hanya dapat bekerja dengan klaster virtual yang memiliki tanda ini dengan nilai ini.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:DescribeVirtualCluster"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/department": "dev"
        }
      }
    }
  ]
}
```

Anda juga dapat menentukan beberapa nilai tanda menggunakan operator syarat. Misalnya, untuk mengizinkan tindakan pada klaster virtual di mana tanda `department` berisi nilai `dev` atau `test`, Anda bisa mengganti blok syarat di contoh sebelumnya dengan berikut ini.

```
"Condition": {
        "StringEquals": {
          "aws:ResourceTag/department": ["dev", "test"]
        }
      }
```

Memerlukan penandaan ketika sumber daya dibuat

Pada contoh di bawah ini, tanda perlu diterapkan saat membuat klaster virtual.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:CreateVirtualCluster"
      ],
      "Resource": "*",
      "Condition": {
```

```
        "StringEquals": {
          "aws:RequestTag/department": "dev"
        }
      }
    }
  ]
}
```

Pernyataan kebijakan berikut mengizinkan pengguna untuk membuat klaster virtual hanya jika klaster memiliki tanda `department`, yang dapat berisi nilai apa pun.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:CreateVirtualCluster"
      ],
      "Resource": "*",
      "Condition": {
        "Null": {
          "aws:RequestTag/department": "false"
        }
      }
    }
  ]
}
```

Tolak akses untuk menambah dan menghapus tanda

Efek dari kebijakan ini adalah untuk menolak izin pengguna untuk menambah atau menghapus tanda apa pun pada klaster virtual yang ditandai dengan tanda `department` yang berisi nilai `dev`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "emr-containers:TagResource",
        "emr-containers:UntagResource"
```

```
      ],
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:ResourceTag/department": "dev"
        }
      }
    }
  ]
}
```

Memecahkan masalah Amazon EMR tentang EKS identitas dan akses

Gunakan informasi berikut untuk membantu Anda mendiagnosis dan memperbaiki masalah umum yang mungkin Anda temui saat bekerja dengan Amazon EMR di EKS danIAM.

Topik

- Saya tidak berwenang untuk melakukan tindakan di Amazon EMR pada EKS
- Saya tidak berwenang untuk melakukan iam: PassRole
- Saya ingin mengizinkan orang di luar AWS akun saya untuk mengakses Amazon saya EMR di EKS sumber daya

Saya tidak berwenang untuk melakukan tindakan di Amazon EMR pada EKS

Jika AWS Management Console memberitahu Anda bahwa Anda tidak berwenang untuk melakukan suatu tindakan, maka Anda harus menghubungi administrator Anda untuk bantuan. Administrator Anda adalah orang yang memberikan nama pengguna dan kata sandi Anda.

Contoh kesalahan berikut terjadi ketika pengguna `mateojackson` mencoba menggunakan konsol untuk melihat detail tentang suatu sumber daya *`my-example-widget`* fiktif, tetapi tidak memiliki izin `emr-containers:`*`GetWidget`* fiktif.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform: emr-
containers:GetWidget on resource: my-example-widget
```

Dalam hal ini, Mateo meminta administratornya untuk memperbarui kebijakannya untuk mengizinkan dia mengakses sumber daya *`my-example-widget`* menggunakan tindakan `emr-containers:`*`GetWidget`*.

Saya tidak berwenang untuk melakukan iam: PassRole

Jika Anda menerima kesalahan yang tidak diizinkan untuk melakukan `iam:PassRole` tindakan, kebijakan Anda harus diperbarui agar Anda dapat meneruskan peran ke EMR AmazonEKS.

Beberapa AWS layanan 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 IAM pengguna bernama `marymajor` mencoba menggunakan konsol untuk melakukan tindakan di Amazon EMR padaEKS. 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 Amazon saya EMR di EKS sumber daya

Anda dapat membuat peran yang dapat digunakan pengguna di akun lain atau orang-orang di luar organisasi Anda untuk mengakses sumber daya Anda. Anda dapat menentukan siapa saja yang dipercaya untuk mengambil peran tersebut. Untuk layanan yang mendukung kebijakan berbasis sumber daya atau daftar kontrol akses (ACLs), Anda dapat menggunakan kebijakan tersebut untuk memberi orang akses ke sumber daya Anda.

Untuk mempelajari selengkapnya, periksa referensi berikut:

- Untuk mengetahui apakah EMR Amazon EKS mendukung fitur-fitur ini, lihatBagaimana EMR Amazon EKS bekerja dengan IAM.
- Untuk mempelajari cara menyediakan akses ke sumber daya Anda di seluruh sumber daya Akun AWS yang Anda miliki, lihat Menyediakan akses ke IAM pengguna lain Akun AWS yang Anda miliki di Panduan IAM Pengguna.

- 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 IAM Pengguna.
- Untuk mempelajari cara menyediakan akses melalui federasi identitas, lihat Menyediakan akses ke pengguna yang diautentikasi secara eksternal (federasi identitas) di 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 Panduan Pengguna. IAM

Pencatatan dan pemantauan

Untuk mendeteksi insiden, menerima peringatan ketika insiden terjadi, dan menanggapinya, gunakan opsi ini dengan Amazon EMR di EKS:

- Pantau Amazon EMR di EKS dengan AWS CloudTrail - AWS CloudTrail menyediakan rekaman tindakan yang diambil oleh pengguna, peran, atau layanan AWS di Amazon EMR di EKS. Ini menangkap panggilan dari konsol Amazon EMR dan panggilan kode ke operasi API Amazon EMR di EKS sebagai peristiwa. Ini memungkinkan Anda untuk menentukan permintaan yang diajukan ke Amazon EMR di EKS, alamat IP dari mana permintaan dibuat, siapa yang mengajukan permintaan, kapan dibuat, dan detail tambahan. Untuk informasi selengkapnya, lihat Pencatatan panggilan API Amazon EMR di EKS menggunakan AWS CloudTrail.
- Gunakan CloudWatch Events dengan Amazon EMR di EKS - CloudWatch Events memberikan stream sistem hampir secara waktu nyata yang menguraikan perubahan dalam sumber daya AWS. CloudWatch Events menjadi sadar akan perubahan operasional saat terjadi, meresponsnya, dan mengambil tindakan korektif seperlunya, dengan mengirim pesan untuk merespons lingkungan, mengaktifkan fungsi, membuat perubahan, dan menangkap informasi status. Untuk menggunakan CloudWatch Events dengan Amazon EMR di EKS, buat aturan yang memicu panggilan API Amazon EMR di EKS melalui CloudTrail. Untuk informasi selengkapnya, lihat Pantau pekerjaan dengan Amazon CloudWatch Events.

Pencatatan panggilan API Amazon EMR di EKS menggunakan AWS CloudTrail

Amazon EMR di EKS terintegrasi dengan AWS CloudTrail, sebuah layanan yang menyediakan catatan tindakan yang dilakukan oleh pengguna, peran, atau layanan AWS dalam Amazon EMR di EKS. CloudTrail menangkap semua panggilan API untuk Amazon EMR di EKS sebagai peristiwa.

Panggilan yang direkam mencakup panggilan dari konsol Amazon EMR di EKS dan panggilan kode ke operasi API Amazon EMR di EKS. Jika membuat jejak, Anda dapat mengaktifkan pengiriman peristiwa CloudTrail berkelanjutan ke bucket Amazon S3, termasuk peristiwa untuk Amazon EMR di EKS. Jika Anda tidak dapat mengonfigurasi jejak, Anda masih dapat melihat peristiwa terbaru dalam konsol CloudTrail di Riwayat peristiwa. Menggunakan informasi yang dikumpulkan oleh CloudTrail, Anda dapat menentukan permintaan yang dibuat ke Amazon EMR di EKS, alamat IP asal permintaan tersebut dibuat, siapa yang membuat permintaan, kapan dibuat, dan detail lainnya.

Untuk mempelajari selengkapnya tentang CloudTrail, lihat AWS CloudTrail Panduan Pengguna.

Informasi Amazon EMR di EKS di CloudTrail

CloudTrail diaktifkan pada akun AWS Anda saat Anda membuat akun tersebut. Ketika aktivitas terjadi di Amazon EMR di EKS, aktivitas tersebut dicatat di peristiwa CloudTrail bersama peristiwa layanan AWS lainnya di Riwayat peristiwa. Anda dapat melihat, mencari, dan mengunduh peristiwa terbaru di akun AWS Anda. Untuk informasi lebih lanjut, lihat Melihat peristiwa dengan riwayat CloudTrail Event.

Untuk catatan peristiwa yang sedang berlangsung di akun AWS Anda, termasuk peristiwa untuk Amazon EMR di EKS, buatlah jejak. Jejak memungkinkan CloudTrail untuk mengirim berkas log ke bucket Amazon S3. Secara default, saat Anda membuat jejak di dalam konsol tersebut, jejak diterapkan ke semua Wilayah AWS. Jejak mencatat peristiwa dari semua Wilayah di partisi AWS dan mengirimkan file log ke bucket Amazon S3 yang Anda tentukan. Selain itu, Anda dapat mengonfigurasi layanan AWS lainnya untuk menganalisis lebih lanjut dan bertindak berdasarkan data peristiwa yang dikumpulkan di log CloudTrail. Untuk informasi selengkapnya, lihat yang berikut:

- Gambaran umum untuk membuat jejak
- Layanan dan integrasi yang didukung CloudTrail
- Mengonfigurasi notifikasi Amazon SNS untuk CloudTrail
- Menerima file log CloudTrail dari beberapa wilayah dan Menerima file log CloudTrail dari beberapa akun

Semua tindakan Amazon EMR di EKS dicatat oleh CloudTrail dan didokumentasikan dalam Dokumentasi API Amazon EMR di EKS. Misalnya, panggilan untuk tindakan `CreateVirtualCluster`, `StartJobRun` dan `ListJobRuns` menghasilkan entri di berkas log CloudTrail.

Setiap entri peristiwa atau log berisi informasi tentang orang yang membuat permintaan tersebut. Informasi identitas membantu Anda menentukan hal berikut:

- Bahwa permintaan dibuat dengan kredensial pengguna root atau pengguna AWS Identity and Access Management (IAM).
- Bahwa permintaan tersebut dibuat dengan kredensial keamanan sementara untuk peran atau pengguna gabungan.
- Apakah permintaan dibuat oleh layanan AWS lain.

Untuk informasi selengkapnya, lihat Elemen identitas pengguna CloudTrail.

Memahami entri file log Amazon EMR di EKS

Jejak adalah konfigurasi yang memungkinkan pengiriman peristiwa sebagai berkas log ke bucket Amazon S3 yang telah Anda tentukan. File log CloudTrail berisi satu atau beberapa entri log. Peristuwa mewakili satu permintaan dari sumber apa pun dan mencakup informasi tentang tindakan yang diminta, tanggal dan waktu tindakan, parameter permintaan, dan sebagainya. Berkas log CloudTrail bukanlah pelacakan tumpukan terurut dari panggilan API publik, sehingga tidak muncul dalam urutan tertentu.

Contoh berikut menunjukkan entri log CloudTrail yang menunjukkan tindakan `ListJobRuns`.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE:admin",
    "arn": "arn:aws:sts::012345678910:assumed-role/Admin/admin",
    "accountId": "012345678910",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::012345678910:role/Admin",
        "accountId": "012345678910",
        "userName": "Admin"
      },
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2020-11-04T21:49:36Z"
      }
    }
```

```
  },
  "eventTime": "2020-11-04T21:52:58Z",
  "eventSource": "emr-containers.amazonaws.com",
  "eventName": "ListJobRuns",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "203.0.113.1",
  "userAgent": "aws-cli/1.11.167 Python/2.7.10 Darwin/16.7.0 botocore/1.7.25",
  "requestParameters": {
    "virtualClusterId": "1K48XXXXXXHCB"
  },
  "responseElements": null,
  "requestID": "890b8639-e51f-11e7-b038-EXAMPLE",
  "eventID": "874f89fa-70fc-4798-bc00-EXAMPLE",
  "readOnly": true,
  "eventType": "AwsApiCall",
  "recipientAccountId": "012345678910"
}
```


Menggunakan Hibah Akses Amazon S3 dengan Amazon di EMR EKS

Ikhtisar Hibah Akses S3 untuk Amazon di EMR EKS

Dengan Amazon EMR merilis 6.15.0 dan yang lebih tinggi, Amazon S3 Access Grants menyediakan solusi kontrol akses yang dapat diskalakan yang dapat Anda gunakan untuk menambah akses ke data Amazon S3 Anda dari Amazon. EMR EKS Jika Anda memiliki konfigurasi izin yang kompleks atau besar untuk data S3, Anda dapat menggunakan Access Grants untuk menskalakan izin data S3 untuk pengguna, peran, dan aplikasi.

Gunakan S3 Access Grants untuk menambah akses ke data Amazon S3 di luar izin yang diberikan oleh peran runtime atau IAM peran yang dilampirkan ke identitas dengan akses ke Amazon Anda di klaster. EMR EKS

Untuk informasi selengkapnya, lihat Mengelola akses dengan Hibah Akses S3 untuk Amazon EMR di Panduan EMR Manajemen Amazon dan Mengelola akses dengan Hibah Akses S3 di Panduan Pengguna Layanan Penyimpanan Sederhana Amazon.

Halaman ini menjelaskan persyaratan untuk menjalankan pekerjaan Spark EMR di Amazon EKS dengan integrasi S3 Access Grants. Dengan EMR Amazon aktifEKS, S3 Access Grants memerlukan pernyataan IAM kebijakan tambahan dalam peran eksekusi untuk pekerjaan Anda, dan konfigurasi

penggantian tambahan untuk. `StartJobRun` API Untuk langkah-langkah menyiapkan Hibah Akses S3 dengan EMR penerapan Amazon lainnya, lihat dokumentasi berikut:

- Menggunakan Hibah Akses S3 dengan Amazon EMR
- Menggunakan Hibah Akses S3 dengan Tanpa Server EMR

Luncurkan Amazon EMR di EKS cluster dengan S3 Access Grants untuk manajemen data

Anda dapat mengaktifkan S3 Access Grants EMR di Amazon EKS dan meluncurkan pekerjaan Spark. Saat aplikasi Anda membuat permintaan untuk data S3, Amazon S3 menyediakan kredenal sementara yang dicakup ke bucket, awalan, atau objek tertentu.

1. Siapkan peran eksekusi pekerjaan untuk Amazon Anda EMR di EKS klaster. Sertakan IAM izin yang diperlukan yang Anda perlukan untuk menjalankan pekerjaan Spark, `s3:GetDataAccess` dan: `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 IAM peran yang untuk eksekusi pekerjaan yang memiliki izin tambahan untuk mengakses S3 secara langsung, maka pengguna mungkin dapat mengakses data terlepas dari izin yang Anda tentukan di S3 Access Grants

2. Kirimkan pekerjaan ke Amazon Anda EMR di EKS klaster dengan label EMR rilis Amazon 6,15 atau lebih tinggi dan `emrfs-site` klasifikasi, seperti yang ditunjukkan contoh berikut. Ganti nilai *`red text`* dengan nilai yang sesuai untuk skenario penggunaan Anda.

```
{
  "name": "myjob",
  "virtualClusterId": "123456",
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "emr-7.2.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "entryPoint_location",
      "entryPointArguments": ["argument1", "argument2"],
       "sparkSubmitParameters": "--class main_class"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "emrfs-site",
        "properties": {
          "fs.s3.s3AccessGrants.enabled": "true",
          "fs.s3.s3AccessGrants.fallbackToIAM": "false"
         }
      }
    ],
  }
}
```

Akses S3 Memberikan pertimbangan dengan Amazon di EMR EKS

Untuk informasi penting dukungan, kompatibilitas, dan perilaku saat Anda menggunakan Hibah Akses Amazon S3 dengan EMR AmazonEKS, lihat pertimbangan Hibah Akses S3 dengan Amazon EMR di Panduan Manajemen Amazon. EMR

Validasi kepatuhan untuk Amazon EMR di EKS

Auditor pihak ketiga menilai keamanan dan kepatuhan Amazon EMR di EKS sebagai bagian dari beberapa program kepatuhan AWS. Program ini mencakup SOC, PCI, FedRAMP, HIPAA, dan lainnya.

Ketahanan di Amazon EMR di EKS

Infrastruktur global AWS dibangun di sekitar AWS Wilayah dan Availability Zones. AWS Wilayah menyediakan beberapa Availability Zone yang terpisah secara fisik dan terisolasi, yang terhubung dengan jaringan berlatensi rendah, throughput yang tinggi, dan sangat redundan. Dengan Availability Zone, Anda dapat mendesain dan mengoperasikan aplikasi dan basis data yang secara otomatis mengalami kegagalan di antara zona tanpa gangguan. Availability Zone lebih tersedia, memiliki toleransi kesalahan, dan dapat diskalakan dibandingkan dengan satu atau beberapa infrastruktur pusat data tradisional.

Untuk informasi selengkapnya tentang AWS Wilayah dan Availability Zone, lihat AWS Infrastruktur Global.

Selain infrastruktur global AWS, Amazon EMR di EKS menawarkan integrasi dengan Amazon S3 melalui EMRFS untuk membantu mendukung ketahanan data dan kebutuhan pencadangan Anda.

Keamanan infrastruktur di Amazon EMR pada EKS

Sebagai layanan terkelola, Amazon EMR dilindungi oleh keamanan jaringan AWS global. Untuk informasi tentang layanan AWS keamanan dan cara AWS melindungi infrastruktur, lihat Keamanan AWS Cloud. Untuk mendesain AWS lingkungan Anda menggunakan praktik terbaik untuk keamanan infrastruktur, lihat Perlindungan Infrastruktur dalam Kerangka Kerja yang AWS Diarsiteksikan dengan Baik Pilar Keamanan.

Anda menggunakan API panggilan yang AWS dipublikasikan untuk mengakses Amazon EMR melalui jaringan. Klien harus mendukung hal-hal berikut:

- Keamanan Lapisan Transportasi (TLS). Kami membutuhkan TLS 1.2 dan merekomendasikan TLS 1.3.
- Suite cipher dengan kerahasiaan maju yang sempurna (PFS) seperti (Ephemeral Diffie-Hellman) atau DHE (Elliptic Curve Ephemeral Diffie-Hellman). ECDHE Sebagian besar sistem modern seperti Java 7 dan versi lebih baru mendukung mode-mode ini.

Selain itu, permintaan harus ditandatangani dengan menggunakan ID kunci akses dan kunci akses rahasia yang terkait dengan IAM prinsipal. Atau Anda bisa menggunakan AWS Security Token Service (AWS STS) untuk membuat kredensial keamanan sementara guna menandatangani permintaan.

Analisis konfigurasi dan kerentanan

AWS menangani tugas-tugas keamanan dasar seperti sistem operasi tamu (OS) dan patching basis data, konfigurasi firewall, dan pemulihan bencana. Prosedur ini telah ditinjau dan disertifikasi oleh pihak ketiga yang sesuai. Untuk detail selengkapnya, lihat sumber daya berikut:

- Validasi kepatuhan untuk Amazon EMR di EKS
- Model Tanggung Jawab Bersama
- Amazon Web Services: Ikhtisar Proses Keamanan (kertas putih)

Connect ke Amazon EMR di EKS Menggunakan VPC endpoints

Anda dapat terhubung langsung ke Amazon EMR di EKS menggunakan VPC endpoints AWS PrivateLink Ketika Anda menggunakan VPC endpoint antarmuka, komunikasi antara VPC dan Amazon EMR di EKS dilakukan sepenuhnya dalam jaringan AWS. Masing-masing VPC endpoint diwakili oleh satu Antarmuka jaringan elastis (ENI) dengan alamat IP privat di subnet VPC Anda.

Antarmuka VPC endpoint menghubungkan VPC Anda langsung ke Amazon EMR di EKS tanpa gateway internet, perangkat NAT, koneksi VPN, atau Koneksi Direct Connect AWS. Instans dalam VPC Anda tidak memerlukan alamat IP publik untuk berkomunikasi dengan API Amazon EMR di EKS.

Anda dapat membuat VPC endpoint antarmuka untuk terhubung ke Amazon EMR di EKS menggunakan perintah AWS Management Console atau AWS Command Line Interface (AWS CLI). Untuk informasi selengkapnya, lihat Membuat Titik Akhir Antarmuka.

Setelah Anda membuat antarmuka VPC endpoint, jika Anda mengaktifkan nama host DNS privat untuk titik akhir, titik akhir Amazon EMR di EKS default menyelesaikan ke VPC endpoint Anda. Titik akhir nama layanan default untuk Amazon EMR di EKS adalah dalam format berikut.

```
emr-containers.Region.amazonaws.com
```

Jika Anda tidak mengaktifkan nama host DNS privat, Amazon VPC menyediakan nama titik akhir DNS yang dapat Anda gunakan dalam format berikut.

```
VPC_Endpoint_ID.emr-containers.Region.vpce.amazonaws.com
```

Untuk informasi selengkapnya, lihat Antarmuka (AWSPrivateLink) di Panduan Pengguna Amazon VPC. Amazon EMR di EKS mendukung panggilan ke semua Tindakan API di dalam VPC Anda.

Anda dapat melampirkan kebijakan VPC endpoint ke VPC endpoint untuk mengontrol akses untuk prinsipal IAM. Anda juga dapat mengasosiasi grup keamanan dengan VPC endpoint untuk mengontrol akses masuk dan keluar berdasarkan asal dan tujuan lalu lintas jaringan, seperti rentang alamat IP. Untuk informasi selengkapnya, lihat Mengontrol Akses ke Layanan dengan VPC Endpoints.

Buat Kebijakan VPC Endpoint untuk Amazon EMR di EKS

Anda dapat membuat kebijakan untuk Amazon VPC endpoint untuk Amazon EMR di EKS untuk menentukan hal berikut:

- Prinsip-prinsip yang dapat atau tidak dapat melakukan tindakan
- Tindakan yang dapat dilakukan
- Sumber daya yang dapat digunakan untuk mengambil tindakan

Untuk informasi selengkapnya, lihat Mengontrol Akses ke Layanan dengan VPC Endpoint dalam Panduan Pengguna Amazon VPC.

Example Kebijakan VPC Endpoint untuk Menolak Semua Akses dari Akun AWS Tertentu

Kebijakan VPC endpoint berikut mneolak akun AWS *123456789012* semua akses ke sumber daya menggunakan titik akhir.

```
{
    "Statement": [
        {
            "Action": "*",
            "Effect": "Allow",
            "Resource": "*",
            "Principal": "*"
        },
        {
            "Action": "*",
            "Effect": "Deny",
            "Resource": "*",
            "Principal": {
                "AWS": [
```

```
                    "123456789012"
                ]
            }
        }
    ]
}
```

Example Kebijakan VPC Endpoint untuk Mengizinkan Akses VPC Hanya ke Prinsipal IAM (Pengguna) Tertentu

Kebijakan VPC endpoint berikut memungkinkan akses penuh hanya ke pengguna IAM *lijuan* di akun AWS *123456789012*. Semua prinsipal IAM lain ditolak aksesnya menggunakan titik akhir.

```
{
    "Statement": [
        {
            "Action": "*",
            "Effect": "Allow",
            "Resource": "*",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::123456789012:user/lijuan"
                ]
            }
        }
    ]
}
```

Example Kebijakan VPC Endpoint untuk Mengizinkan Operasi Read-Only Amazon EMR di EKS

Kebijakan VPC endpoint berikut hanya mengizinkan akun AWS *123456789012* untuk melakukan tindakan Amazon EMR di EKS khusus.

Tindakan yang ditentukan memberikan akses setara dengan read-only untuk Amazon EMR di EKS. Semua tindakan lain pada VPC ditolak untuk akun yang ditentukan. Semua akun lain ditolak akses apa pun. Untuk daftar tindakan Amazon EMR di EKS, lihat Kunci Tindakan, Sumber Daya, dan Kondisi untuk Amazon EMR di EKS.

```
{
    "Statement": [
        {
```

```
            "Action": [
                "emr-containers:DescribeJobRun",
                "emr-containers:DescribeVirtualCluster",
                "emr-containers:ListJobRuns",
                "emr-containers:ListTagsForResource",
                "emr-containers:ListVirtualClusters"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Principal": {
                "AWS": [
                    "123456789012"
                ]
            }
        }
    ]
}
```

Example Kebijakan VPC Endpoint Menolak Akses ke Klaster Virtual Tertentu

Kebijakan VPC endpoint berikut memungkinkan akses penuh untuk semua akun dan prinsipal, namun menolak akses apa pun untuk akun AWS *123456789012* untuk tindakan yang dilakukan pada klaster virtual dengan ID klaster *A1B2CD34EF5G*. Amazon EMR lainnya pada tindakan EKS yang tidak mendukung izin tingkat sumber daya untuk klaster virtual masih diizinkan. Untuk daftar tindakan Amazon EMR di EKS dan jenis sumber daya terkait, lihat Kunci Tindakan, Sumber Daya, dan Kondisi untuk Amazon EMR di EKS - di AWS Identity and Access ManagementPanduan Pengguna.

```
{
    "Statement": [
        {
            "Action": "*",
            "Effect": "Allow",
            "Resource": "*",
            "Principal": "*"
        },
        {
            "Action": "*",
            "Effect": "Deny",
            "Resource": "arn:aws:emr-containers:us-west-2:123456789012:/
virtualclusters/A1B2CD34EF5G",
            "Principal": {
```

```
                "AWS": [
                    "123456789012"
                ]
            }
        }
    ]
}
```

Atur akses lintas akun untuk Amazon EMR di EKS

Anda dapat mengatur akses lintas-akun untuk Amazon EMR di EKS. Akses lintas akun memungkinkan pengguna dari satu akun AWS untuk menjalankan tugas Amazon EMR di EKS dan mengakses data yang mendasari milik akun AWS lain.

Prasyarat

Untuk menyiapkan akses lintas-akun untuk Amazon EMR di EKS, Anda akan menyelesaikan tugas saat masuk ke akun AWS:

- `AccountA` - Sebuah akun AWS di mana Anda telah membuat klaster virtual Amazon EMR di EKS dengan mendaftarkan Amazon EMR dengan namespace pada klaster EKS.
- `AccountB` - Sebuah akun AWS yang berisi bucket Amazon S3 atau tabel DynamoDB yang Anda inginkan untuk diakses tugas Amazon EMR di EKS.

Anda harus menyiapkan hal berikut dalam akun AWS Anda mengatur akses lintas akun:

- Klaster virtual Amazon EMR di EKS di `AccountA` di mana Anda ingin menjalankan tugas.
- Peran eksekusi tugas di `AccountA` yang memiliki izin yang diperlukan untuk menjalankan tugas di klaster virtual. Untuk informasi lebih lanjut, lihat Untuk membuat peran eksekusi tugas dan Menggunakan peran eksekusi tugas dengan Amazon EMR di EKS.

Cara mengakses bucket Amazon S3 lintas akun atau tabel DynamoDB

Untuk mengatur akses lintas-akun untuk Amazon EMR di EKS, selesaikan langkah-langkah berikut.

1. Buat bucket Amazon S3, `cross-account-bucket`, di `AccountB`. Untuk informasi lebih lanjut, lihat Membuat bucket. Jika Anda ingin memiliki akses lintas-akun ke DynamoDB, Anda juga

dapat membuat tabel DynamoDB di `AccountB`. Untuk informasi selengkapnya, lihat Membuat tabel DynamoDB.

2. Buat IAM role `Cross-Account-Role-B` dalam `AccountB` yang dapat mengakses `cross-account-bucket`.

   1. Masuklah ke konsol IAM.
   2. Pilih Peran dan buat peran baru: `Cross-Account-Role-B`. Untuk informasi selengkapnya tentang cara membuat IAM role, lihat Membuat IAM role dalam Panduan Pengguna IAM.
   3. 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 Baru dalam Panduan Pengguna IAM.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::cross-account-bucket",
                "arn:aws:s3:::cross-account-bucket/*"
            ]
        }
    ]
}
```

   Jika akses DynamoDB diperlukan, 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 Buat tabel DynamoDB dalam Panduan Pengguna IAM.

   Berikut ini adalah kebijakan untuk mengakses tabel DynamoDB, `CrossAccountTable`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": "dynamodb:*",
            "Resource": "arn:aws:dynamodb:MyRegion:AccountB:table/
CrossAccountTable"
        }
    ]
}
```

3. Cara mengedit hubungan kepercayaan untuk peran `Cross-Account-Role-B`.

    1. Untuk mengonfigurasi hubungan kepercayaan untuk peran, pilih tab Hubungan Kepercayaan di konsol IAM untuk peran yang dibuat di langkah 2: `Cross-Account-Role-B`.
    2. Pilih Edit Hubungan Kepercayaan.
    3. Tambahkan dokumen kebijakan berikut, yang memungkinkan `Job-Execution-Role-A` dalam `AccountA` untuk mengasumsikan peran `Cross-Account-Role-B` ini.

    ```
    {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "AWS": "arn:aws:iam::AccountA:role/Job-Execution-Role-A"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
    ```

4. Berikan `Job-Execution-Role-A` `AccountA` dengan - STS Asumsikan izin peran untuk mengasumsikan`Cross-Account-Role-B`.

    1. Dalam konsol IAM untuk akun AWS `AccountA`, pilih `Job-Execution-Role-A`.
    2. Tambahkan pernyataan kebijakan berikut pada `Job-Execution-Role-A` untuk mengizinkan tindakan `AssumeRole` di peran `Cross-Account-Role-B`.

    ```
    {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Action": "sts:AssumeRole",
    ```

```
            "Resource": "arn:aws:iam::AccountB:role/Cross-Account-Role-B"
        }
    ]
}
```

5. Untuk akses Amazon S3, atur parameter `spark-submit` berikut (`spark conf`) saat mengirimkan tugas ke Amazon EMR di EKS.

> **Note**
>
> Secara default, EMRFS menggunakan peran eksekusi tugas untuk mengakses bucket S3 dari tugas. Tapi saat `customAWSCredentialsProvider` diatur ke `AssumeRoleAWSCredentialsProvider`, EMRFS menggunakan peran yang sesuai yang Anda tentukan dengan `ASSUME_ROLE_CREDENTIALS_ROLE_ARN` bukan dari `Job-Execution-Role-A` untuk akses Amazon S3.

- `--conf spark.hadoop.fs.s3.customAWSCredentialsProvider=com.amazonaws.emr.AssumeRol`
- `--conf spark.kubernetes.driverEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN=arn:aws:iam::Ac Cross-Account-Role-B \`
- `--conf spark.executorEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN=arn:aws:iam::AccountB:ro Cross-Account-Role-B \`

> **Note**
>
> Anda harus menetapkan `ASSUME_ROLE_CREDENTIALS_ROLE_ARN` baik untuk pelaksana maupun driver `env` dalam konfigurasi tugas spark.

Untuk akses lintas akun DynamoDB, Anda harus mengatur `--conf spark.dynamodb.customAWSCredentialsProvider=com.amazonaws.emr.AssumeRoleAWSCr`

6. Jalankan Amazon EMR pada tugas EKS dengan akses lintas-akun, seperti yang ditunjukkan contoh berikut.

```
aws emr-containers start-job-run \
--virtual-cluster-id 123456 \
--name myjob \
--execution-role-arn execution-role-arn \
--release-label emr-6.2.0-latest \
--job-driver '{"sparkSubmitJobDriver": {"entryPoint": "entryPoint_location",
 "entryPointArguments": ["arguments_list"], "sparkSubmitParameters": "--class
 <main_class> --conf spark.executor.instances=2 --conf spark.executor.memory=2G
 --conf spark.executor.cores=2 --conf spark.driver.cores=1 --conf
 spark.hadoop.fs.s3.customAWSCredentialsProvider=com.amazonaws.emr.AssumeRoleAWSCredentials
 --conf
 spark.kubernetes.driverEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN=arn:aws:iam::AccountB:role/
Cross-Account-Role-B --conf
 spark.executorEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN=arn:aws:iam::AccountB:role/
Cross-Account-Role-B"}} ' \
--configuration-overrides '{"applicationConfiguration": [{"classification":
 "spark-defaults", "properties": {"spark.driver.memory": "2G"}}],
 "monitoringConfiguration": {"cloudWatchMonitoringConfiguration":
 {"logGroupName": "log_group_name", "logStreamNamePrefix": "log_stream_prefix"},
 "persistentAppUI":"ENABLED",  "s3MonitoringConfiguration": {"logUri": "s3://
my_s3_log_location" }}}'
```

Menandai Sumber Daya Amazon EMR di EKS Anda

Untuk membantu Anda mengelola Amazon EMR di EKS, Anda dapat menetapkan metadata Anda sendiri ke setiap sumber daya menggunakan tanda. Topik ini memberikan gambaran umum dari fungsi tanda dan menunjukkan kepada Anda cara membuat tanda.

Topik

Dasar tanda

Tanda adalah sebuah label yang Anda tetapkan ke sebuah sumber daya AWS. Setiap tanda terdiri atas sebuah kunci dan sebuah nilai opsional, yang keduanya Anda tentukan.

Tanda memungkinkan Anda untuk mengategorikan sumber daya AWS dengan atribut seperti tujuan, pemilik, atau lingkungan. Saat Anda memiliki banyak sumber daya dengan jenis yang sama, Anda dapat dengan cepat mengidentifikasi sumber daya tertentu berdasarkan tanda yang telah Anda tetapkan. Misalnya, Anda dapat menentukan serangkaian tanda untuk Amazon EMR Anda pada klaster EKS untuk membantu Anda melacak setiap pemilik dan tingkat tumpukan klaster. Kami menyarankan agar Anda merancang serangkaian konsisten kunci tanda untuk setiap jenis sumber daya. Kemudian Anda dapat mencari dan memfilter sumber daya berdasarkan tanda yang Anda tambahkan.

Tanda tidak secara otomatis ditetapkan ke sumber daya Anda. Setelah Anda menambahkan sebuah tanda, Anda dapat mengedit kunci serta nilai tanda atau menghilangkan tanda dari sumber daya kapanpun yang Anda mau. Jika Anda menghapus sebuah sumber daya, tag apa pun untuk sumber daya tersebut juga dihapus.

Tanda tidak memiliki makna semantik pada Amazon EMR di EKS dan diterjemahkan sebagai serangkaian karakter saja.

Nilai tanda bisa berupa string kosong, tapi tidak null. Kunci tanda tidak bisa berupa string kosong. Jika Anda menambahkan tanda yang memiliki kunci yang sama dengan tanda yang ada pada sumber daya tersebut, nilai yang baru akan menimpa nilai sebelumnya.

Jika Anda menggunakan AWS Identity and Access Management (IAM), Anda dapat mengontrol pengguna mana dalam akun AWS Anda yang memiliki izin untuk mengelola tanda.

Untuk contoh kebijakan kontrol akses berbasis tanda, lihat Kebijakan untuk kendali akses berbasis tanda.

Beri tanda pada sumber daya Anda

Anda dapat menandai klaster virtual dan tugas berjalan baru atau yang sudah ada yang berada dalam status aktif. Status aktif untuk tugas berjalan meliputi: `PENDING`, `SUBMITTED`, `RUNNING`, dan `CANCEL_PENDING`. Status aktif untuk klaster virtual meliputi: `RUNNING`, `TERMINATING` dan `ARRESTED`. Untuk informasi lebih lanjut, lihat Status tugas berjalan dan Status klaster virtual.

Ketika klaster virtual dihentikan, tanda dibersihkan dan tidak lagi dapat diakses.

Jika Anda menggunakan API Amazon EMR di EKS, AWS CLI, atau SDK AWS, Anda dapat memasang tanda ke sumber daya baru menggunakan parameter tanda pada tindakan API yang relevan. Anda dapat memasang tanda ke sumber daya yang ada menggunakan tindakan API `TagResource`.

Anda dapat menggunakan beberapa tindakan penciptaan sumber daya untuk menentukan tanda untuk sumber daya saat sumber daya diciptakan. Dalam kasus ini, jika tanda tidak dapat diterapkan saat sumber daya sedang dibuat, sumber daya gagal untuk dibuat. Mekanisme ini memastikan bahwa sumber daya yang Anda inginkan untuk ditandai pada penciptaan dibuat dengan tanda tertentu atau tidak dibuat sama sekali. Jika Anda menandai sumber daya pada saat penciptaan, Anda tidak perlu untuk menjalankan skrip penandaan khusus setelah penciptaan sumber daya.

Tabel berikut menjelaskan sumber daya Amazon EMR di EKS yang dapat ditandai.

Sumber Daya	Mendukung tanda	Propagasi dukungan tanda	Dukungan penandaan pada penciptaan (API Amazon EMR di EKS, AWS CLI, dan SDK AWS)	API untuk pembuatan (tanda dapat ditambahkan selama pembuatan)
Klaster virtual	Ya	Tidak. Tanda yang terkait	Ya	CreateVirtualCluster

Sumber Daya	Mendukung tanda	Propagasi dukungan tanda	Dukungan penandaan pada penciptaan (API Amazon EMR di EKS, AWS CLI, dan SDK AWS)	API untuk pembuatan (tanda dapat ditambahkan selama pembuatan)
		dengan klaster virtual tidak menyebarkan ke tugas berjalan yang dikirimkan ke cluster virtual tersebut.		
Tugas berjalan	Ya	Tidak	Ya	StartJobRun

Batasan tag

Batasan dasar berikut berlaku untuk tag:

- Jumlah maksimum tanda per sumber daya – 50
- Untuk setiap sumber daya, setiap kunci tag harus unik, dan setiap kunci tag hanya dapat memiliki satu nilai.
- Panjang kunci maksimum – 128 karakter Unicode dalam UTF-8
- Panjang nilai maksimum – 256 karakter Unicode dalam UTF-8
- Jika skema penandaan Anda digunakan di beberapa layanan dan sumber daya AWS, ingatlah bahwa layanan lain mungkin memiliki pembatasan pada karakter yang diizinkan. Karakter yang secara umum diperbolehkan adalah huruf, angka, spasi yang dapat diwakili dalam UTF-8, serta karakter berikut: + - = . _ : / @.
- Kunci dan nilai tag peka huruf besar dan kecil.
- Nilai tanda bisa berupa string kosong, tapi tidak null. Kunci tanda tidak bisa berupa string kosong.
- Jangan gunakan `aws:`, `AWS:`, atau kombinasi huruf besar atau kecil seperti prefiks baik untuk kunci atau nilai. Ini hanya diperuntukkan bagi penggunaan AWS.

Bkerja dengan tanda menggunakan AWS CLI dan API Amazon EMR di EKS

Gunakan perintah AWS CLI berikut atau operasi API Amazon EMR di EKS untuk menambahkan, memperbarui, membuat daftar, dan menghapus tanda untuk sumber daya Anda.

Tugas	AWS CLI	Tindakan API
Penambahan atau penimpaan satu tag atau lebih	tag-resource	TagResource
Daftar tanda untuk sumber daya	list-tags-for-resource	ListTagsForResource
Penghapusan satu tag atau lebih	untag-resource	UntagResource

Contoh berikut menunjukkan cara menandai atau menghapus tanda pada sumber daya menggunakan AWS CLI.

Contoh 1: Menandai klaster virtual yang ada

Perintah berikut memberi tanda klaster virtual yang ada.

```
aws emr-containers tag-resource --resource-arn resource_ARN --tags team=devs
```

Contoh 2: Untag klaster virtual yang ada

Perintah berikut menghapus tanda dari klaster virtual yang ada.

```
aws emr-containers untag-resource --resource-arn resource_ARN --tag-keys tag_key
```

Contoh 3: Daftar tanda untuk sumber daya

Perintah berikut membuat daftar tanda yang terkait dengan sumber daya yang ada.

```
aws emr-containers list-tags-for-resource --resource-arn resource_ARN
```

Memecahkan masalah Amazon EMR di EKS

Bagian ini menjelaskan cara memecahkan masalah dengan Amazon EMR di EKS. Untuk informasi tentang cara memecahkan masalah umum dengan Amazon EMR, lihat Memecahkan masalah klaster di Panduan Manajemen Amazon EMR.

Topik

Pekerjaan pemecahan masalah yang menggunakan PersistentVolumeClaims (PVC)

Jika kamu perlu membuat, mencantumkan, atau menghapus PersistentVolumeClaims (PVC) untuk sebuah job tetapi tidak menambahkan izin PVC ke role emr-container Kubernetes default, maka pekerjaan akan gagal saat kamu mengirimkannya. Tanpa izin ini, peran emr-container tidak dapat membuat peran yang diperlukan untuk driver Spark atau klien Spark. Tidak cukup menambahkan izin ke driver Spark atau peran klien, seperti yang disarankan oleh pesan kesalahan. Peran utama emr-container harus menyertakan izin yang diperlukan juga. Bagian ini menjelaskan cara menambahkan izin yang diperlukan ke peran utama emr-container.

Verifikasi

Untuk memverifikasi apakah peran emr-container Anda memiliki izin yang diperlukan atau tidak, setel variabel NAMESPACE dengan nilai Anda sendiri dan kemudian jalankan perintah berikut:

```
export NAMESPACE=YOUR_VALUE
kubectl describe role emr-containers -n ${NAMESPACE}
```

Selain itu, untuk memverifikasi apakah peran Spark dan klien memiliki izin yang diperlukan, jalankan perintah berikut:

```
kubectl describe role emr-containers-role-spark-driver -n ${NAMESPACE}
kubectl describe role emr-containers-role-spark-client -n ${NAMESPACE}
```

Jika izin tidak ada, lanjutkan dengan tambalan, sebagai berikut.

Patch

1. Jika pekerjaan tanpa izin sedang berjalan, hentikan pekerjaan ini.
2. Buat file bernama RBAC_Patch.py sebagai berikut:

```
import os
import subprocess as sp
import tempfile as temp
import json
import argparse
import uuid

def delete_if_exists(dictionary: dict, key: str):
    if dictionary.get(key, None) is not None:
        del dictionary[key]

def doTerminalCmd(cmd):
    with temp.TemporaryFile() as f:
        process = sp.Popen(cmd, stdout=f, stderr=f)
        process.wait()
        f.seek(0)
        msg = f.read().decode()
    return msg

def patchRole(roleName, namespace, extraRules, skipConfirmation=False):
    cmd = f"kubectl get role {roleName} -n {namespace} --output json".split(" ")
    msg = doTerminalCmd(cmd)
    if "(NotFound)" in msg and "Error" in msg:
        print(msg)
        return False
    role = json.loads(msg)
    rules = role["rules"]
    rulesToAssign = extraRules[::]
    passedRules = []
    for rule in rules:
        apiGroups = set(rule["apiGroups"])
        resources = set(rule["resources"])
        verbs = set(rule["verbs"])
        for extraRule in extraRules:
            passes = 0
            apiGroupsExtra = set(extraRule["apiGroups"])
```

```
            resourcesExtra = set(extraRule["resources"])
            verbsExtra = set(extraRule["verbs"])
            passes += len(apiGroupsExtra.intersection(apiGroups)) >=
len(apiGroupsExtra)
            passes += len(resourcesExtra.intersection(resources)) >=
len(resourcesExtra)
            passes += len(verbsExtra.intersection(verbs)) >= len(verbsExtra)
            if passes >= 3:
                if extraRule not in passedRules:
                    passedRules.append(extraRule)
                    if extraRule in rulesToAssign:
                        rulesToAssign.remove(extraRule)
                break
    prompt_text = "Apply Changes?"
    if len(rulesToAssign) == 0:
        print(f"The role {roleName} seems to already have the necessary
permissions!")
        prompt_text = "Proceed anyways?"
    for ruleToAssign in rulesToAssign:
        role["rules"].append(ruleToAssign)
    delete_if_exists(role, "creationTimestamp")
    delete_if_exists(role, "resourceVersion")
    delete_if_exists(role, "uid")
    new_role = json.dumps(role, indent=3)
    uid = uuid.uuid4()
    filename = f"Role-{roleName}-New_Permissions-{uid}-TemporaryFile.json"
    try:
        with open(filename, "w+") as f:
            f.write(new_role)
            f.flush()
        prompt = "y"
        if not skipConfirmation:
            prompt = input(
                doTerminalCmd(f"kubectl diff -f {filename}".split(" ")) +
f"\n{prompt_text} y/n: "
            ).lower().strip()
            while prompt != "y" and prompt != "n":
                prompt = input("Please make a valid selection. y/n:
").lower().strip()
        if prompt == "y":
            print(doTerminalCmd(f"kubectl apply -f {filename}".split(" ")))
    except Exception as e:
        print(e)
    os.remove(f"./{filename}")
```

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument("-n", "--namespace",
                        help="Namespace of the Role. By default its the
 VirtualCluster's namespace",
                        required=True,
                        dest="namespace"
                        )

    parser.add_argument("-p", "--no-prompt",
                        help="Applies the patches without asking first",
                        dest="no_prompt",
                        default=False,
                        action="store_true"
                        )
    args = parser.parse_args()

    emrRoleRules = [
        {
            "apiGroups": [""],
            "resources": ["persistentvolumeclaims"],
            "verbs": ["list", "create", "delete"]
         }

    ]

    driverRoleRules = [
        {
            "apiGroups": [""],
            "resources": ["persistentvolumeclaims"],
            "verbs": ["list", "create", "delete"]
        },
        {
            "apiGroups": [""],
            "resources": ["services"],
            "verbs": ["get", "list", "describe", "create", "delete", "watch"]
        }
    ]

    clientRoleRules = [
        {
            "apiGroups": [""],
            "resources": ["persistentvolumeclaims"],
```

```
            "verbs": ["list", "create", "delete"]
        }
    ]

    patchRole("emr-containers", args.namespace, emrRoleRules, args.no_prompt)
    patchRole("emr-containers-role-spark-driver", args.namespace, driverRoleRules,
args.no_prompt)
    patchRole("emr-containers-role-spark-client", args.namespace, clientRoleRules,
args.no_prompt)
```

3. Jalankan skrip Python:

```
python3 RBAC_Patch.py -n ${NAMESPACE}
```

4. Perbedaan kubectl antara izin baru dan yang lama akan muncul. Tekan y untuk menambal peran.
5. Verifikasi tiga peran dengan izin tambahan sebagai berikut:

```
kubectl describe role -n ${NAMESPACE}
```

6. Jalankan skrip python:

```
python3 RBAC_Patch.py -n ${NAMESPACE}
```

7. Setelah menjalankan perintah, ia akan menampilkan perbedaan kubectl antara izin baru dan yang lama. Tekan y untuk menambal peran.
8. Verifikasi tiga peran dengan izin tambahan:

```
kubectl describe role -n ${NAMESPACE}
```

9. Kirimkan pekerjaan lagi.

Tambalan manual

Jika izin yang diperlukan aplikasi Anda berlaku untuk sesuatu selain aturan PVC, Anda dapat secara manual menambahkan izin Kubernetes untuk klaster virtual Amazon EMR Anda sesuai kebutuhan.

> **Note**
>
> Peran emr-kontainer adalah peran utama. Ini berarti bahwa ia harus menyediakan semua izin yang diperlukan sebelum Anda dapat mengubah peran driver atau klien yang mendasarinya.

1. Unduh izin saat ini ke file yaml dengan menjalankan perintah di bawah ini:

```
kubectl get role -n ${NAMESPACE} emr-containers -o yaml >> emr-containers-role-patch.yaml
kubectl get role -n ${NAMESPACE} emr-containers-role-spark-driver -o yaml >> driver-role-patch.yaml
kubectl get role -n ${NAMESPACE} emr-containers-role-spark-client -o yaml >> client-role-patch.yaml
```

2. Berdasarkan izin yang dibutuhkan aplikasi Anda, edit setiap file dan tambahkan aturan tambahan seperti berikut ini:

   - emr-containers-role-patch.yaml

```
- apiGroups:
  - ""
  resources:
  - persistentvolumeclaims
  verbs:
  - list
  - create
  - delete
```

   - driver-role-patch.yaml

```
- apiGroups:
  - ""
  resources:
  - persistentvolumeclaims
  verbs:
  - list
  - create
  - delete
- apiGroups:
  - ""
  resources:
```

```
  - services
  verbs:
  - get
  - list
  - describe
  - create
  - delete
  - watch
```

- client-role-patch.yaml

```
- apiGroups:
  - ""
  resources:
  - persistentvolumeclaims
  verbs:
  - list
  - create
  - delete
```

3. Hapus atribut berikut dengan nilainya. Ini diperlukan untuk menerapkan pembaruan.
   - KreasiTimestamp
   - ResourceVersion
   - uid
4. Akhirnya, jalankan tambalan:

```
kubectl apply -f emr-containers-role-patch.yaml
kubectl apply -f driver-role-patch.yaml
kubectl apply -f client-role-patch.yaml
```

Memecahkan masalah Amazon EMR pada penskalaan otomatis vertikal EKS

Lihat bagian berikut jika Anda mengalami masalah saat menyiapkan Amazon EMR pada operator penskalaan otomatis vertikal EKS pada klaster Amazon EKS dengan Operator Lifecycle Manager. Untuk informasi lebih lanjut termasuk langkah-langkah untuk menyelesaikan instalasi, lihatMenggunakan penskalaan otomatis vertikal dengan pekerjaan Amazon EMR Spark.

403 Dilarang

Jika Anda mengikuti langkah-langkahInstal Operator Lifecycle Manager (OLM) di kluster Amazon EKS, menjalankan `olm status` perintah, dan mengembalikan `403 Forbidden` kesalahan seperti di bawah ini, Anda mungkin tidak memperoleh token otentikasi ke repositori Amazon ECR untuk operator.

Untuk mengatasi masalah ini, ulangi langkah Instal EMR Amazon pada operator penskalaan otomatis vertikal EKS untuk mendapatkan token. Kemudian, coba instalasi lagi.

```
Error: FATA[0002] Failed to run bundle: pull bundle image: error pulling image IMAGE.
error resolving name : unexpected status code [manifests latest]: 403 Forbidden
```

Namespace Kubernetes tidak ditemukan

Saat Anda mengatur Amazon EMR pada operator penskalaan otomatis vertikal EKS pada klaster Amazon EKS, Anda mungkin mendapatkan `namespaces not found` kesalahan seperti yang ditunjukkan di sini:

```
FATA[0020] Failed to run bundle: create catalog: error creating catalog source:
 namespaces "NAME" not found.
```

Jika namespace yang Anda tentukan tidak ada, OLM tidak akan menginstal operator autoscaling vertikal. Untuk mengatasi masalah ini, gunakan perintah berikut untuk membuat namespace. Kemudian, coba instalasi lagi.

```
kubectl create namespace NAME
```

Galat saat menyimpan kredensi Docker

Untuk mengatur penskalaan otomatis vertikal, Anda harus mengautentikasi dan mengambil Amazon EMR pada gambar Docker terkait autoscaling vertikal EKS. Ketika Anda melakukan ini, Anda mungkin mendapatkan kesalahan seperti yang berikut jika Docker tidak berjalan:

```
aws ecr get-login-password \
 --region $REGION | docker login \
 --username AWS \
 --password-stdin $ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com
```

```
Error saving credentials: error storing credentials - err: exit status 1
out: 'Post "http://ipc/registry/credstore-updated": dial unix backend.sock: connect: no
 such file or directory'
```

Untuk mengatasi masalah ini, konfirmasikan bahwa Docker sedang berjalan atau membuka Docker Desktop. Kemudian, coba simpan kredensialmu lagi.

Memecahkan masalah Amazon EMR pada operator EKS

Lihat bagian berikut jika Anda mengalami masalah dengan Amazon EMR pada operator EKS Spark. Untuk informasi lebih lanjut termasuk langkah-langkah untuk menyelesaikan instalasi, lihatMenjalankan pekerjaan Spark dengan operator Spark.

Kesalahan pada instalasi bagan Helm

Jika Anda mengikuti langkah-langkah Instal operator Spark dan mengembalikan `INSTALLATION FAILED` kesalahan seperti yang di bawah ini ketika Anda mencoba menginstal atau memverifikasi bagan Helm, Anda mungkin tidak memperoleh token otentikasi ke repositori Amazon ECR untuk operator.

Untuk mengatasi masalah ini, ulangi langkah di Instal operator Spark untuk autentikasi Helm client Anda ke registrasi Amazon ECR. Kemudian, coba langkah instalasi lagi.

```
Error: INSTALLATION FAILED: Kubernetes cluster unreachable: the server has asked for
 the client to provide credentials
```

UnsupportedFileSystemException: Tidak FileSystem untuk skema “s3"

Anda mungkin menemukan pengecualian berikut di thread “main”:

```
org.apache.hadoop.fs.UnsupportedFileSystemException: No FileSystem for scheme "s3"
```

Jika ini terjadi, tambahkan pengecualian berikut ke `SparkApplication` spesifikasi:

```
 hadoopConf:
   # EMRFS filesystem
   fs.s3.customAWSCredentialsProvider:
 com.amazonaws.auth.WebIdentityTokenCredentialsProvider
```

```
    fs.s3.impl: com.amazon.ws.emr.hadoop.fs.EmrFileSystem
    fs.AbstractFileSystem.s3.impl: org.apache.hadoop.fs.s3.EMRFSDelegate
    fs.s3.buffer.dir: /mnt/s3
    fs.s3.getObject.initialSocketTimeoutMilliseconds: "2000"
    mapreduce.fileoutputcommitter.algorithm.version.emr_internal_use_only.EmrFileSystem:
 "2"
    mapreduce.fileoutputcommitter.cleanup-
failures.ignored.emr_internal_use_only.EmrFileSystem: "true"
 sparkConf:
    # Required for EMR Runtime
    spark.driver.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/hadoop-
aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/share/aws/
emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/security/conf:/
usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-glue-datacatalog-
spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-serde.jar:/usr/share/aws/
sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/hadoop/extrajars/*
    spark.driver.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/lib/
native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/native
    spark.executor.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/hadoop-
aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/share/aws/
emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/security/conf:/
usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-glue-datacatalog-
spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-serde.jar:/usr/share/aws/
sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/hadoop/extrajars/*
    spark.executor.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/lib/
native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/native
```

Amazon EMR pada titik akhir dan kuota layanan EKS

Berikut ini adalah titik akhir layanan dan kuota layanan untuk Amazon EMR di EKS. Untuk terhubung secara terprogram ke AWS layanan, Anda menggunakan titik akhir. Selain AWS titik akhir standar, beberapa AWS layanan menawarkan titik akhir FIPS di Wilayah tertentu. Untuk informasi selengkapnya, lihat AWS titik akhir layanan. Kuota layanan, juga disebut sebagai batas, adalah jumlah maksimum sumber daya layanan atau operasi untuk AWS akun Anda. Untuk informasi lebih lanjut, lihat AWS kuota layanan.

Titik akhir layanan

Wilayah AWS nama	Kode	Titik akhir	Protokol
AS Timur (N. Virginia)	us-east-1	`emr-containers.us-east-1.amazonaws.com`	HTTPS
AS Timur (Ohio)	us-east-2	`emr-containers.us-east-2.amazonaws.com`	HTTPS
AS Barat (California Utara)	us-west-1	`emr-containers.us-west-1.amazonaws.com`	HTTPS
US West (Oregon)	us-west-2	`emr-containers.us-west-2.amazonaws.com`	HTTPS
Asia Pacific (Tokyo)	ap-northeast-1	`emr-containers.ap-northeast-1.amazonaws.com`	HTTPS
Asia Pacific (Seoul)	ap-northeast-2	`emr-containers.ap-northeast-2.amazonaws.com`	HTTPS
Asia Pacific (Mumbai)	ap-south-1	`emr-containers.ap-south-1.amazonaws.com`	HTTPS
Asia Pasifik (Singapura)	ap-southeast-1	`emr-containers.ap-southeast-1.amazonaws.com`	HTTPS

Wilayah AWS nama	Kode	Titik akhir	Protokol
Asia Pasifik (Sydney)	ap-southeast-2	`emr-containers.ap-southeast-2.amazonaws.com`	HTTPS
Asia Pasifik (Hong Kong)	ap-east-1	`emr-containers.ap-east-1.amazonaws.com`	HTTPS
Kanada (Pusat)	ca-central-1	`emr-containers.ca-central-1.amazonaws.com`	HTTPS
Eropa (Frankfurt)	eu-central-1	`emr-containers.eu-central-1.amazonaws.com`	HTTPS
Eropa (Irlandia)	eu-west-1	`emr-containers.eu-west-1.amazonaws.com`	HTTPS
Europe (London)	eu-west-2	`emr-containers.eu-west-2.amazonaws.com`	HTTPS
Europe (Stockholm)	eu-north-1	`emr-containers.eu-north-1.amazonaws.com`	HTTPS
South America (Sao Paulo)	sa-east-1	`emr-containers.sa-east-1.amazonaws.com`	HTTPS
Timur Tengah (Bahrain)	me-south-1	`emr-containers.me-south-1.amazonaws.com`	HTTPS
AWS GovCloud (AS-Timur)	us-gov-east-1	`emr-containers.us-gov-east-1.amazonaws.com`	HTTPS
AWS GovCloud (AS-Barat)	us-gov-west-1	`emr-containers.us-gov-west-1.amazonaws.com`	HTTPS

Kuota layanan

Amazon EMR di EKS membatasi permintaan API berikut untuk setiap AWS akun berdasarkan per wilayah. Untuk informasi selengkapnya tentang cara penerapan throttling, lihat Throttling Permintaan

API di Referensi API Amazon EC2. Anda dapat meminta peningkatan kuota pembatasan API untuk akun Anda. AWS

Tindakan API	Kapasitas maksimum bucket	Tingkat isi ulang bucket (per detik)
CancelJobRun	25	1
CreateManagedEndpoint	25	1
CreateVirtualCluster	25	1
DeleteManagedEndpoint	25	1
DeleteVirtualCluster	25	1
DescribeJobRun	100	20
DescribeManagedEndpoint	100	5
DescribeVirtualCluster	100	5
ListJobRun	100	5
ListManagedEndpoint	25	1
ListVirtualCluster	100	5
StartJobRun	25	1
At the AWS account level, the bucket maximum capacity and refill rate for the sum of all API actions listed in this table	200	20

Amazon EMR pada EKS rilis

EMRRilis Amazon adalah seperangkat aplikasi open-source dari ekosistem big data. Setiap rilis terdiri dari berbagai aplikasi, komponen, dan fitur big data yang Anda pilih agar EMR Amazon EKS disebarkan dan dikonfigurasi saat menjalankan pekerjaan.

Dimulai dengan EMR rilis Amazon 5.32.0 dan 6.2.0, Anda dapat menerapkan Amazon di. EMR EKS Opsi penyebaran ini tidak tersedia dengan versi EMR rilis Amazon sebelumnya. Anda harus menentukan versi rilis yang didukung ketika Anda mengirimkan tugas Anda.

EMRAmazon EKS menggunakan bentuk label rilis berikut: `emr-x.x.x-latest` atau `emr-x.x.x-yyyymmdd` dengan tanggal rilis tertentu. Misalnya, `emr-7.2.0-latest` atau `emr-7.2.0-20210129`. Saat Anda menggunakan `-latest` akhiran, Anda memastikan bahwa EMR versi Amazon Anda selalu menyertakan pembaruan keamanan terbaru.

> **Note**
>
> Untuk perbandingan antara Amazon EKS dan EMR Amazon yang EMR sedang berjalanEC2, lihat Amazon EMR FAQs di AWS situs web.

Topik

- Amazon EMR pada rilis EKS 6.6.0
- Amazon EMR pada rilis EKS 6.5.0
- Amazon EMR pada rilis EKS 6.4.0
- Amazon EMR pada rilis EKS 6.3.0
- Amazon EMR pada rilis EKS 6.2.0
- Amazon EMR pada rilis EKS 5.36.0
- Amazon EMR pada rilis EKS 5.35.0
- Amazon EMR pada rilis EKS 5.34.0
- Amazon EMR pada rilis EKS 5.33.0
- Amazon EMR pada rilis EKS 5.32.0

Amazon EMR pada rilis EKS 7.2.0

Halaman ini menjelaskan fungsionalitas baru dan yang diperbarui untuk Amazon EMR yang khusus untuk EMR Amazon saat EKS penyebaran. Untuk detail tentang Amazon yang EMR berjalan di Amazon EC2 dan tentang rilis Amazon EMR 7.2.0 secara umum, lihat Amazon EMR 7.2.0 di Panduan EMR Rilis Amazon.

Amazon EMR pada rilis EKS 7.2

Rilis Amazon EMR 7.2.0 berikut tersedia untuk Amazon EMR diEKS. Pilih emr-7.2.0- XXXX rilis tertentu untuk melihat detail lebih lanjut seperti tag gambar kontainer terkait.

Flink releases

Rilis Amazon EMR 7.2.0 berikut tersedia untuk EMR Amazon EKS saat Anda menjalankan aplikasi Flink.

- emr-7.2.0-flink-terbaru
- emr-7.2.0-flink-20240610

Spark releases

Rilis Amazon EMR 7.2.0 berikut tersedia untuk EMR Amazon EKS saat Anda menjalankan aplikasi Spark.

- emr-7.2.0-terbaru
- emr-7.2.0-20240610
- emr-7.2.0-spark-rapids-latest
- emr-7.2.0-spark-rapids-20240610
- emr-7.2.0-java11-latest
- emr-7.2.0-java11-20240610
- emr-7.2.0-java8-latest
- emr-7.2.0-java8-20240610
- emr-7.2.0-spark-rapids-java8-latest
- emr-7.2.0-spark-rapids-java8-20240610
- notebook-spark/emr-7.2.0-latest
- notebook-spark/emr-7.2.0-20240610
- notebook-spark/emr-7.2.0-spark-rapids-latest
- notebook-spark/emr-7.2.0-spark-rapids-20240610
- notebook-spark/emr-7.2.0-java11-latest
- notebook-spark/emr-7.2.0-java11-20240610
- notebook-spark/emr-7.2.0-java8-latest
- notebook-spark/emr-7.2.0-java8-20240610
- notebook-spark/emr-7.2.0-spark-rapids-java8-latest
- notebook-spark/emr-7.2.0-spark-rapids-java8-20240610
- notebook-python/emr-7.2.0-latest
- notebook-python/emr-7.2.0-20240610
- notebook-python/emr-7.2.0-spark-rapids-latest
- notebook-python/emr-7.2.0-spark-rapids-20240610
- notebook-python/emr-7.2.0-java11-latest
- notebook-python/emr-7.2.0-java11-20240610
- notebook-python/emr-7.2.0-java8-latest
- notebook-python/emr-7.2.0-java8-20240610
- notebook-python/emr-7.2.0-spark-rapids-java8-latest

- notebook-python/emr-7.2.0-spark-rapids-java8-20240610
- livy/emr-7.2.0-latest
- livy/emr-7.2.0-20240610
- livy/emr-7.2.0-java11-latest
- livy/emr-7.2.0-java11-20240610
- livy/emr-7.2.0-java8-latest
- livy/emr-7.2.0-java8-20240610

Catatan rilis

Catatan rilis untuk Amazon EMR di EKS 7.2.0

- Aplikasi yang didukung - AWS SDK for Java 2.23.18 and 1.12.705, Apache Spark 3.5.1-amzn-1, Apache Hudi 0.14.1-amzn-0, Apache Iceberg 1.5.0-amzn-0, Delta 3.1.0, Apache Spark RAPIDS 24.02.0-amzn-1, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.18.1-amzn-0, Flink Operator 1.8.0-amzn-1
- Komponen yang didukung - `aws-sagemaker-spark-sdkemr-ddb`,`emr-goodies`,,`emr-s3-select`,`emrfs`,`hadoop-client`,`hudi`,`hudi-spark`,`iceberg`,`spark-kubernetes`.
- Klasifikasi konfigurasi yang didukung

  Untuk digunakan dengan StartJobRundan CreateManagedEndpointAPIs:

Klasifikasi	Deskripsi
`core-site`	Ubah nilai dalam file `core-site.xml` Hadoop.
`emrfs-site`	Ubah EMRFS pengaturan.
`spark-metrics`	Ubah nilai dalam file `metrics.properties` Spark.
`spark-defaults`	Ubah nilai dalam file `spark-defaults.conf` Spark.
`spark-env`	Ubah nilai di lingkungan Spark.

Klasifikasi	Deskripsi
`spark-hive-site`	Ubah nilai dalam file `hive-site.xml` Spark.
`spark-log4j2`	Ubah nilai dalam file `log4j2.properties` Spark.
`emr-job-submitter`	Konfigurasi untuk pod pengirim pekerjaan.

Untuk digunakan secara khusus dengan CreateManagedEndpointAPIs:

Klasifikasi	Deskripsi
`jeg-config`	Ubah nilai dalam file Jupyter Enterprise Gatewayjupyter_enterprise_gateway_config.py .
`jupyter-kernel-overrides`	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering sesuai dengan XML file konfigurasi untuk aplikasi, sepertispark-hive-site.xml. Untuk informasi selengkapnya, lihat Mengkonfigurasi Aplikasi.

Fitur penting

Fitur-fitur berikut disertakan dengan rilis 7.2.0 Amazon EMR diEKS.

- Peningkatan aplikasi - Amazon EMR pada peningkatan aplikasi EKS 7.2.0 termasuk Spark 3.5.1, Flink 1.18.1, dan Flink Operator 1.8.0.
- Autoscaler untuk pembaruan Flink - Rilis 7.2.0 menggunakan konfigurasi open source `job.autoscaler.restart.time-tracking.enabled` untuk mengaktifkan estimasi waktu penskalaan ulang, jadi Anda tidak perlu lagi menetapkan nilai empiris secara manual untuk memulai ulang waktu. Jika Anda menjalankan 7.1.0 atau lebih rendah, Anda masih dapat menggunakan penskalaan EMR otomatis Amazon.

- Integrasi Apache Hudi Apache Flink di EMR Amazon EKS di - Rilis ini menambahkan integrasi antara Apache Hudi dan Apache Flink, sehingga Anda dapat menggunakan operator Flink Kubernetes untuk menjalankan pekerjaan Hudi. Hudi memungkinkan Anda menggunakan operasi tingkat rekaman yang dapat Anda gunakan untuk menyederhanakan manajemen data dan pengembangan pipa data.
- Integrasi Amazon S3 Express One Zone dengan EMR Amazon aktif EKS - Dengan 7.2.0 dan yang lebih tinggi, Anda dapat mengunggah data ke S3 Express One Zone dengan Amazon aktif. EMR EKS S3 Express One Zone adalah kelas penyimpanan Amazon S3 zona tunggal berkinerja tinggi yang memberikan akses data milidetik satu digit yang konsisten untuk sebagian besar aplikasi yang sensitif terhadap latensi. Pada saat rilis, S3 Express One Zone memberikan latensi terendah dan penyimpanan objek cloud kinerja tertinggi di Amazon S3.
- Support untuk konfigurasi default di operator Spark - Operator Spark di Amazon EKS sekarang mendukung konfigurasi default yang sama dengan model start job run EMR di Amazon EKS untuk 7.2.0 dan yang lebih tinggi. Ini berarti bahwa fitur seperti Amazon S3 dan EMRFS tidak lagi memerlukan konfigurasi manual dalam file yaml.

emr-7.2.0-terbaru

Catatan rilis: `emr-7.2.0-latest` saat ini menunjuk ke `emr-7.2.0-20240610`.

Wilayah: `emr-7.2.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-7.2.0:latest`

emr-7.2.0-20240610

Catatan rilis: `7.2.0-20240610` dirilis pada bulan Desember 2023. Ini adalah rilis awal Amazon EMR 7.2.0 (Spark).

Wilayah: `emr-7.2.0-20240610` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-7.2.0:20240610`

emr-7.2.0-flink-terbaru

Catatan rilis: `emr-7.2.0-flink-latest` saat ini menunjuk ke `emr-7.2.0-flink-20240610`.

Wilayah: `emr-7.2.0-flink-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-7.2.0-flink:latest`

emr-7.2.0-flink-20240610

Catatan rilis: `7.2.0-flink-20240610` dirilis pada Desember 2023. Ini adalah rilis awal Amazon EMR 7.2.0 (Flink).

Wilayah: `emr-7.2.0-flink-20240610` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-7.2.0-flink:20240610`

Amazon EMR pada rilis EKS 7.1.0

Halaman ini menjelaskan fungsionalitas baru dan yang diperbarui untuk Amazon EMR yang khusus untuk EMR Amazon saat EKS penyebaran. Untuk detail tentang Amazon yang EMR berjalan di Amazon EC2 dan tentang rilis Amazon EMR 7.1.0 secara umum, lihat Amazon EMR 7.1.0 di Panduan EMR Rilis Amazon.

Amazon EMR pada rilis EKS 7.1

Rilis Amazon EMR 7.1.0 berikut tersedia untuk Amazon EMR diEKS. Pilih emr-7.1.0- XXXX rilis tertentu untuk melihat detail lebih lanjut seperti tag gambar kontainer terkait.

Flink releases

Rilis Amazon EMR 7.1.0 berikut tersedia untuk EMR Amazon EKS saat Anda menjalankan aplikasi Flink.

- emr-7.1.0-flink-terbaru
- emr-7.1.0-flink-20240321

Spark releases

Rilis Amazon EMR 7.1.0 berikut tersedia untuk EMR Amazon EKS saat Anda menjalankan aplikasi Spark.

- emr-7.1.0-terbaru

- emr-7.1.0-20240321
- emr-7.1.0-spark-rapids-latest
- emr-7.1.0-spark-rapids-20240321
- emr-7.1.0-java11-latest
- emr-7.1.0-java11-20240321
- emr-7.1.0-java8-latest
- emr-7.1.0-java8-20240321
- emr-7.1.0-spark-rapids-java8-latest
- emr-7.1.0-spark-rapids-java8-20240321
- notebook-spark/emr-7.1.0-latest
- notebook-spark/emr-7.1.0-20240321
- notebook-spark/emr-7.1.0-spark-rapids-latest
- notebook-spark/emr-7.1.0-spark-rapids-20240321
- notebook-spark/emr-7.1.0-java11-latest
- notebook-spark/emr-7.1.0-java11-20240321
- notebook-spark/emr-7.1.0-java8-latest
- notebook-spark/emr-7.1.0-java8-20240321
- notebook-spark/emr-7.1.0-spark-rapids-java8-latest
- notebook-spark/emr-7.1.0-spark-rapids-java8-20240321
- notebook-python/emr-7.1.0-latest
- notebook-python/emr-7.1.0-20240321
- notebook-python/emr-7.1.0-spark-rapids-latest
- notebook-python/emr-7.1.0-spark-rapids-20240321
- notebook-python/emr-7.1.0-java11-latest
- notebook-python/emr-7.1.0-java11-20240321
- notebook-python/emr-7.1.0-java8-latest
- notebook-python/emr-7.1.0-java8-20240321
- notebook-python/emr-7.1.0-spark-rapids-java8-latest
- notebook-python/emr-7.1.0-spark-rapids-java8-20240321
- livy/emr-7.1.0-latest

- livy/emr-7.1.0-20240321
- livy/emr-7.1.0-java11-latest
- livy/emr-7.1.0-java11-20240321
- livy/emr-7.1.0-java8-latest
- livy/emr-7.1.0-java8-20240321

Catatan rilis

Catatan rilis untuk Amazon EMR di EKS 7.1.0

- Aplikasi yang didukung - AWS SDK for Java 2.23.18 and 1.12.656, Apache Spark 3.5.0-amzn-1, Apache Hudi 0.14.1-amzn-0, Apache Iceberg 1.4.3-amzn-0, Delta 3.0.0, Apache Spark RAPIDS 23.10.0-amzn-1, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.18.1-amzn-0, Flink Operator 1.6.1-amzn-1
- Komponen yang didukung - `aws-sagemaker-spark-sdkemr-ddb`,`emr-goodies`,,`emr-s3-select`,`emrfs`,`hadoop-client`,`hudi`,`hudi-spark`,`iceberg`,`spark-kubernetes`.
- Klasifikasi konfigurasi yang didukung

  Untuk digunakan dengan StartJobRundan CreateManagedEndpointAPIs:

Klasifikasi	Deskripsi
`core-site`	Ubah nilai dalam file `core-site.xml` Hadoop.
`emrfs-site`	Ubah EMRFS pengaturan.
`spark-metrics`	Ubah nilai dalam file `metrics.properties` Spark.
`spark-defaults`	Ubah nilai dalam file `spark-defaults.conf` Spark.
`spark-env`	Ubah nilai di lingkungan Spark.
`spark-hive-site`	Ubah nilai dalam file `hive-site.xml` Spark.

Klasifikasi	Deskripsi
`spark-log4j2`	Ubah nilai dalam file `log4j2.properties` Spark.
`emr-job-submitter`	Konfigurasi untuk pod pengirim pekerjaan.

Untuk digunakan secara khusus dengan CreateManagedEndpointAPIs:

Klasifikasi	Deskripsi
`jeg-config`	Ubah nilai dalam file Jupyter Enterpris e Gateway`jupyter_enterprise _gateway_config.py` .
`jupyter-kernel-overrides`	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering sesuai dengan XML file konfigurasi untuk aplikasi, sepertis`park-hive-site.xml`. Untuk informasi selengkapnya, lihat Mengkonfigurasi Aplikasi.

Fitur penting

Fitur-fitur berikut disertakan dengan rilis 7.1.0 Amazon EMR diEKS.

- Dukungan Apache Livy untuk Amazon EMR di EKS - Dengan Amazon EMR pada EKS rilis 7.1.0 dan yang lebih tinggi, Anda dapat menggunakan Apache Livy di cluster EKS Amazon untuk membuat antarmuka Apache REST Livy untuk mengirimkan pekerjaan Spark atau cuplikan kode Spark. Melakukannya memungkinkan Anda mengambil hasil secara sinkron dan asinkron, sambil tetap memanfaatkan Amazon pada EMR manfaatEKS, seperti runtime Spark yang EMR dioptimalkan Amazon, titik akhir Livy yang diaktifkan, dan pengalaman penyiapan SSL terprogram.

emr-7.1.0-terbaru

Catatan rilis: `emr-7.1.0-latest` saat ini menunjuk ke `emr-7.1.0-20240321`.

Wilayah: `emr-7.1.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-7.1.0:latest`

emr-7.1.0-20240321

Catatan rilis: `7.1.0-20240321` dirilis pada bulan Desember 2023. Ini adalah rilis awal Amazon EMR 7.1.0 (Spark).

Wilayah: `emr-7.1.0-20240321` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-7.1.0:20240321`

emr-7.1.0-flink-terbaru

Catatan rilis: `emr-7.1.0-flink-latest` saat ini menunjuk ke `emr-7.1.0-flink-20240321`.

Wilayah: `emr-7.1.0-flink-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-7.1.0-flink:latest`

emr-7.1.0-flink-20240321

Catatan rilis: `7.1.0-flink-20240321` dirilis pada Desember 2023. Ini adalah rilis awal Amazon EMR 7.1.0 (Flink).

Wilayah: `emr-7.1.0-flink-20240321` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-7.1.0-flink:20240321`

Amazon EMR pada rilis EKS 7.0.0

Halaman ini menjelaskan fungsionalitas baru dan yang diperbarui untuk Amazon EMR yang khusus untuk EMR Amazon saat EKS penyebaran. Untuk detail tentang Amazon yang EMR berjalan

di Amazon EC2 dan tentang rilis Amazon EMR 7.0.0 secara umum, lihat Amazon EMR 7.0.0 di Panduan Rilis Amazon. EMR

Amazon EMR pada rilis EKS 7.0

Rilis Amazon EMR 7.0.0 berikut tersedia untuk Amazon EMR di. EKS Pilih emr-7.0.0- XXXX rilis tertentu untuk melihat detail lebih lanjut seperti tag gambar kontainer terkait.

Flink releases

Rilis Amazon EMR 7.0.0 berikut tersedia untuk EMR Amazon EKS saat Anda menjalankan aplikasi Flink.

- emr-7.0.0-flink-terbaru
- emr-7.0.0-flink-2024321
- emr-7.0.0-batu pipi-20231211

Spark releases

Rilis Amazon EMR 7.0.0 berikut tersedia untuk EMR Amazon EKS saat Anda menjalankan aplikasi Spark.

- emr-7.0.0-terbaru
- emr-7.0.0-20231211
- emr-7.0.0-spark-rapids-latest
- emr-7.0.0-spark-rapids-20231211
- emr-7.0.0-java11-latest
- emr-7.0.0-java11-20231211
- emr-7.0.0-java8-latest
- emr-7.0.0-java8-20231211
- emr-7.0.0-spark-rapids-java8-latest
- emr-7.0.0-spark-rapids-java8-20231211
- notebook-spark/emr-7.0.0-latest
- notebook-spark/emr-7.0.0-20231211

- notebook-spark/emr-7.0.0-spark-rapids-latest
- notebook-spark/emr-7.0.0-spark-rapids-20231211
- notebook-spark/emr-7.0.0-java11-latest
- notebook-spark/emr-7.0.0-java11-20231211
- notebook-spark/emr-7.0.0-java8-latest
- notebook-spark/emr-7.0.0-java8-20231211
- notebook-spark/emr-7.0.0-spark-rapids-java8-latest
- notebook-spark/emr-7.0.0-spark-rapids-java8-20231211
- notebook-python/emr-7.0.0-latest
- notebook-python/emr-7.0.0-20231211
- notebook-python/emr-7.0.0-spark-rapids-latest
- notebook-python/emr-7.0.0-spark-rapids-20231211
- notebook-python/emr-7.0.0-java11-latest
- notebook-python/emr-7.0.0-java11-20231211
- notebook-python/emr-7.0.0-java8-latest
- notebook-python/emr-7.0.0-java8-20231211
- notebook-python/emr-7.0.0-spark-rapids-java8-latest
- notebook-python/emr-7.0.0-spark-rapids-java8-20231211

Catatan rilis

Catatan rilis untuk Amazon EMR di EKS 7.0.0

- Aplikasi yang didukung - AWS SDK for Java 2.20.160-amzn-0 and 1.12.595, Apache Spark 3.5.0-amzn-0, Apache Flink 1.18.0-amzn-0, Flink Operator 1.6.1, Apache Hudi 0.14.0-amzn-1, Apache Iceberg 1.4.2-amzn-0, Delta 3.0.0, Apache Spark RAPIDS 23.10.0-amzn-0, Jupyter Enterprise Gateway 2.6.0
- Komponen yang didukung - `aws-sagemaker-spark-sdkemr-ddb`,`emr-goodies`,,`emr-s3-select`,`emrfs`,`hadoop-client`,`hudi`,`hudi-spark`,`iceberg`,`spark-kubernetes`.
- Klasifikasi konfigurasi yang didukung

  Untuk digunakan dengan StartJobRundan CreateManagedEndpointAPIs:

Klasifikasi	Deskripsi
`core-site`	Ubah nilai dalam file `core-site.xml` Hadoop.
`emrfs-site`	Ubah EMRFS pengaturan.
`spark-metrics`	Ubah nilai dalam file `metrics.properties` Spark.
`spark-defaults`	Ubah nilai dalam file `spark-defaults.conf` Spark.
`spark-env`	Ubah nilai di lingkungan Spark.
`spark-hive-site`	Ubah nilai dalam file `hive-site.xml` Spark.
`spark-log4j`	Ubah nilai dalam file `log4j2.properties` Spark.
`emr-job-submitter`	Konfigurasi untuk pod pengirim pekerjaan.

Untuk digunakan secara khusus dengan CreateManagedEndpointAPIs:

Klasifikasi	Deskripsi
`jeg-config`	Ubah nilai dalam file Jupyter Enterprise Gateway`jupyter_enterprise_gateway_config.py` .
`jupyter-kernel-overrides`	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering sesuai dengan XML file konfigurasi untuk aplikasi, sepertispark-hive-site.xml. Untuk informasi selengkapnya, lihat Mengkonfigurasi Aplikasi.

Fitur penting

Fitur-fitur berikut disertakan dengan rilis 7.0 Amazon EMR diEKS.

- Peningkatan aplikasi - Amazon EMR pada peningkatan aplikasi EKS 7.0.0 termasuk Spark 3.5, Flink 1.18, dan Flink Operator 1.6.1.
- Penyetelan otomatis parameter Flink Autoscaler — Parameter default yang digunakan Flink Autoscaler untuk perhitungan penskalaannya mungkin bukan nilai optimal untuk pekerjaan tertentu. Amazon EMR di EKS 7.0.0 menggunakan tren historis metrik tertentu yang diambil untuk menghitung parameter optimal yang disesuaikan untuk pekerjaan tersebut.

Perubahan

Perubahan berikut disertakan dengan rilis 7.0 Amazon EMR diEKS.

- Amazon Linux 2023 - Dengan Amazon EMR pada EKS 7.0.0 dan yang lebih tinggi, semua gambar kontainer didasarkan pada Amazon Linux 2023.
- Spark menggunakan Java 17 sebagai runtime default - Amazon EMR di EKS 7.0.0 Spark menggunakan Java 17 sebagai runtime default. Jika perlu, Anda dapat beralih untuk menggunakan Java 8 atau Java 11 dengan label rilis yang sesuai seperti yang disediakan dalam Amazon EMR pada rilis EKS 7.0 daftar.

emr-7.0.0-terbaru

Catatan rilis: emr-7.0.0-latest saat ini menunjuk ke emr-7.0.0-2024321.

Wilayah: emr-7.0.0-latest tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: emr-7.0.0:latest

emr-7.0.0-2024321

Catatan rilis: `7.0.0-2024321` dirilis pada 11 Maret 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-7.0.0-2024321` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-7.0.0:2024321`

emr-7.0.0-20231211

Catatan rilis: `7.0.0-20231211` dirilis pada bulan Desember 2023. Ini adalah rilis awal Amazon EMR 7.0.0 (Spark).

Wilayah: `emr-7.0.0-20231211` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-7.0.0:20231211`

emr-7.0.0-flink-terbaru

Catatan rilis: `emr-7.0.0-flink-latest` saat ini menunjuk ke `emr-7.0.0-flink-2024321`.

Wilayah: `emr-7.0.0-flink-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-7.0.0-flink:latest`

emr-7.0.0-flink-2024321

Catatan rilis: `7.0.0-flink-2024321` dirilis pada 11 Maret 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-7.0.0-flink-2024321` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-7.0.0-flink:2024321`

emr-7.0.0-batu pipi-20231211

Catatan rilis: `7.0.0-flink-20231211` dirilis pada Desember 2023. Ini adalah rilis awal Amazon EMR 7.0.0 (Flink).

Wilayah: `emr-7.0.0-flink-20231211` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-7.0.0-flink:20231211`

Amazon EMR pada rilis EKS 6.15.0

Halaman ini menjelaskan fungsionalitas baru dan yang diperbarui untuk Amazon EMR yang khusus untuk EMR Amazon saat EKS penyebaran. Untuk detail tentang Amazon yang EMR berjalan di Amazon EC2 dan tentang rilis Amazon EMR 6.15.0 secara umum, lihat Amazon EMR 6.15.0 di Panduan Rilis Amazon. EMR

Amazon EMR pada rilis EKS 6,15

Rilis Amazon EMR 6.15.0 berikut tersedia untuk Amazon EMR di. EKS Pilih emr-6.15.0- XXXX rilis tertentu untuk melihat detail lebih lanjut seperti tag gambar kontainer terkait.

Flink releases

Rilis Amazon EMR 6.15.0 berikut tersedia untuk EMR Amazon EKS saat Anda menjalankan aplikasi Flink.

- emr-6.15.0-flink-terbaru
- emr-6.15.0-batu pipi-20240105
- emr-6.15.0-batu pipi-20231109

Spark releases

Rilis Amazon EMR 6.15.0 berikut tersedia untuk EMR Amazon EKS saat Anda menjalankan aplikasi Spark.

- emr-6.15.0-terbaru
- emr-6.15.0-20231109

- emr-6.15.0-spark-rapids-latest
- emr-6.15.0-spark-rapids-20231109
- emr-6.15.0-java11-latest
- emr-6.15.0-java11-20231109
- emr-6.15.0-java17-latest
- emr-6.15.0-java17-20231109
- emr-6.15.0-java17-al2023-latest
- emr-6.15.0-java17-al2023-20231109
- emr-6.15.0-spark-rapids-java17-latest
- emr-6.15.0-spark-rapids-java17-20231109
- emr-6.15.0-spark-rapids-java17-al2023-latest
- emr-6.15.0-spark-rapids-java17-al2023-20231109
- notebook-spark/emr-6.15.0-latest
- notebook-spark/emr-6.15.0-20231109
- notebook-spark/emr-6.15.0-spark-rapids-latest
- notebook-spark/emr-6.15.0-spark-rapids-20231109
- notebook-spark/emr-6.15.0-java11-latest
- notebook-spark/emr-6.15.0-java11-20231109
- notebook-spark/emr-6.15.0-java17-latest
- notebook-spark/emr-6.15.0-java17-20231109
- notebook-spark/emr-6.15.0-java17-al2023-latest
- notebook-spark/emr-6.15.0-java17-al2023-20231109
- notebook-python/emr-6.15.0-latest
- notebook-python/emr-6.15.0-20231109
- notebook-python/emr-6.15.0-spark-rapids-latest
- notebook-python/emr-6.15.0-spark-rapids-20231109
- notebook-python/emr-6.15.0-java11-latest
- notebook-python/emr-6.15.0-java11-20231109
- notebook-python/emr-6.15.0-java17-latest
- notebook-python/emr-6.15.0-java17-20231109

- notebook-python/emr-6.15.0-java17-al2023-latest
- notebook-python/emr-6.15.0-java17-al2023-20231109

Catatan rilis

Catatan rilis untuk Amazon EMR di EKS 6.15.0

- Aplikasi yang didukung - AWS SDK for Java 1.12.569, Apache Spark 3.4.1-amzn-2, Apache Flink 1.17.1-amzn-1, Apache Hudi 0.14.0-amzn-0, Apache Iceberg 1.4.0-amzn-0, Delta 2.4.0, Apache Spark RAPIDS 23.08.01-amzn-0, Jupyter Enterprise Gateway 2.6.0
- Komponen yang didukung - `aws-sagemaker-spark-sdkemr-ddb`,`emr-goodies`,,`emr-s3-select`,`emrfs`,`hadoop-client`,`hudi`,`hudi-spark`,`iceberg`,`spark-kubernetes`.
- Klasifikasi konfigurasi yang didukung

  Untuk digunakan dengan StartJobRundan CreateManagedEndpointAPIs:

Klasifikasi	Deskripsi
`core-site`	Ubah nilai dalam file `core-site.xml` Hadoop.
`emrfs-site`	Ubah EMRFS pengaturan.
`spark-metrics`	Ubah nilai dalam file `metrics.properties` Spark.
`spark-defaults`	Ubah nilai dalam file `spark-defaults.conf` Spark.
`spark-env`	Ubah nilai di lingkungan Spark.
`spark-hive-site`	Ubah nilai dalam file `hive-site.xml` Spark.
`spark-log4j`	Ubah nilai dalam file `log4j2.properties` Spark.
`emr-job-submitter`	Konfigurasi untuk pod pengirim pekerjaan.

Untuk digunakan secara khusus dengan CreateManagedEndpointAPIs:

Klasifikasi	Deskripsi
`jeg-config`	Ubah nilai dalam file Jupyter Enterprise Gatewayjupyter_enterprise_gateway_config.py .
`jupyter-kernel-overrides`	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering sesuai dengan XML file konfigurasi untuk aplikasi, sepertispark-hive-site.xml. Untuk informasi selengkapnya, lihat Mengkonfigurasi Aplikasi.

Fitur penting

Fitur-fitur berikut disertakan dengan rilis 6.15 Amazon EMR diEKS.

- EMRAmazon aktif EKS dengan Apache Flink - Dengan Amazon EMR di EKS 6.15.0, Anda dapat menjalankan aplikasi berbasis Apache Flink Anda bersama dengan jenis aplikasi lain di cluster Amazon yang sama. EKS Ini membantu meningkatkan pemanfaatan sumber daya dan menyederhanakan manajemen infrastruktur. Anda dapat memanfaatkan Instans Spot di aplikasi Flink dengan penonaktifan yang anggun, dan mencapai waktu restart yang lebih cepat dengan pemulihan berbutir halus dan pemulihan lokal tugas dengan Amazon. EBS Fitur aksesibilitas dan pemantauan mencakup kemampuan untuk meluncurkan aplikasi Flink dengan toples yang disimpan di Amazon S3, akses ke Katalog Data AWS Glue, pemantauan integrasi dengan Amazon S3 dan CloudWatch Amazon, dan rotasi log kontainer.

emr-6.15.0-terbaru

Catatan rilis: `emr-6.15.0-latest` saat ini menunjuk ke `emr-6.15.0-20240105`.

Wilayah: `emr-6.15.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.15.0:latest`

emr-6.15.0-20240105

Catatan rilis: `6.15.0-20240105` dirilis pada 17 Januari 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.15.0-20240105` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.15.0:20240105`

emr-6.15.0-20231109

Catatan rilis: `6.15.0-20231109` dirilis pada 17 November 2023. Ini adalah rilis awal Amazon EMR 6.15.0.

Wilayah: `emr-6.15.0-20231109` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.15.0:20231109`

emr-6.15.0-flink-terbaru

Catatan rilis: `emr-6.15.0-flink-latest` saat ini menunjuk ke `emr-6.15.0-flink-20240105`.

Wilayah: `emr-6.15.0-flink-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.15.0-flink:latest`

emr-6.15.0-batu pipi-20240105

Catatan rilis: `6.15.0-flink-20240105` dirilis pada 17 Januari 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.15.0-flink-20240105` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.15.0-flink:20240105`

emr-6.15.0-batu pipi-20231109

Catatan rilis: `6.15.0-flink-20231109` dirilis pada 17 November 2023. Ini adalah rilis awal Amazon EMR 6.15.0.

Wilayah: `emr-6.15.0-flink-20231109` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.15.0-flink:20231109`

Amazon EMR pada rilis EKS 6.14.0

Halaman ini menjelaskan fungsionalitas baru dan yang diperbarui untuk Amazon EMR yang khusus untuk EMR Amazon saat EKS penyebaran. Untuk detail tentang Amazon yang EMR berjalan di Amazon EC2 dan tentang rilis Amazon EMR 6.14.0 secara umum, lihat Amazon EMR 6.14.0 di Panduan Rilis Amazon. EMR

Amazon EMR pada rilis EKS 6,14

Rilis Amazon EMR 6.14.0 berikut tersedia untuk Amazon EMR di. EKS Pilih emr-6.14.0- XXXX rilis tertentu untuk melihat detail lebih lanjut seperti tag gambar kontainer terkait.

- emr-6.14.0-terbaru
- emr-6.14.0-20231005
- emr-6.14.0-spark-rapids-latest
- emr-6.14.0-spark-rapids-20231005
- emr-6.14.0-java11-latest
- emr-6.14.0-java11-20231005
- emr-6.14.0-java17-latest
- emr-6.14.0-java17-20231005
- emr-6.14.0-java17-al2023-latest
- emr-6.14.0-java17-al2023-20231005
- emr-6.14.0-spark-rapids-java17-latest
- emr-6.14.0-spark-rapids-java17-20231005
- emr-6.14.0-spark-rapids-java17-al2023-latest
- emr-6.14.0-spark-rapids-java17-al2023-20231005

- notebook-spark/emr-6.14.0-latest
- notebook-spark/emr-6.14.0-20231005
- notebook-spark/emr-6.14.0-spark-rapids-latest
- notebook-spark/emr-6.14.0-spark-rapids-20231005
- notebook-spark/emr-6.14.0-java11-latest
- notebook-spark/emr-6.14.0-java11-20231005
- notebook-spark/emr-6.14.0-java17-latest
- notebook-spark/emr-6.14.0-java17-20231005
- notebook-spark/emr-6.14.0-java17-al2023-latest
- notebook-spark/emr-6.14.0-java17-al2023-20231005
- notebook-python/emr-6.14.0-latest
- notebook-python/emr-6.14.0-20231005
- notebook-python/emr-6.14.0-spark-rapids-latest
- notebook-python/emr-6.14.0-spark-rapids-20231005
- notebook-python/emr-6.14.0-java11-latest
- notebook-python/emr-6.14.0-java11-20231005
- notebook-python/emr-6.14.0-java17-latest
- notebook-python/emr-6.14.0-java17-20231005
- notebook-python/emr-6.14.0-java17-al2023-latest
- notebook-python/emr-6.14.0-java17-al2023-20231005

Catatan rilis

Catatan rilis untuk Amazon EMR di EKS 6.14.0

- Aplikasi yang didukung - AWS SDK for Java 1.12.543, Apache Spark 3.4.1-amzn-1, Apache Hudi 0.13.1-amzn-2, Apache Iceberg 1.3.0-amzn-0, Delta 2.4.0, Apache Spark 23.06.0-amzn-2, Jupyter Enterprise Gateway 2.7.0 RAPIDS
- Komponen yang didukung - `aws-sagemaker-spark-sdkemr-ddb`,`emr-goodies`,,`emr-s3-select`,`emrfs`,`hadoop-client`,`hudi`,`hudi-spark`,`iceberg`,`spark-kubernetes`.
- Klasifikasi konfigurasi yang didukung

  Untuk digunakan dengan StartJobRundan CreateManagedEndpointAPIs:

Klasifikasi	Deskripsi
`core-site`	Ubah nilai dalam file `core-site.xml` Hadoop.
`emrfs-site`	Ubah EMRFS pengaturan.
`spark-metrics`	Ubah nilai dalam file `metrics.properties` Spark.
`spark-defaults`	Ubah nilai dalam file `spark-defaults.conf` Spark.
`spark-env`	Ubah nilai di lingkungan Spark.
`spark-hive-site`	Ubah nilai dalam file `hive-site.xml` Spark.
`spark-log4j`	Ubah nilai dalam file `log4j2.properties` Spark.
`emr-job-submitter`	Konfigurasi untuk pod pengirim pekerjaan.

Untuk digunakan secara khusus dengan CreateManagedEndpointAPIs:

Klasifikasi	Deskripsi
`jeg-config`	Ubah nilai dalam file Jupyter Enterprise Gateway`jupyter_enterprise_gateway_config.py` .
`jupyter-kernel-overrides`	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering sesuai dengan XML file konfigurasi untuk aplikasi, sepertispark-hive-site.xml. Untuk informasi selengkapnya, lihat Mengkonfigurasi Aplikasi.

Fitur penting

Fitur-fitur berikut disertakan dengan rilis 6.14 Amazon EMR diEKS.

- Dukungan Apache Livy - Amazon EMR EKS sekarang mendukung Apache Livy dengan. spark-submit

emr-6.14.0-terbaru

Catatan rilis: emr-6.14.0-latest saat ini menunjuk ke emr-6.14.0-20231005.

Wilayah: emr-6.14.0-latest tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: emr-6.14.0:latest

emr-6.14.0-20231005

Catatan rilis: 6.14.0-20231005 dirilis pada 17 Oktober 2023. Ini adalah rilis awal Amazon EMR 6.14.0.

Wilayah: emr-6.14.0-20231005 tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: emr-6.14.0:20231005

Amazon EMR pada rilis EKS 6.13.0

Halaman ini menjelaskan fungsionalitas baru dan yang diperbarui untuk Amazon EMR yang khusus untuk EMR Amazon saat EKS penyebaran. Untuk detail tentang Amazon yang EMR berjalan di Amazon EC2 dan tentang rilis Amazon EMR 6.13.0 secara umum, lihat Amazon EMR 6.13.0 di Panduan Rilis Amazon. EMR

Amazon EMR pada rilis EKS 6,13

Rilis Amazon EMR 6.13.0 berikut tersedia untuk Amazon EMR di. EKS Pilih emr-6.13.0- XXXX rilis tertentu untuk melihat detail lebih lanjut seperti tag gambar kontainer terkait.

- emr-6.13.0-terbaru
- emr-6.13.0-20230814
- emr-6.13.0-spark-rapids-latest
- emr-6.13.0-spark-rapids-20230814
- emr-6.13.0-java11-latest
- emr-6.13.0-java11-20230814
- emr-6.13.0-java17-latest
- emr-6.13.0-java17-20230814
- emr-6.13.0-java17-al2023-latest
- emr-6.13.0-java17-al2023-20230814
- emr-6.13.0-spark-rapids-java17-latest
- emr-6.13.0-spark-rapids-java17-20230814
- emr-6.13.0-spark-rapids-java17-al2023-latest
- emr-6.13.0-spark-rapids-java17-al2023-20230814
- notebook-spark/emr-6.13.0-latest
- notebook-spark/emr-6.13.0-20230814
- notebook-spark/emr-6.13.0-spark-rapids-latest
- notebook-spark/emr-6.13.0-spark-rapids-20230814
- notebook-spark/emr-6.13.0-java11-latest
- notebook-spark/emr-6.13.0-java11-20230814
- notebook-spark/emr-6.13.0-java17-latest
- notebook-spark/emr-6.13.0-java17-20230814
- notebook-spark/emr-6.13.0-java17-al2023-latest
- notebook-spark/emr-6.13.0-java17-al2023-20230814
- notebook-python/emr-6.13.0-latest
- notebook-python/emr-6.13.0-20230814

- notebook-python/emr-6.13.0-spark-rapids-latest
- notebook-python/emr-6.13.0-spark-rapids-20230814
- notebook-python/emr-6.13.0-java11-latest
- notebook-python/emr-6.13.0-java11-20230814
- notebook-python/emr-6.13.0-java17-latest
- notebook-python/emr-6.13.0-java17-20230814
- notebook-python/emr-6.13.0-java17-al2023-latest
- notebook-python/emr-6.13.0-java17-al2023-20230814

Catatan rilis

Catatan rilis untuk Amazon EMR di EKS 6.13.0

- Aplikasi yang didukung - AWS SDK for Java 1.12.513, Apache Spark 3.4.1-amzn-0, Apache Hudi 0.13.1-amzn-0, Apache Iceberg 1.3.0-amzn-0, Delta 2.4.0, Apache Spark 23.06.0-amzn-1, Jupyter Enterprise Gateway 2.6.0.amzn RAPIDS
- Komponen yang didukung - `aws-sagemaker-spark-sdkemr-ddb`,`emr-goodies`,,`emr-s3-select`,`emrfs`,`hadoop-client`,`hudi`,`hudi-spark`,`iceberg`,`spark-kubernetes`.
- Klasifikasi konfigurasi yang didukung

  Untuk digunakan dengan StartJobRundan CreateManagedEndpointAPIs:

Klasifikasi	Deskripsi
`core-site`	Ubah nilai dalam file `core-site.xml` Hadoop.
`emrfs-site`	Ubah EMRFS pengaturan.
`spark-metrics`	Ubah nilai dalam file `metrics.properties` Spark.
`spark-defaults`	Ubah nilai dalam file `spark-defaults.conf` Spark.
`spark-env`	Ubah nilai di lingkungan Spark.

Klasifikasi	Deskripsi
`spark-hive-site`	Ubah nilai dalam file `hive-site.xml` Spark.
`spark-log4j`	Ubah nilai dalam file `log4j2.properties` Spark.
`emr-job-submitter`	Konfigurasi untuk pod pengirim pekerjaan.

Untuk digunakan secara khusus dengan CreateManagedEndpointAPIs:

Klasifikasi	Deskripsi
`jeg-config`	Ubah nilai dalam file Jupyter Enterprise Gatewayjupyter_enterprise_gateway_config.py .
`jupyter-kernel-overrides`	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering sesuai dengan XML file konfigurasi untuk aplikasi, sepertispark-hive-site.xml. Untuk informasi selengkapnya, lihat Mengkonfigurasi Aplikasi.

Fitur penting

Fitur-fitur berikut disertakan dengan rilis 6.13 Amazon EMR diEKS.

- Amazon Linux 2023 - Dengan Amazon EMR pada EKS 6.13 dan lebih tinggi, Anda dapat meluncurkan Spark dengan AL2 023 sebagai sistem operasi bersama dengan runtime Java 17. Untuk melakukan ini, gunakan label rilis `al2023` dengan namanya. Sebagai contoh: `emr-6.13.0-java17-al2023-latest`. Kami menyarankan Anda memvalidasi dan menjalankan pengujian kinerja sebelum memindahkan beban kerja produksi ke AL2 023 dan Java 17.

- EMRAmazon EKS dengan Apache Flink (pratinjau publik) - Amazon EMR pada EKS rilis 6.13 dan dukungan yang lebih tinggi Apache Flink, tersedia di pratinjau publik. Dengan peluncuran ini, Anda dapat menjalankan aplikasi berbasis Apache Flink Anda bersama dengan jenis aplikasi lain di cluster Amazon yang sama. EKS Ini membantu meningkatkan pemanfaatan sumber daya dan menyederhanakan manajemen infrastruktur. Jika Anda sudah menjalankan kerangka kerja data besar di AmazonEKS, Anda sekarang dapat membiarkan Amazon EMR mengotomatiskan penyediaan dan pengelolaan Anda.

emr-6.13.0-terbaru

Catatan rilis: `emr-6.13.0-latest` saat ini menunjuk ke `emr-6.13.0-20230814`.

Wilayah: `emr-6.13.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.13.0:latest`

emr-6.13.0-20230814

Catatan rilis: `6.13.0-20230814` dirilis pada 7 September 2023. Ini adalah rilis awal Amazon EMR 6.13.0.

Wilayah: `emr-6.13.0-20230814` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.13.0:20230814`

Amazon EMR pada rilis EKS 6.12.0

Halaman ini menjelaskan fungsionalitas baru dan yang diperbarui untuk Amazon EMR yang khusus untuk EMR Amazon saat EKS penyebaran. Untuk detail tentang Amazon yang EMR berjalan di Amazon EC2 dan tentang rilis Amazon EMR 6.12.0 secara umum, lihat Amazon EMR 6.12.0 di Panduan Rilis Amazon. EMR

Amazon EMR pada rilis EKS 6.12

Rilis Amazon EMR 6.12.0 berikut tersedia untuk Amazon EMR di. EKS Pilih emr-6.12.0- XXXX rilis tertentu untuk melihat detail lebih lanjut seperti tag gambar kontainer terkait.

- emr-6.12.0-terbaru
- emr-6.12.0-20240321
- emr-6.12.0-20230701
- emr-6.12.0- spark-rapids-latest
- emr-6.12.0-spark-cepat-20230701
- emr-6.12.0-java11-terbaru
- emr-6.12.0-java11-20230701
- emr-6.12.0-java17-terbaru
- emr-6.12.0-java17-20230701
- emr-6.12.0- 17-terbaru spark-rapids-java
- emr-6.12.0- 17-20230701 spark-rapids-java
- notebook-spark/emr-6.12.0-terbaru
- notebook-spark/emr-6.12.0-20230701
- notebook-spark/emr-6.12.0- spark-rapids-latest
- notebook-spark/emr-6.12.0-spark-rapids-20230701
- notebook-python/emr-6.12.0-terbaru
- notebook-python/emr-6.12.0-20230701
- notebook-python/emr-6.12.0- spark-rapids-latest
- notebook-python/emr-6.12.0-spark-rapids-20230701

Catatan rilis

Catatan rilis untuk Amazon EMR di EKS 6.12.0

- Aplikasi yang didukung - AWS SDK for Java 1.12.490, Apache Spark 3.4.0-amzn-0, Apache Hudi 0.13.1-amzn-0, Apache Iceberg 1.3.0-amzn-0, Delta 2.4.0, Apache Spark 23.06.0-amzn-0, Jupyter Enterprise Gateway 2.6.0 RAPIDS
- Komponen yang didukung - `aws-sagemaker-spark-sdkemr-ddb`,`emr-goodies`,,`emr-s3-select`,`emrfs`,`hadoop-client`,`hudi`,`hudi-spark`,`iceberg`,`spark-kubernetes`.
- Klasifikasi konfigurasi yang didukung

  Untuk digunakan dengan StartJobRundan CreateManagedEndpointAPIs:

Klasifikasi	Deskripsi
`core-site`	Ubah nilai dalam file `core-site.xml` Hadoop.
`emrfs-site`	Ubah EMRFS pengaturan.
`spark-metrics`	Ubah nilai dalam file `metrics.properties` Spark.
`spark-defaults`	Ubah nilai dalam file `spark-defaults.conf` Spark.
`spark-env`	Ubah nilai di lingkungan Spark.
`spark-hive-site`	Ubah nilai dalam file `hive-site.xml` Spark.
`spark-log4j`	Ubah nilai dalam file `log4j2.properties` Spark.
`emr-job-submitter`	Konfigurasi untuk pod pengirim pekerjaan.

Untuk digunakan secara khusus dengan CreateManagedEndpointAPIs:

Klasifikasi	Deskripsi
`jeg-config`	Ubah nilai dalam file Jupyter Enterprise Gateway`jupyter_enterprise_gateway_config.py` .
`jupyter-kernel-overrides`	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering sesuai dengan XML file konfigurasi untuk aplikasi, sepertispark-hive-site.xml. Untuk informasi selengkapnya, lihat Mengkonfigurasi Aplikasi.

Fitur penting

Fitur-fitur berikut disertakan dengan rilis 6.12 Amazon EMR diEKS.

- Java 17 - Dengan Amazon EMR di EKS 6.12 dan lebih tinggi, Anda dapat meluncurkan Spark dengan runtime Java 17. Untuk melakukan ini, berikan emr-6.12.0-java17-latest sebagai label rilis. Kami menyarankan Anda memvalidasi dan menjalankan pengujian kinerja sebelum memindahkan beban kerja produksi dari versi sebelumnya dari gambar Java ke gambar Java 17.

emr-6.12.0-terbaru

Catatan rilis: emr-6.12.0-latest saat ini menunjuk ke emr-6.12.0-20240321.

Wilayah: emr-6.12.0-latest tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: emr-6.12.0:latest

emr-6.12.0-20240321

Catatan rilis: 6.12.0-20240321 dirilis pada 11 Maret 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: emr-6.12.0-20240321 tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: emr-6.12.0:20240321

emr-6.12.0-20230701

Catatan rilis: 6.12.0-20230701 dirilis pada 1 Juli 2023. Ini adalah rilis awal Amazon EMR 6.12.0.

Wilayah: emr-6.12.0-20230701 tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.12.0:20230701`

Amazon EMR pada rilis EKS 6.11.0

Halaman ini menjelaskan fungsionalitas baru dan yang diperbarui untuk Amazon EMR yang khusus untuk EMR Amazon saat EKS penyebaran. Untuk detail tentang Amazon yang EMR berjalan di Amazon EC2 dan tentang rilis Amazon EMR 6.11.0 secara umum, lihat Amazon EMR 6.11.0 di Panduan Rilis Amazon. EMR

Amazon EMR pada rilis EKS 6.11

Rilis Amazon EMR 6.11.0 berikut tersedia untuk Amazon EMR di. EKS Pilih emr-6.11.0- XXXX rilis tertentu untuk melihat detail lebih lanjut seperti tag gambar kontainer terkait.

- emr-6.11.0-terbaru
- emr-6.11.0-20230905
- emr-6.11.0-20230509

- emr-6.11.0- spark-rapids-latest
- emr-6.11.0-spark-cepat-20230509
- emr-6.11.0-java11-terbaru
- emr-6.11.0-java11-20230509
- notebook-spark/emr-6.11.0-terbaru
- notebook-spark/emr-6.11.0-20230509
- notebook-python/emr-6.11.0-terbaru
- notebook-python/emr-6.11.0-20230509

Catatan rilis

Catatan rilis untuk Amazon EMR di EKS 6.11.0

- Aplikasi yang didukung - AWS SDK for Java 1.12.446, Apache Spark 3.3.2-amzn-0, Apache Hudi 0.13.0-amzn-0, Apache Iceberg 1.2.0-amzn-0, Delta 2.2.0, Apache Spark 23.02.0-amzn-0, Jupyter Enterprise Gateway 2.6.0 RAPIDS

- Komponen yang didukung - `aws-sagemaker-spark-sdkemr-ddb`,`emr-goodies`,,`emr-s3-select`,`emrfs`,`hadoop-client`,`hudi`,`hudi-spark`,`iceberg`,`spark-kubernetes`.
- Klasifikasi konfigurasi yang didukung

  Untuk digunakan dengan StartJobRundan CreateManagedEndpointAPIs:

Klasifikasi	Deskripsi
`core-site`	Ubah nilai dalam file `core-site.xml` Hadoop.
`emrfs-site`	Ubah EMRFS pengaturan.
`spark-metrics`	Ubah nilai dalam file `metrics.properties` Spark.
`spark-defaults`	Ubah nilai dalam file `spark-defaults.conf` Spark.
`spark-env`	Ubah nilai di lingkungan Spark.
`spark-hive-site`	Ubah nilai dalam file `hive-site.xml` Spark.
`spark-log4j`	Ubah nilai dalam file `log4j.properties` Spark.

  Untuk digunakan secara khusus dengan CreateManagedEndpointAPIs:

Klasifikasi	Deskripsi
`jeg-config`	Ubah nilai dalam file Jupyter Enterprise Gateway`jupyter_enterprise_gateway_config.py` .
`jupyter-kernel-overrides`	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering sesuai dengan XML file konfigurasi untuk aplikasi, sepertispark-hive-site.xml. Untuk informasi selengkapnya, lihat Mengkonfigurasi Aplikasi.

Fitur penting

Fitur-fitur berikut disertakan dengan rilis 6.11 Amazon EMR diEKS.

- Amazon EMR pada gambar EKS dasar di Galeri ECR Publik Amazon — Jika Anda menggunakan kemampuan gambar khusus, gambar dasar kami menyediakan stoples, konfigurasi, dan pustaka penting untuk berinteraksi dengan EMR AmazonEKS. Anda sekarang dapat menemukan gambar dasar di Galeri ECR Publik Amazon.
- Rotasi log kontainer percikan - Amazon EMR di EKS 6.11 mendukung rotasi log kontainer Spark. Anda dapat mengaktifkan kemampuan dengan `containerLogRotationConfiguration` dalam `MonitoringConfiguration` pengoperasian `StartJobRun`API. Anda dapat mengonfigurasi `rotationSize` dan `maxFilestoKeep` untuk menentukan jumlah dan ukuran file log yang Anda inginkan Amazon EKS untuk disimpan EMR di driver Spark dan pod pelaksana. Untuk informasi selengkapnya, lihat Menggunakan rotasi log kontainer Spark.
- Dukungan gunung berapi di operator Spark dan spark-submit — EMR Amazon EKS di 6.11 mendukung menjalankan pekerjaan Spark dengan Volcano sebagai penjadwal kustom Kubernetes di operator Spark dan spark-submit. Anda dapat menggunakan fitur seperti penjadwalan geng, manajemen antrian, preemption, dan penjadwalan berbagi adil untuk mencapai throughput penjadwalan yang tinggi dan kapasitas yang dioptimalkan. Untuk informasi selengkapnya, lihat Menggunakan Volcano sebagai penjadwal khusus untuk Apache Spark di Amazon EMR di EKS.

emr-6.11.0-terbaru

Catatan rilis: `emr-6.11.0-latest` saat ini menunjuk ke `emr-20230905`.

Wilayah: `emr-6.11.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.11.0:latest`

emr-6.11.0-20230905

Catatan rilis: `6.11.0-20230905` dirilis pada 29 September 2023. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.11.0-20230509` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.11.0:20230509`

emr-6.11.0-20230509

Catatan rilis: `6.11.0-20230509` dirilis pada 9 Mei 2023. Ini adalah rilis awal Amazon EMR 6.11.0.

Wilayah: `emr-6.11.0-20230509` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.11.0:20230509`

Amazon EMR pada rilis EKS 6.10.0

Rilis Amazon EMR 6.10.0 berikut tersedia untuk Amazon EMR di. EKS Pilih emr-6.10.0- XXXX rilis tertentu untuk melihat detail lebih lanjut seperti tag gambar kontainer terkait.

- emr-6.10.0-terbaru
- emr-6.10.0-20230905
- emr-6.10.0-20230624
- emr-6.10.0-20230421
- emr-6.10.0-20230403
- emr-6.10.0-20230220
- emr-6.10.0- spark-rapids-latest
- emr-6.10.0-spark-cepat-20230624
- emr-6.10.0-spark-cepat-20230220
- emr-6.10.0-java11-terbaru
- emr-6.10.0-java11-20230624
- emr-6.10.0-java11-20230220

- notebook-spark/emr-6.10.0-terbaru
- notebook-spark/emr-6.10.0-20230624
- notebook-spark/emr-6.10.0-20230220
- notebook-python/emr-6.10.0-terbaru
- notebook-python/emr-6.10.0-20230624
- notebook-python/emr-6.10.0-20230220

Catatan rilis untuk Amazon EMR 6.10.0

- Aplikasi yang didukung - AWS SDK for Java 1.12.397, Spark 3.3.1-amzn-0, Hudi 0.12.2-amzn-0, Iceberg 1.1.0-amzn-0, Delta 2.2.0.
- Komponen yang didukung - `aws-sagemaker-spark-sdkemr-ddb`,`emr-goodies`,,`emr-s3-select`,`emrfs`,`hadoop-client`,`hudi`,`hudi-spark`,`iceberg`,`spark-kubernetes`.
- Klasifikasi konfigurasi yang didukung:

  Untuk digunakan dengan StartJobRundan CreateManagedEndpointAPIs:

Klasifikasi	Deskripsi
`core-site`	Ubah nilai dalam file Hadoop. `core-site.xml`
`emrfs-site`	Ubah EMRFS pengaturan.
`spark-metrics`	Ubah nilai dalam `metrics.properties` file Spark.
`spark-defaults`	Ubah nilai dalam `spark-defaults.conf` file Spark.
`spark-env`	Ubah nilai di lingkungan Spark.
`spark-hive-site`	Ubah nilai dalam `hive-site.xml` file Spark.
`spark-log4j`	Ubah nilai dalam `log4j.properties` file Spark.

Untuk digunakan secara khusus dengan CreateManagedEndpointAPIs:

Klasifikasi	Deskripsi
`jeg-config`	Ubah nilai dalam file Jupyter Enterprise Gatewayjupyter_enterprise_gateway_config.py .
`jupyter-kernel-overrides`	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering sesuai dengan XML file konfigurasi untuk aplikasi, sepertispark-hive-site.xml. Untuk informasi selengkapnya, lihat Mengkonfigurasi Aplikasi.

Fitur penting

- Operator Spark - Dengan Amazon EMR di EKS 6.10.0 dan yang lebih tinggi, Anda dapat menggunakan operator Kubernetes untuk Apache Spark, atau operator Spark, untuk menyebarkan dan mengelola aplikasi Spark dengan runtime rilis Amazon di cluster Amazon Anda sendiri. EMR EKS Untuk informasi selengkapnya, lihat Menjalankan pekerjaan Spark dengan operator Spark.
- Java 11 - Dengan Amazon EMR di EKS 6.10 dan lebih tinggi, Anda dapat meluncurkan Spark dengan runtime Java 11. Untuk melakukan ini, berikan `emr-6.10.0-java11-latest` sebagai label rilis. Sebaiknya Anda memvalidasi dan menjalankan pengujian kinerja sebelum memindahkan beban kerja produksi dari gambar Java 8 ke gambar Java 11.
- Untuk integrasi Amazon Redshift untuk Apache Spark, EMR Amazon EKS di 6.10.0 menghapus dependensi`minimal-json.jar`, dan secara otomatis menambahkan jar terkait yang `spark-redshift` diperlukan ke jalur kelas pelaksana untuk Spark:,, dan. `spark-redshift.jar` `spark-avro.jar` `RedshiftJDBC.jar`

Perubahan

- EMRFSKomitter yang dioptimalkan S3 sekarang diaktifkan secara default untuk parket,ORC, dan format berbasis teks (termasuk dan). CSV JSON

emr-6.10.0-terbaru

Catatan rilis: `emr-6.10.0-latest` saat ini menunjuk ke `emr-6.10.0-20230905`.

Wilayah: `emr-6.10.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.10.0:latest`

emr-6.10.0-20230905

Catatan rilis: `6.10.0-20230905` dirilis pada 29 September 2023. Dibandingkan dengan rilis sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.10.0-20230905` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.10.0:20230905`

emr-6.10.0-20230624

Catatan rilis: `6.10.0-20230624` dirilis pada 7 Juli 2023. Dibandingkan dengan rilis sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.10.0-20230624` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.10.0:20230624`

emr-6.10.0-20230421

Catatan rilis: `6.10.0-20230421` dirilis pada 28 April 2023. Dibandingkan dengan rilis sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.10.0-20230421` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.10.0:20230421`

emr-6.10.0-20230403

Catatan rilis: `6.10.0-20230403` dirilis pada 12 April 2023. Dibandingkan dengan rilis sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.10.0-20230403` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.10.0:20230403`

emr-6.10.0-20230220

Catatan rilis: `emr-6.10.0-20230220` dirilis pada 20 Februari 2023. Ini adalah rilis awal Amazon EMR 6.10.0.

Wilayah: `emr-6.10.0-20230220` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.10.0:20230220`

Amazon EMR pada rilis EKS 6.9.0

Rilis Amazon EMR 6.9.0 berikut tersedia untuk Amazon EMR di. EKS Pilih emr-6.9.0- XXXX rilis tertentu untuk melihat detail lebih lanjut seperti tag gambar kontainer terkait.

- emr-6.9.0-terbaru
- emr-6.9.0-20230905
- emr-6.9.0-20230624
- emr-6.9.0-20221108
- emr-6.9.0- spark-rapids-latest
- emr-6.9.0-spark-cepat-20230624
- emr-6.9.0-spark-cepat-20221108
- notebook-spark/emr-6.9.0-terbaru
- notebook-spark/emr-6.9.0-20230624
- notebook-spark/emr-6.9.0-20221108
- notebook-python/emr-6.9.0-terbaru

- notebook-python/emr-6.9.0-20230624
- notebook-python/emr-6.9.0-20221108

Catatan rilis untuk Amazon EMR 6.9.0

- Aplikasi yang didukung - AWS SDK for Java 1.12.331, Spark 3.3.0-amzn-1, Hudi 0.12.1-amzn-0, Iceberg 0.14.1-amzn-0, Delta 2.1.0.
- Komponen yang didukung - `aws-sagemaker-spark-sdkemr-ddb`,`emr-goodies`,,`emr-s3-select`,`emrfs`,`hadoop-client`,`hudi`,`hudi-spark`,`iceberg`,`spark-kubernetes`.
- Klasifikasi konfigurasi yang didukung:

  Untuk digunakan dengan StartJobRundan CreateManagedEndpointAPIs:

Klasifikasi	Deskripsi
`core-site`	Ubah nilai dalam file core-site.xml Hadoop.
`emrfs-site`	Ubah EMRFS pengaturan.
`spark-metrics`	Ubah nilai dalam file metrics.properties Spark.
`spark-defaults`	Ubah nilai dalam file spark-defaults.conf Spark.
`spark-env`	Ubah nilai di lingkungan Spark.
`spark-hive-site`	Ubah nilai dalam file hive-site.xml Spark.
`spark-log4j`	Ubah nilai dalam file log4j.properties Spark.

  Untuk digunakan secara khusus dengan CreateManagedEndpointAPIs:

Klasifikasi	Deskripsi
`jeg-config`	Ubah nilai dalam file Jupyter Enterpris e Gateway`jupyter_enterprise _gateway_config.py` .

Klasifikasi	Deskripsi
`jupyter-kernel-overrides`	Ubah nilai untuk Gambar Kernel di file Spesifikasi Kernel Jupyter.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering sesuai dengan XML file konfigurasi untuk aplikasi, sepertis`park-hive-site.xml`. Untuk informasi selengkapnya, lihat Mengkonfigurasi Aplikasi.

Fitur penting

- RAPIDSAkselerator Nvidia untuk Apache Spark - EMR Amazon EKS akan mempercepat Spark menggunakan tipe instans unit pemrosesan EC2 grafis ()GPU. Untuk menggunakan gambar Spark dengan RAPIDS Accelerator, tentukan label rilis sebagai emr-6.9.0-. spark-rapids-latest Kunjungi halaman dokumentasi untuk mempelajari lebih lanjut.
- Konektor Spark-Redshift - Integrasi Amazon Redshift untuk Apache Spark disertakan dalam rilis Amazon 6.9.0 dan yang lebih baru. EMR Sebelumnya alat open-source, integrasi asli adalah konektor Spark yang dapat Anda gunakan untuk membangun aplikasi Apache Spark yang membaca dan menulis ke data di Amazon Redshift dan Amazon Redshift Serverless. Untuk informasi selengkapnya, lihat Menggunakan integrasi Amazon Redshift untuk Apache Spark di Amazon EMR di EKS.
- Delta Lake - Delta Lake adalah format penyimpanan sumber terbuka yang memungkinkan pembangunan danau data dengan konsistensi transaksional, definisi kumpulan data yang konsisten, perubahan evolusi skema, dan dukungan mutasi data. Kunjungi Menggunakan Danau Delta untuk mempelajari lebih lanjut.
- Ubah PySpark parameter - Titik akhir interaktif sekarang mendukung modifikasi parameter Spark yang terkait dengan PySpark sesi di EMR Studio Jupyter Notebook. Kunjungi Memodifikasi parameter PySpark sesi untuk mempelajari lebih lanjut.

Masalah terselesaikan

- Saat Anda menggunakan konektor DynamoDB dengan Spark di EMR Amazon versi 6.6.0, 6.7.0, dan 6.8.0, semua pembacaan dari tabel Anda mengembalikan hasil kosong, meskipun pemisahan input mereferensikan data yang tidak kosong. Amazon EMR rilis 6.9.0 memperbaiki masalah ini.

- Amazon EMR di EKS 6.8.0 salah mengisi hash build di metadata file Parquet yang dihasilkan menggunakan Apache Spark. Masalah ini dapat menyebabkan alat yang mengurai string versi metadata dari file Parket yang dihasilkan oleh Amazon EMR di EKS 6.8.0 gagal.

Masalah yang diketahui

- Jika Anda menggunakan integrasi Amazon Redshift untuk Apache Spark dan memiliki waktu, jadwal, stempel waktu, atau timestamptz dengan presisi mikrodetik dalam format Parket, konektor membulatkan nilai waktu ke nilai milidetik terdekat. Sebagai solusinya, gunakan parameter format bongkar teks. `unload_s3_format`

emr-6.9.0-terbaru

Catatan rilis: `emr-6.9.0-latest` saat ini menunjuk ke `emr-6.9.0-20230905`.

Wilayah: `emr-6.9.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.9.0:latest`

emr-6.9.0-20230905

Catatan rilis:`emr-6.9.0-20230905`. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.9.0-20230905` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.9.0:20230905`

emr-6.9.0-20230624

Catatan rilis: `emr-6.9.0-20230624` dirilis pada 7 Juli 2023.

Wilayah: `emr-6.9.0-20230624` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.9.0:20230624`

emr-6.9.0-20221108

Catatan rilis: `emr-6.9.0-20221108` dirilis pada 08 Desember 2022. Ini adalah rilis awal Amazon EMR 6.9.0.

Wilayah: `emr-6.9.0-20221108` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.9.0:20221108`

Amazon EMR pada rilis EKS 6.8.0

Rilis Amazon EMR 6.8.0 berikut tersedia untuk Amazon EMR di. EKS Pilih emr-6.8.0- XXXX rilis tertentu untuk melihat detail lebih lanjut seperti tag gambar kontainer terkait.

- emr-6.8.0-terbaru
- emr-6.8.0-20230905
- emr-6.8.0-20230624
- emr-6.8.0-20221219
- emr-6.8.0-20220802

Catatan rilis untuk Amazon EMR 6.8.0

- Aplikasi yang didukung - AWS SDK for Java 1.12.170, Spark 3.3.0-amzn-0, Hudi 0.11.1-amzn-0, Iceberg 0.14.0-amzn-0.
- Komponen yang didukung - `aws-sagemaker-spark-sdkemr-ddb`,`emr-goodies`,,`emr-s3-select`,`emrfs`,`hadoop-client`,`hudi`,`hudi-spark`,`iceberg`,`spark-kubernetes`.
- Klasifikasi konfigurasi yang didukung:

Klasifikasi	Deskripsi
`core-site`	Ubah nilai dalam file core-site.xml Hadoop.
`emrfs-site`	Ubah EMRFS pengaturan.
`spark-metrics`	Ubah nilai dalam file metrics.properties Spark.

Klasifikasi	Deskripsi
`spark-defaults`	Ubah nilai dalam file spark-defaults.conf Spark.
`spark-env`	Ubah nilai di lingkungan Spark.
`spark-hive-site`	Ubah nilai dalam file hive-site.xml Spark.
`spark-log4j`	Ubah nilai dalam file log4j.properties Spark.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering sesuai dengan XML file konfigurasi untuk aplikasi, seperti`spark-hive-site.xml`. Untuk informasi selengkapnya, lihat Mengkonfigurasi Aplikasi.

Fitur penting

- Spark3.3.0 - Amazon EMR di EKS 6.8 menyertakan Spark 3.3.0, yang mendukung penggunaan label pemilih node terpisah untuk pod pelaksana driver Spark. Label baru ini memungkinkan Anda untuk menentukan tipe node untuk driver dan pod eksekutor secara terpisah di StartJobRun API, tanpa menggunakan templat pod.
  - Properti pemilih node driver: spark.kubernetes.driver.node.selector. [labelKey]
  - Properti pemilih node pelaksana: spark.kubernetes.executor.node.selector. [labelKey]
- Pesan kegagalan pekerjaan yang disempurnakan - Rilis ini memperkenalkan konfigurasi `spark.stage.extraDetailsOnFetchFailures.enabled` dan `spark.stage.extraDetailsOnFetchFailures.maxFailuresToInclude` melacak kegagalan tugas karena kode pengguna. Detail ini akan digunakan untuk meningkatkan pesan kegagalan yang ditampilkan di log driver ketika tahap dibatalkan karena kegagalan pengambilan acak.

Nama properti	Nilai default	Arti	Sejak versi
`spark.stage.extraDetailsOnF`	false	Jika disetel ke`true`, properti ini digunakan untuk menyempurnakan pesan	emr-6.8

Nama properti	Nilai default	Arti	Sejak versi
`etchFailu res.enabled`		kegagalan pekerjaan yang ditampilkan di log driver saat tahap dibatalkan karena Kegagalan Pengambilan Aduk. Secara default, 5 kegagalan tugas terakhir yang disebabkan oleh kode pengguna dilacak, dan pesan kesalahan kegagalan ditambahkan di Log Driver. Untuk meningkat kan jumlah kegagalan tugas dengan pengecual ian pengguna untuk dilacak, lihat konfigura sispark.sta ge.extraD etailsOnF etchFailu res.maxFa iluresToI nclude .	

Nama properti	Nilai default	Arti	Sejak versi
`spark.stage.extraDetailsOnFetchFailures.maxFailuresToInclude`	5	Jumlah kegagalan tugas untuk melacak per tahap dan upaya. Properti ini digunakan untuk menyempurnakan pesan kegagalan pekerjaan dengan pengecualian pengguna yang ditampilkan di log driver saat tahap dibatalkan karena Kegagalan Pengambilan Aduk. Properti ini hanya berfungsi jika Config spark.stage.extraDetailsOnFetchFailures.enabled disetel ke true.	emr-6.8

Untuk informasi selengkapnya, lihat dokumentasi konfigurasi Apache Spark.

Masalah yang diketahui

- Amazon EMR di EKS 6.8.0 salah mengisi hash build di metadata file Parquet yang dihasilkan menggunakan Apache Spark. Masalah ini dapat menyebabkan alat yang mengurai string versi metadata dari file Parket yang dihasilkan oleh Amazon EMR di EKS 6.8.0 gagal. Pelanggan yang mengurai string versi dari metadata Parket dan bergantung pada hash build harus beralih ke EMR versi Amazon yang berbeda dan menulis ulang file.

Masalah terselesaikan

- Kemampuan Kernel interupsi untuk pySpark kernel - Dalam proses beban kerja interaktif yang dipicu oleh mengeksekusi sel di notebook dapat dihentikan dengan menggunakan kemampuan tersebut. `Interrupt Kernel` Perbaikan telah diperkenalkan sehingga fungsi ini berfungsi untuk pySpark kernel. Ini juga tersedia di open source di Changes untuk menangani interupsi untuk PySpark Kubernetes Kernel #1115.

emr-6.8.0-terbaru

Catatan rilis: `emr-6.8.0-latest` saat ini menunjuk ke `emr-6.8.0-20230624`.

Wilayah: `emr-6.8.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.8.0:latest`

emr-6.8.0-20230905

Catatan rilis: `emr-6.8.0-20230905` dirilis pada 29 September 2023. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.8.0-20230905` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.8.0:20230905`

emr-6.8.0-20230624

Catatan rilis: `emr-6.8.0-20230624` dirilis pada 7 Juli 2023. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.8.0-20230624` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.8.0:20230624`

emr-6.8.0-20221219

Catatan rilis: `emr-6.8.0-20221219` dirilis pada 19 Jan 2023. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.8.0-20221219` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.8.0:20221219`

emr-6.8.0-20220802

Catatan rilis: `emr-6.8.0-20220802` dirilis pada 27 Sep 2022. Ini adalah rilis awal Amazon EMR 6.8.0.

Wilayah: `emr-6.8.0-20220802` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.8.0:20220802`

Amazon EMR pada rilis EKS 6.7.0

Rilis Amazon EMR 6.7.0 berikut tersedia untuk Amazon EMR di. EKS Pilih emr-6.7.0- XXXX rilis tertentu untuk melihat detail lebih lanjut seperti tag gambar kontainer terkait.

- emr-6.7.0-terbaru
- emr-6.7.0-20240321
- emr-6.7.0-20230624
- emr-6.7.0-20221219
- emr-6.7.0-20220630

Catatan rilis untuk Amazon EMR 6.7.0

- Aplikasi yang didukung - Spark 3.2.1-amzn-0, Jupyter Enterprise Gateway 2.6, Hudi 0.11-amzn-0, Iceberg 0.13.1.
- Komponen yang didukung - `aws-hm-client` (Konektor Glue), `aws-sagemaker-spark-sdk`, `emr-s3-select`, `emrfs`, `emr-ddb`, `hudi-spark`.
- Dengan upgrade ke JEG 2.6, manajemen kernel sekarang asinkron, yang berarti bahwa JEG tidak memblokir transaksi ketika peluncuran kernel sedang berlangsung. Ini sangat meningkatkan pengalaman pengguna dengan memberikan yang berikut:
  - kemampuan untuk menjalankan perintah di notebook yang sedang berjalan saat peluncuran kernel lainnya sedang berlangsung

  - kemampuan untuk meluncurkan beberapa kernel secara bersamaan tanpa memengaruhi kernel yang sudah berjalan
- Klasifikasi konfigurasi yang didukung:

Klasifikasi	Deskripsi
`core-site`	Ubah nilai dalam file Hadoop`core-site.xml` .
`emrfs-site`	Ubah EMRFS pengaturan.
`spark-metrics`	Ubah nilai dalam `metrics.properties` file Spark.
`spark-defaults`	Ubah nilai dalam `spark-defaults.conf` file Spark.
`spark-env`	Ubah nilai di lingkungan Spark.
`spark-hive-site`	Ubah nilai dalam `hive-site.xml` file Spark.
`spark-log4j`	Ubah nilai dalam `log4j.properties` file Spark.

  Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering sesuai dengan XML file konfigurasi untuk aplikasi, seperti`spark-hive-site.xml`. Untuk informasi lebih lanjut, lihat Mengonfigurasi Aplikasi.

Masalah terselesaikan

- Amazon EMR di EKS 6.7 memperbaiki masalah di 6.6 saat menggunakan fungsionalitas templat pod Apache Spark dengan titik akhir interaktif. Masalah ini hadir di Amazon EMR pada EKS rilis 6.4, 6.5 dan 6.6. Anda sekarang dapat menggunakan templat pod untuk menentukan bagaimana driver Spark dan pod pelaksana Anda dimulai saat menggunakan titik akhir interaktif untuk menjalankan analitik interaktif.

- Di Amazon sebelumnya EMR pada EKS rilis, Jupyter Enterprise Gateway akan memblokir transaksi saat peluncuran kernel sedang berlangsung, dan ini menghambat eksekusi sesi notebook yang sedang berjalan. Anda sekarang dapat menjalankan perintah di notebook yang sedang berjalan saat peluncuran kernel lainnya sedang berlangsung. Anda juga dapat meluncurkan beberapa kernel secara bersamaan tanpa risiko kehilangan konektivitas ke kernel yang sudah berjalan.

emr-6.7.0-terbaru

Catatan rilis: `emr-6.7.0-latest` saat ini menunjuk ke `emr-6.7.0-20240321`.

Wilayah: `emr-6.7.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.7.0:latest`

emr-6.7.0-20240321

Catatan rilis: `emr-6.7.0-20240321` dirilis pada 11 Maret 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.7.0-20240321` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.7.0:20240321`

emr-6.7.0-20230624

Catatan rilis: `emr-6.7.0-20230624` dirilis pada 7 Juli 2023. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.7.0-20230624` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.7.0:20230624`

emr-6.7.0-20221219

Catatan rilis: `emr-6.7.0-20221219` dirilis pada 19 Januari 2023. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.7.0-20221219` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.7.0:20221219`

emr-6.7.0-20220630

Catatan rilis: `emr-6.7.0-20220630` dirilis pada 12 Juli 2022. Ini adalah rilis awal Amazon EMR 6.7.0.

Wilayah: `emr-6.7.0-20220630` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.7.0:20220630`

Amazon EMR pada rilis EKS 6.6.0

Rilis Amazon EMR 6.6.0 berikut tersedia untuk Amazon EMR di. EKS Pilih emr-6.6.0- XXXX rilis tertentu untuk melihat detail lebih lanjut seperti tag gambar kontainer terkait.

- emr-6.6.0-terbaru
- emr-6.6.0-20240321
- emr-6.6.0-20230624
- emr-6.6.0-20221219
- emr-6.6.0-20220411

Catatan rilis untuk Amazon EMR 6.6.0

- Aplikasi yang didukung - Spark 3.2.0-amzn-0, Jupyter Enterprise Gateway (titik akhir, pratinjau publik), Hudi 0.10.1-amzn-0, Iceberg 0.13.1.
- Komponen yang didukung - `aws-hm-client` (Konektor Glue), `aws-sagemaker-spark-sdk`, `emr-s3-select`, `emrfs`, `emr-ddb`, `hudi-spark`.

- Klasifikasi konfigurasi yang didukung:

Klasifikasi	Deskripsi
`core-site`	Ubah nilai dalam file core-site.xml Hadoop.
`emrfs-site`	Ubah EMRFS pengaturan.
`spark-metrics`	Ubah nilai dalam file metrics.properties Spark.
`spark-defaults`	Ubah nilai dalam file spark-defaults.conf Spark.
`spark-env`	Ubah nilai di lingkungan Spark.
`spark-hive-site`	Ubah nilai dalam file hive-site.xml Spark.
`spark-log4j`	Ubah nilai dalam file log4j.properties Spark.

  Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering sesuai dengan XML file konfigurasi untuk aplikasi, spark-hive-site seperti.xml.xml. Untuk informasi lebih lanjut, lihat Mengonfigurasi Aplikasi.

Masalah yang diketahui

- Fungsionalitas template Spark pod dengan titik akhir interaktif tidak berfungsi EMR di Amazon pada EKS rilis 6.4, 6.5, dan 6.6.

Masalah terselesaikan

- Log endpoint interaktif diunggah ke Cloudwatch dan S3.

emr-6.6.0-terbaru

Catatan rilis: `emr-6.6.0-latest` saat ini menunjuk ke `emr-6.6.0-20240321`.

Wilayah: `emr-6.6.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.6.0:latest`

emr-6.6.0-20240321

Catatan rilis: `emr-6.6.0-20240321` dirilis pada 11 Maret 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.6.0-20240321` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.6.0:20240321`

emr-6.6.0-20230624

Catatan rilis: `emr-6.6.0-20230624` dirilis pada 27 Jan 2023. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.6.0-20230624` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.6.0:20230624`

emr-6.6.0-20221219

Catatan rilis: `emr-6.6.0-20221219` dirilis pada 27 Jan 2023. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.6.0-20221219` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.6.0:20221219`

emr-6.6.0-20220411

Catatan rilis: `emr-6.6.0-20220411` dirilis pada 20 Mei 2022. Ini adalah rilis awal Amazon EMR 6.6.0.

Wilayah: `emr-6.6.0-20220411` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.6.0:20220411`

Amazon EMR pada rilis EKS 6.5.0

Rilis Amazon EMR 6.5.0 berikut tersedia untuk Amazon EMR diEKS. Pilih emr-6.5.0- XXXX rilis tertentu untuk melihat detail lebih lanjut seperti tag gambar kontainer terkait.

- emr-6.5.0-terbaru
- emr-6.5.0-20240321
- emr-6.5.0-20221219
- emr-6.5.0-20220802
- emr-6.5.0-20211119

Catatan rilis untuk Amazon EMR 6.5.0

- Aplikasi yang didukung - Spark 3.1.2-amzn-1, Jupyter Enterprise Gateway (titik akhir, pratinjau publik).
- Komponen yang didukung - `aws-hm-client` (Konektor Glue), `aws-sagemaker-spark-sdk`, `emr-s3-select`, `emrfs`, `emr-ddb`, `hudi-spark`.
- Klasifikasi konfigurasi yang didukung:

Klasifikasi	Deskripsi
`core-site`	Ubah nilai dalam file core-site.xml Hadoop.
`emrfs-site`	Ubah EMRFS pengaturan.
`spark-metrics`	Ubah nilai dalam file metrics.properties Spark.
`spark-defaults`	Ubah nilai dalam file spark-defaults.conf Spark.
`spark-env`	Ubah nilai di lingkungan Spark.
`spark-hive-site`	Ubah nilai dalam file hive-site.xml Spark.
`spark-log4j`	Ubah nilai dalam file log4j.properties Spark.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering sesuai dengan XML file konfigurasi untuk aplikasi, spark-hive-site seperti.xml.xml. Untuk informasi lebih lanjut, lihat Mengonfigurasi Aplikasi.

Masalah yang Diketahui

- Fungsionalitas template Spark pod dengan titik akhir interaktif tidak berfungsi EMR di Amazon pada EKS rilis 6.4 dan 6.5.

emr-6.5.0-terbaru

Catatan rilis: `emr-6.5.0-latest` saat ini menunjuk ke `emr-6.5.0-20240321`.

Wilayah: `emr-6.5.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.5.0:latest`

emr-6.5.0-20240321

Catatan rilis: `emr-6.5.0-20240321` dirilis pada 11 Maret 2024. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.5.0-20240321` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.5.0:20240321`

emr-6.5.0-20221219

Catatan rilis: `emr-6.5.0-20221219` dirilis pada 19 Jan 2023. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.5.0-20221219` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.5.0:20221219`

emr-6.5.0-20220802

Catatan rilis: `emr-6.5.0-20220802` dirilis pada 24 Agustus 2022. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui.

Wilayah: `emr-6.5.0-20220802` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.5.0:20220802`

emr-6.5.0-20211119

Catatan rilis: `emr-6.5.0-20211119` dirilis pada 20 Jan 2022. Ini adalah rilis awal Amazon EMR 6.5.0.

Wilayah: `emr-6.5.0-20211119` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.5.0:20211119`

Amazon EMR pada rilis EKS 6.4.0

Rilis Amazon EMR 6.4.0 berikut tersedia untuk Amazon EMR diEKS. Pilih emr-6.4.0- XXXX rilis tertentu untuk melihat detail lebih lanjut seperti tag gambar kontainer terkait.

- emr-6.4.0-terbaru
- emr-6.4.0-20240321
- emr-6.4.0-20221219
- emr-6.4.0-20210830

Catatan rilis untuk Amazon EMR 6.4.0

- Aplikasi yang didukung - Spark 3.1.2-amzn-0, Jupyter Enterprise Gateway (titik akhir, pratinjau publik).
- Komponen yang didukung - `aws-hm-client` (Konektor Glue), `aws-sagemaker-spark-sdk`, `emr-s3-select`, `emrfs`, `emr-ddb`, `hudi-spark`.

- Klasifikasi konfigurasi yang didukung:

Klasifikasi	Deskripsi
`core-site`	Ubah nilai dalam file core-site.xml Hadoop.
`emrfs-site`	Ubah EMRFS pengaturan.
`spark-metrics`	Ubah nilai dalam file metrics.properties Spark.
`spark-defaults`	Ubah nilai dalam file spark-defaults.conf Spark.
`spark-env`	Ubah nilai di lingkungan Spark.
`spark-hive-site`	Ubah nilai dalam file hive-site.xml Spark.
`spark-log4j`	Ubah nilai dalam file log4j.properties Spark.

  Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering sesuai dengan XML file konfigurasi untuk aplikasi, spark-hive-site seperti.xml.xml. Untuk informasi lebih lanjut, lihat Mengonfigurasi Aplikasi.

Masalah yang diketahui

- Fungsionalitas template Spark pod dengan titik akhir interaktif tidak berfungsi EMR di Amazon pada EKS rilis 6.4.

emr-6.4.0-terbaru

Catatan rilis: `emr-6.4.0-latest` saat ini menunjuk ke `emr-6.4.0-20240321`.

Wilayah: `emr-6.4.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.4.0:latest`

emr-6.4.0-20240321

Catatan rilis: `emr-6.4.0-20240321` dirilis pada 11 Maret 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.4.0-20240321` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.4.0:20240321`

emr-6.4.0-20221219

Catatan rilis: `emr-6.4.0-20221219` dirilis pada 27 Jan 2023. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru saja ditambahkan.

Wilayah: `emr-6.4.0-20221219` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.4.0:20221219`

emr-6.4.0-20210830

Catatan rilis: `emr-6.4.0-20210830` dirilis pada 9 Desember 2021. Ini adalah rilis awal Amazon EMR 6.4.0.

Wilayah: `emr-6.4.0-20210830` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.4.0:20210830`

Amazon EMR pada rilis EKS 6.3.0

Rilis Amazon EMR 6.3.0 berikut tersedia untuk Amazon EMR diEKS. Pilih emr-6.3.0- XXXX rilis tertentu untuk melihat detail lebih lanjut seperti tag gambar kontainer terkait.

- emr-6.3.0-terbaru
- emr-6.3.0-20240321
- emr-6.3.0-20220802

- emr-6.3.0-20211008
- emr-6.3.0-20210802
- emr-6.3.0-20210429

Catatan rilis untuk Amazon EMR 6.3.0

- Fitur baru - Dimulai dengan Amazon EMR 6.3.0 dalam seri rilis 6.x, EMR Amazon EKS mendukung fitur template pod Spark. Anda juga dapat mengaktifkan fitur rotasi log peristiwa Spark untuk EMR AmazonEKS. Untuk informasi selengkapnya, silakan lihat Menggunakan templat pod dan Menggunakan rotasi log peristiwa Spark.
- Aplikasi yang didukung - Spark 3.1.1-amzn-0, Jupyter Enterprise Gateway (titik akhir, pratinjau publik).
- Komponen yang didukung - `aws-hm-client` (Konektor Glue), `aws-sagemaker-spark-sdk`, `emr-s3-select`, `emrfs`, `emr-ddb`, `hudi-spark`.
- Klasifikasi konfigurasi yang didukung:

Klasifikasi	Deskripsi
`core-site`	Ubah nilai dalam file core-site.xml Hadoop.
`emrfs-site`	Ubah EMRFS pengaturan.
`spark-metrics`	Ubah nilai dalam file metrics.properties Spark.
`spark-defaults`	Ubah nilai dalam file spark-defaults.conf Spark.
`spark-env`	Ubah nilai di lingkungan Spark.
`spark-hive-site`	Ubah nilai dalam file hive-site.xml Spark.
`spark-log4j`	Ubah nilai dalam file log4j.properties Spark.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering sesuai dengan XML file konfigurasi untuk aplikasi, spark-hive-site seperti.xml.xml. Untuk informasi lebih lanjut, lihat Mengonfigurasi Aplikasi.

emr-6.3.0-terbaru

Catatan rilis: `emr-6.3.0-latest` saat ini menunjuk ke `emr-6.3.0-20240321`.

Wilayah: `emr-6.3.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.3.0:latest`

emr-6.3.0-20240321

Catatan rilis: `emr-6.3.0-20240321` dirilis pada 11 Maret 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.3.0-20240321` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.3.0:20240321`

emr-6.3.0-20220802

Catatan rilis: `emr-6.3.0-20220802` dirilis pada 27 Sep 2022. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui.

Wilayah: `emr-6.3.0-20220802` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.3.0:20220802`

emr-6.3.0-20211008

Catatan rilis: `emr-6.3.0-20211008` dirilis pada 9 Desember 2021. Dibandingkan dengan versi sebelumnya, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-6.3.0-20211008` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.3.0:20211008`

emr-6.3.0-20210802

Catatan rilis: `emr-6.3.0-20210802` dirilis pada 2 Agustus 2021. Dibandingkan dengan versi sebelumnya, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-6.3.0-20210802` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.3.0:20210802`

emr-6.3.0-20210429

Catatan rilis: `emr-6.3.0-20210429` dirilis pada tanggal 29 April 2021. Ini adalah rilis awal Amazon EMR 6.3.0.

Wilayah: `emr-6.3.0-20210429` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.3.0:20210429`

Amazon EMR pada rilis EKS 6.2.0

Rilis Amazon EMR 6.2.0 berikut tersedia untuk Amazon EMR diEKS. Pilih emr-6.2.0- XXXX rilis tertentu untuk melihat detail lebih lanjut seperti tag gambar kontainer terkait.

- emr-6.2.0-latest
- emr-6.2.0-20240321
- emr-6.2.0-20220802
- emr-6.2.0-20211008
- emr-6.2.0-20210802
- emr-6.2.0-20210615
- emr-6.2.0-20210129
- emr-6.2.0-20201218
- emr-6.2.0-20201201

Catatan rilis untuk Amazon EMR 6.2.0

- Aplikasi yang didukung - Spark 3.0.1-amzn-0, Jupyter Enterprise Gateway (titik akhir, pratinjau publik).
- Komponen yang didukung - `aws-hm-client` (Konektor Glue), `aws-sagemaker-spark-sdk`, `emr-s3-select`, `emrfs`, `emr-ddb`, `hudi-spark`.
- Klasifikasi konfigurasi yang didukung:

Klasifikasi	Deskripsi
`core-site`	Ubah nilai dalam file core-site.xml Hadoop.
`emrfs-site`	Ubah EMRFS pengaturan.
`spark-metrics`	Ubah nilai dalam file metrics.properties Spark.
`spark-defaults`	Ubah nilai dalam file spark-defaults.conf Spark.
`spark-env`	Ubah nilai di lingkungan Spark.
`spark-hive-site`	Ubah nilai dalam file hive-site.xml Spark.
`spark-log4j`	Ubah nilai dalam file log4j.properties Spark.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering sesuai dengan XML file konfigurasi untuk aplikasi, spark-hive-site seperti.xml.xml. Untuk informasi lebih lanjut, lihat Mengonfigurasi Aplikasi.

emr-6.2.0-latest

Catatan rilis: `emr-6.2.0-latest` saat ini menunjuk ke `emr-6.2.0-20240321`.

Wilayah: `emr-6.2.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.2.0:20240321`

emr-6.2.0-20240321

Catatan rilis: `emr-6.2.0-20240321` dirilis pada 11 Maret 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-6.2.0-20240321` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.2.0:20240321`

emr-6.2.0-20220802

Catatan rilis: `emr-6.2.0-20220802` dirilis pada 27 Sep 2022. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui.

Wilayah: `emr-6.2.0-20220802` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-6.2.0:20220802`

emr-6.2.0-20211008

Catatan rilis: `emr-6.2.0-20211008` dirilis pada 9 Desember 2021. Dibandingkan dengan versi sebelumnya, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-6.2.0-20211008` tersedia di Wilayah Berikut: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Tanda gambar kontainer: `emr-6.2.0:20211008`

emr-6.2.0-20210802

Catatan rilis: `emr-6.2.0-20210802` dirilis pada 2 Agustus 2021. Dibandingkan dengan versi sebelumnya, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-6.2.0-20210802` tersedia di Wilayah Berikut: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Tanda gambar kontainer: `emr-6.2.0:20210802`

emr-6.2.0-20210615

Catatan rilis: `emr-6.2.0-20210615` dirilis pada 15 Juni 2021. Dibandingkan dengan versi sebelumnya, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-6.2.0-20210615` tersedia di Wilayah Berikut: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Tanda gambar kontainer: `emr-6.2.0:20210615`

emr-6.2.0-20210129

Catatan rilis: `emr-6.2.0-20210129` dirilis pada tanggal 29 Januari 2021. Dibandingkan dengan `emr-6.2.0-20201218`, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-6.2.0-20210129` tersedia di Wilayah berikut: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Tanda gambar kontainer: `emr-6.2.0-20210129`

emr-6.2.0-20201218

Catatan rilis: `emr-6.2.0-20201218` dirilis pada tanggal 18 Desember 2020. Dibandingkan dengan `emr-6.2.0-20201201`, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-6.2.0-20201218` tersedia di Wilayah berikut: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Tanda gambar kontainer: `emr-6.2.0-20201218`

emr-6.2.0-20201201

Catatan rilis: `emr-6.2.0-20201201` dirilis pada tanggal 1 Desember 2020. Ini adalah rilis awal Amazon EMR 6.2.0.

Wilayah: `emr-6.2.0-20201201` tersedia di Wilayah Berikut: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Tanda gambar kontainer: `emr-6.2.0-20201201`

Amazon EMR pada rilis EKS 5.36.0

Rilis Amazon EMR 5.36.0 berikut tersedia untuk Amazon di. EMR EKS Pilih emr-5.36.0- XXXX rilis tertentu untuk melihat detail lebih lanjut seperti tag gambar kontainer terkait.

- emr-5.36.0-terbaru
- emr-5.36.0-20240321
- emr-5.36.0-20221219
- emr-5.36.0-20220620
- emr-5.36.0-20220525

Catatan rilis untuk Amazon EMR 5.36.0

- Memperbaiki masalah keamanan log4j2.
- Aplikasi yang didukung - Spark 2.4.8-amzn-2, Jupyter Enterprise Gateway (titik akhir, pratinjau publik; Kernel Scala tidak didukung), livy-0.7.1, fluentd-4.0.0.
- Komponen yang didukung - aws-hm-client,, emr-ddb aws-sagemaker-spark-sdk, emr-goodies, emr-kinesis, kerberos-server.
- Klasifikasi konfigurasi yang didukung:

Klasifikasi	Deskripsi
`core-site`	Ubah nilai dalam file core-site.xml Hadoop.
`emrfs-site`	Ubah EMRFS pengaturan.
`spark-metrics`	Ubah nilai dalam file metrics.properties Spark.
`spark-defaults`	Ubah nilai dalam file spark-defaults.conf Spark.
`spark-env`	Ubah nilai di lingkungan Spark.
`spark-hive-site`	Ubah nilai dalam file hive-site.xml Spark.
`spark-log4j`	Ubah nilai dalam file log4j.properties Spark.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering sesuai dengan XML file konfigurasi untuk aplikasi, spark-hive-site seperti.xml.xml. Untuk informasi lebih lanjut, lihat Mengonfigurasi Aplikasi.

emr-5.36.0-terbaru

Catatan rilis: `emr-5.36.0-latest` saat ini menunjuk ke `emr-5.36.0-20240321`.

Wilayah: `emr-5.36.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-5.36.0:latest`

emr-5.36.0-20240321

Catatan rilis: `emr-5.36.0-20240321` dirilis pada 11 Maret 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-5.36.0-20240321` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-5.36.0:20240321`

emr-5.36.0-20221219

Catatan rilis: `emr-5.36.0-20221219` dirilis pada 27 Jan 2023. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui.

Wilayah: `emr-5.36.0-20221219` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-5.36.0:20221219`

emr-5.36.0-20220620

Catatan rilis: `emr-5.36.0-20220620` dirilis pada 27 Juli 2022. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui.

Wilayah: `emr-5.36.0-20220620` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-5.36.0:20220620`

emr-5.36.0-20220525

Catatan rilis: `emr-5.36.0-20220525` dirilis pada 16 Juni 2022. Ini adalah rilis awal Amazon EMR 5.36.0.

Wilayah: `emr-5.36.0-20220525` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-5.36.0:20220525`

Amazon EMR pada rilis EKS 5.35.0

Rilis Amazon EMR 5.35.0 berikut tersedia untuk Amazon EMR di. EKS Pilih emr-5.35.0- XXXX rilis tertentu untuk melihat detail lebih lanjut seperti tag gambar kontainer terkait.

- emr-5.35.0-terbaru
- emr-5.35.0-20240321
- emr-5.35.0-20221219
- emr-5.35.0-20220802
- emr-5.35.0-20220307

Catatan rilis untuk Amazon EMR 5.35.0

- Memperbaiki masalah keamanan log4j2.
- Aplikasi yang didukung - Spark 2.4.8-amzn-1, Hudi 0.9.0-amzn-2, Jupyter Enterprise Gateway (titik akhir, pratinjau publik; Kernel Scala tidak didukung).
- Komponen yang didukung - aws-hm-client (Konektor Glue),, emr-s3-select aws-sagemaker-spark-sdk, emrfs, emr-ddb, hudi-spark.
- Klasifikasi konfigurasi yang didukung:

Klasifikasi	Deskripsi
`core-site`	Ubah nilai dalam file core-site.xml Hadoop.
`emrfs-site`	Ubah EMRFS pengaturan.
`spark-metrics`	Ubah nilai dalam file metrics.properties Spark.
`spark-defaults`	Ubah nilai dalam file spark-defaults.conf Spark.
`spark-env`	Ubah nilai di lingkungan Spark.
`spark-hive-site`	Ubah nilai dalam file hive-site.xml Spark.
`spark-log4j`	Ubah nilai dalam file log4j.properties Spark.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering sesuai dengan XML file konfigurasi untuk aplikasi, spark-hive-site seperti.xml.xml. Untuk informasi lebih lanjut, lihat Mengonfigurasi Aplikasi.

emr-5.35.0-terbaru

Catatan rilis: `emr-5.35.0-latest` saat ini menunjuk ke `emr-5.35.0-20240321`.

Wilayah: `emr-5.35.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-5.35.0:latest`

emr-5.35.0-20240321

Catatan rilis: `emr-5.35.0-20240321` dirilis pada 11 Maret 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-5.35.0-20240321` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-5.35.0:20240321`

emr-5.35.0-20221219

Catatan rilis: `emr-5.35.0-20221219` dirilis pada 27 Jan 2023. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui.

Wilayah: `emr-5.35.0-20221219` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-5.35.0:20221219`

emr-5.35.0-20220802

Catatan rilis: `emr-5.35.0-20220802` dirilis pada 27 Sep 2022. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui.

Wilayah: `emr-5.35.0-20220802` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-5.35.0:20220802`

emr-5.35.0-20220307

Catatan rilis: `emr-5.35.0-20220307` dirilis pada 30 Maret 2022. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui.

Wilayah: `emr-5.35.0-20220307` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-5.35.0:20220307`

Amazon EMR pada rilis EKS 5.34.0

Rilis Amazon EMR 5.34.0 berikut tersedia untuk Amazon EMR di. EKS Pilih emr-5.34.0- XXXX rilis tertentu untuk melihat detail lebih lanjut seperti tag gambar kontainer terkait.

- emr-5.34.0-terbaru
- emr-5.34.0-20240321

- emr-5.34.0-20220802

Catatan rilis untuk Amazon EMR 5.34.0

- Aplikasi yang didukung - Spark 2.4.8-amzn-0, Jupyter Enterprise Gateway (titik akhir, pratinjau publik; Kernel Scala tidak didukung).
- Komponen yang didukung - `aws-hm-client` (Konektor Glue), `aws-sagemaker-spark-sdk`, `emr-s3-select`, `emrfs`, `emr-ddb`, `hudi-spark`.
- Klasifikasi konfigurasi yang didukung:

Klasifikasi	Deskripsi
`core-site`	Ubah nilai dalam file core-site.xml Hadoop.
`emrfs-site`	Ubah EMRFS pengaturan.
`spark-metrics`	Ubah nilai dalam file metrics.properties Spark.
`spark-defaults`	Ubah nilai dalam file spark-defaults.conf Spark.
`spark-env`	Ubah nilai di lingkungan Spark.
`spark-hive-site`	Ubah nilai dalam file hive-site.xml Spark.
`spark-log4j`	Ubah nilai dalam file log4j.properties Spark.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering sesuai dengan XML file konfigurasi untuk aplikasi, spark-hive-site seperti.xml.xml. Untuk informasi lebih lanjut, lihat Mengonfigurasi Aplikasi.

emr-5.34.0-terbaru

Catatan rilis: `emr-5.34.0-latest` saat ini menunjuk ke `emr-5.34.0-20220802`.

Wilayah: `emr-5.34.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-5.34.0:latest`

emr-5.34.0-20240321

Catatan rilis: `emr-5.34.0-20240321` dirilis pada 11 Maret 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-5.34.0-20240321` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-5.34.0:20240321`

emr-5.34.0-20220802

Catatan rilis: `emr-5.34.0-20220802` dirilis pada 24 Agustus 2022. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui.

Wilayah: `emr-5.34.0-20220802` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-5.34.0:20220802`

emr-5.34.0-20211208

Catatan rilis: `emr-5.34.0-20211208` dirilis pada 20 Jan 2022. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui.

Wilayah: `emr-5.34.0-20211208` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-5.34.0:20211208`

Amazon EMR pada rilis EKS 5.33.0

Rilis Amazon EMR 5.33.0 berikut tersedia untuk Amazon EMR di. EKS Pilih emr-5.33.0- XXXX rilis tertentu untuk melihat detail lebih lanjut seperti tag gambar kontainer terkait.

- emr-5.33.0-latest

- emr-5.33.0-20240321
- emr-5.33.0-20221219
- emr-5.33.0-20220802
- emr-5.33.0-20211008
- emr-5.33.0-20210802
- emr-5.33.0-20210615
- emr-5.33.0-20210323

Catatan rilis untuk Amazon EMR 5.33.0

- Fitur baru - Dimulai dengan Amazon EMR 5.33.0 dalam seri rilis 5.x, EMR Amazon EKS mendukung fitur template pod Spark. Untuk informasi selengkapnya, lihat Menggunakan templat pod.
- Aplikasi yang didukung - Spark 2.4.7-amzn-1, Jupyter Enterprise Gateway (titik akhir, pratinjau publik; Kernel Scala tidak didukung).
- Komponen yang didukung - `aws-hm-client` (Konektor Glue), `aws-sagemaker-spark-sdk`, `emr-s3-select`, `emrfs`, `emr-ddb`, `hudi-spark`.
- Klasifikasi konfigurasi yang didukung:

Klasifikasi	Deskripsi
`core-site`	Ubah nilai dalam file core-site.xml Hadoop.
`emrfs-site`	Ubah EMRFS pengaturan.
`spark-metrics`	Ubah nilai dalam file metrics.properties Spark.
`spark-defaults`	Ubah nilai dalam file spark-defaults.conf Spark.
`spark-env`	Ubah nilai di lingkungan Spark.
`spark-hive-site`	Ubah nilai dalam file hive-site.xml Spark.
`spark-log4j`	Ubah nilai dalam file log4j.properties Spark.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering sesuai dengan XML file konfigurasi untuk aplikasi, spark-hive-site seperti.xml.xml. Untuk informasi lebih lanjut, lihat Mengonfigurasi Aplikasi.

emr-5.33.0-latest

Catatan rilis: `emr-5.33.0-latest` saat ini menunjuk ke `emr-5.33.0-20240321`.

Wilayah: `emr-5.33.0-latest` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-5.33.0:latest`

emr-5.33.0-20240321

Catatan rilis: `emr-5.33.0-20240321` dirilis pada 11 Maret 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-5.33.0-20240321` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-5.33.0:20240321`

emr-5.33.0-20221219

Catatan rilis: `emr-5.33.0-20221219` dirilis pada 19 Jan 2023. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: `emr-5.33.0-20221219` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-5.33.0:20221219`

emr-5.33.0-20220802

Catatan rilis: `emr-5.33.0-20220802` dirilis pada 24 Agustus 2022. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui.

Wilayah: `emr-5.33.0-20220802` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-5.33.0:20220802`

emr-5.33.0-20211008

Catatan rilis: `emr-5.33.0-20211008` dirilis pada 9 Desember 2021. Dibandingkan dengan versi sebelumnya, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-5.33.0-20211008` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-5.33.0:20211008`

emr-5.33.0-20210802

Catatan rilis: `emr-5.33.0-20210802` dirilis pada 2 Agustus 2021. Dibandingkan dengan versi sebelumnya, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-5.33.0-20210802` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-5.33.0:20210802`

emr-5.33.0-20210615

Catatan rilis: `emr-5.33.0-20210615` dirilis pada 15 Juni 2021. Dibandingkan dengan versi sebelumnya, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-5.33.0-20210615` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-5.33.0:20210615`

emr-5.33.0-20210323

Catatan rilis: `emr-5.33.0-20210323` dirilis pada tanggal 23 Maret 2021. Ini adalah rilis awal Amazon EMR 5.33.0.

Wilayah: `emr-5.33.0-20210323` tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-5.33.0-20210323`

Amazon EMR pada rilis EKS 5.32.0

Rilis Amazon EMR 5.32.0 berikut tersedia untuk Amazon EMR di. EKS Pilih emr-5.32.0- XXXX rilis tertentu untuk melihat detail lebih lanjut seperti tag gambar kontainer terkait.

- emr-5.32.0-latest
- emr-5.32.0-20240321
- emr-5.32.0-20220802
- emr-5.32.0-20211008
- emr-5.32.0-20210802
- emr-5.32.0-20210615
- emr-5.32.0-20210129
- emr-5.32.0-20201218
- emr-5.32.0-20201201

Catatan rilis untuk Amazon EMR 5.32.0

- Aplikasi yang didukung - Spark 2.4.7-amzn-0, Jupyter Enterprise Gateway (titik akhir, pratinjau publik; Kernel Scala tidak didukung).
- Komponen yang didukung - `aws-hm-client` (Konektor Glue), `aws-sagemaker-spark-sdk`, `emr-s3-select`, `emrfs`, `emr-ddb`, `hudi-spark`.
- Klasifikasi konfigurasi yang didukung:

Klasifikasi	Deskripsi
`core-site`	Ubah nilai dalam file core-site.xml Hadoop.
`emrfs-site`	Ubah EMRFS pengaturan.
`spark-metrics`	Ubah nilai dalam file metrics.properties Spark.
`spark-defaults`	Ubah nilai dalam file spark-defaults.conf Spark.

Klasifikasi	Deskripsi
spark-env	Ubah nilai di lingkungan Spark.
spark-hive-site	Ubah nilai dalam file hive-site.xml Spark.
spark-log4j	Ubah nilai dalam file log4j.properties Spark.

Klasifikasi konfigurasi memungkinkan Anda menyesuaikan aplikasi. Ini sering sesuai dengan XML file konfigurasi untuk aplikasi, spark-hive-site seperti.xml.xml. Untuk informasi lebih lanjut, lihat Mengonfigurasi Aplikasi.

emr-5.32.0-latest

Catatan rilis: emr-5.32.0-latest saat ini menunjuk ke emr-5.32.0-20240321.

Wilayah: emr-5.32.0-latest tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: emr-5.32.0:latest

emr-5.32.0-20240321

Catatan rilis: emr-5.32.0-20240321 dirilis pada 11 Maret 2024. Dibandingkan dengan rilis sebelumnya, rilis ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui dan perbaikan kritis.

Wilayah: emr-5.32.0-20240321 tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: emr-5.32.0:20240321

emr-5.32.0-20220802

Catatan rilis: emr-5.32.0-20220802 dirilis pada 24 Agustus 2022. Dibandingkan dengan versi sebelumnya, versi ini telah diperbarui dengan paket Amazon Linux yang baru diperbarui.

Wilayah: emr-5.32.0-20220802 tersedia di semua Wilayah yang didukung oleh Amazon EMR diEKS. Untuk informasi selengkapnya, lihat Amazon EMR di titik akhir EKS layanan.

Tanda gambar kontainer: `emr-5.32.0:20220802`

emr-5.32.0-20211008

Catatan rilis: `emr-5.32.0-20211008` dirilis pada 9 Desember 2021. Dibandingkan dengan versi sebelumnya, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-5.32.0-20211008` tersedia di Wilayah Berikut: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Tanda gambar kontainer: `emr-5.32.0:20211008`

emr-5.32.0-20210802

Catatan rilis: `emr-5.32.0-20210802` dirilis pada 2 Agustus 2021. Dibandingkan dengan versi sebelumnya, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-5.32.0-20210802` tersedia di Wilayah Berikut: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Tanda gambar kontainer: `emr-5.32.0:20210802`

emr-5.32.0-20210615

Catatan rilis: `emr-5.32.0-20210615` dirilis pada 15 Juni 2021. Dibandingkan dengan versi sebelumnya, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-5.32.0-20210615` tersedia di Wilayah Berikut: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Tanda gambar kontainer: `emr-5.32.0:20210615`

emr-5.32.0-20210129

Catatan rilis: `emr-5.32.0-20210129` dirilis pada tanggal 29 Januari 2021. Dibandingkan dengan `emr-5.32.0-20201218`, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-5.32.0-20210129` tersedia di Wilayah berikut: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Tanda gambar kontainer: `emr-5.32.0-20210129`

emr-5.32.0-20201218

Catatan rilis: `5.32.0-20201218` dirilis pada tanggal 18 Desember 2020. Dibandingkan dengan `5.32.0-20201201`, versi ini berisi perbaikan masalah dan pembaruan keamanan.

Wilayah: `emr-5.32.0-20201218` tersedia di Wilayah berikut: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Tanda gambar kontainer: `emr-5.32.0-20201218`

emr-5.32.0-20201201

Catatan rilis: `5.32.0-20201201` dirilis pada tanggal 1 Desember 2020. Ini adalah rilis awal Amazon EMR 5.32.0.

Wilayah: `5.32.0-20201201d` tersedia di Wilayah berikut: US East (N. Virginia), US West (Oregon), Asia Pacific (Tokyo), Europe (Ireland), South America (Sao Paulo).

Tanda gambar kontainer: `emr-5.32.0-20201201`

Riwayat dokumen

Tabel berikut menjelaskan perubahan penting pada dokumentasi sejak rilis terakhir Amazon EMR diEKS. Untuk informasi lebih lanjut tentang pembaruan dokumentasi ini, Anda dapat berlangganan RSS umpan.

Perubahan	Deskripsi	Tanggal
Rilis baru	Amazon EMR pada rilis EKS 7.2.0	Juli 25, 2024
Rilis baru	Amazon EMR pada rilis EKS 7.1.0	April 17, 2024
Rilis baru	Amazon EMR pada rilis EKS 7.0.0	22 Desember 2023
Rilis baru	Amazon EMR pada rilis EKS 6.15.0	17 November 2023
Rilis baru	Amazon EMR pada rilis EKS 6.14.0	17 Oktober 2023
Perbarui konten	Ubah nama “titik akhir terkelola” menjadi titik akhir interaktif; Ketersediaan umum titik akhir interaktif	September 29, 2023
Rilis baru	Amazon EMR pada rilis EKS 6.13.0, dan dokumen pratinjau publik untuk Menjalankan pekerjaan Flink dengan Amazon EMR di EKS	12 September 2023
Rilis baru	Amazon EMR pada rilis EKS 6.12.0	21 Juli 2023
Konten baru	Ditambahkan Menggunakan Volcano sebagai penjadwal khusus untuk Apache Spark di Amazon EMR di EKS	13 Juni 2023
Konten baru	Ditambahkan Menggunakan Volcano sebagai penjadwal khusus untuk Apache Spark di Amazon EMR di EKS	13 Juni 2023
Konten baru	Ditambahkan Menggunakan rotasi log kontainer Spark	12 Juni 2023

Perubahan	Deskripsi	Tanggal
Perbarui konten	Memperbarui dokumentasi gambar kustom untuk menemukan informasi gambar dasar di Galeri ECR Publik Amazon.	8 Juni 2023
Rilis baru	Amazon EMR pada rilis EKS 6.11.0	8 Juni 2023
Konten baru	Menambahkan Menjalankan pekerjaan Spark dengan operator Spark dan mengatur ulang bagian Job Runs di bawahMenjalankan pekerjaan dengan Amazon EMR di EKS.	Juni 5, 2023
Konten baru	Ditambahkan dua bagian: Menggunakan penskalaan otomatis vertikal dengan pekerjaan Amazon EMR Spark dan Menggunakan notebook Jupyter yang dihosting sendiri	4 Mei 2023
Halaman riwayat dokumen	Membuat halaman riwayat dokumen untuk Amazon EMR diEKS.	13 Maret 2023
Halaman kebijakan terkelola	Membuat halaman kebijakan terkelola untuk Amazon EMR diEKS.	13 Maret 2023

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.