aws

Amazon EMR Serverless 用户指南

Amazon EMR

Amazon EMR: Amazon EMR Serverless 用户指南

Table of Contents

什么是 Amazon EMR Serverless？

Amazon EMR Serverless 是 Amazon EMR 一个的部署选项，提供了无服务器运行时环境。这简化了使用 Apache Spark 和 Apache Hive 等最新开源框架的分析应用程序的操作。有了 EMR Serverless，您无需配置、优化、保护或操作集群，即可使用这些框架运行应用程序。

EMR Serverless 有助于避免为数据处理作业配置过多或过少的资源。EMR Serverless 可自动确定应用程序所需的资源，获取这些资源来处理作业，并在作业完成后释放资源。对于应用程序需要在几秒钟内得到响应的用例，比如交互式数据分析，可在创建应用程序时预先初始化应用程序所需的资源。

借助 EMR Serverless，您将继续获得 Amazon EMR 的优势，例如开源兼容性、并发性以及针对流行框架优化的运行时性能。

EMR Serverless 适合那些希望使用开源框架轻松操作应用程序的客户。还提供了快速启动作业、自动容量管理和简单的成本控制。

概念

在本节中，我们将介绍《EMR Serverless 用户指南》中出现的 EMR Serverless 术语和概念。

发行版

Amazon EMR 发行版是一组来自大数据生态系统的开源应用程序。每个发行版都包含不同的大数据应用程序、组件和功能，您可以从中选择来部署和配置 EMR Serverless，以便运行您的应用程序。创建应用程序时，请指定其发行版。选择要在应用程序中使用的 Amazon EMR 发行版和开源框架版本。要了解有关预发行版的更多信息，请参阅 Amazon EMR Serverless 发行版。

应用程序

通过 EMR Serverless，您可以创建一个或多个使用开源分析框架的 EMR Serverless 应用程序。要创建应用程序，请指定以下属性：

- 您要使用的开源框架版本的 Amazon EMR 发行版。要确定发行版，请参阅 Amazon EMR Serverless 发行版。
- 您希望应用程序使用的特定运行时，例如 Apache Spark 或 Apache Hive。

创建应用程序后，请向其提交数据处理作业或交互请求。

每个 EMR Serverless 应用程序都在安全的 Amazon Virtual Private Cloud（VPC）上运行，与其他应用程序严格分开。此外，使用 AWS Identity and Access Management (IAM) 策略来定义哪些用户和角色可以访问该应用程序。还可以指定限制来控制和跟踪应用程序产生的使用成本。

当必须执行以下操作时，请考虑创建多个应用程序：

- 使用不同的开源框架
- 针对不同的用例使用不同版本的开源框架
- 从一个版本升级到另一个版本时执行 A/B 测试
- 为测试和生产场景维护独立的逻辑环境
- 为不同团队提供独立的逻辑环境，并进行独立的成本控制和使用跟踪
- 将不同的 line-of-business应用程序分开

EMR Serverless 是一项区域服务，可简化工作负载在一个区域内跨多个可用区的运行方式。要了解有关如何将应用程序与 EMR Serverless 结合使用的更多信息，请参阅 与 EMR Serverless 应用程序交互并进行配置。

任务运行

作业运行是提交给 EMR Serverless 应用程序的请求，应用程序会以异步方式执行并跟踪其完成过程。作业的示例包括您提交给 Apache Hive 应用程序的 HiveQL 查询，或者您提交给 Apache PySpark Spark 应用程序的数据处理脚本。提交任务时，必须指定在 IAM 中创作的运行时角色，该角色 AWS 用于访问资源，例如 Amazon S3 对象。您可以向应用程序提交多个作业运行请求，并且每个作业运行都可以使用不同的运行时角色来访问 AWS 资源。EMR Serverless 应用程序在收到作业后立即开始执行作业，并同时运行多个作业请求。要了解有关 EMR Serverless 如何运行作业的更多信息，请参阅运行任务。

工作线程

EMR Serverless 应用程序在内部使用工作线程来执行工作负载。这些工作线程的默认大小取决于您的应用程序类型和 Amazon EMR 发行版。计划作业运行时，请覆盖这些大小。

当您提交作业时，EMR Serverless 会计算应用程序在作业中所需的资源，并调度工作线程。EMR Serverless 可将工作负载分解为多个任务，下载映像，预置和设置工作线程，并在作业完成后停用。EMR Serverless 会根据作业每个阶段所需的工作负载和并行度自动扩展或缩减工作线程。这种自动扩展使您无需预估应用程序运行工作负载所需的工作线程数。

预初始化容量

EMR Serverless 提供了预初始化容量功能，可使工作线程在几秒钟内完成初始化并做好响应准备。此容量可以为应用程序创建一个热工作线程池。要为每个应用程序配置此功能，请设置应用程序的 `initial-capacity` 参数。配置预初始化容量后，作业可立即启动，以便实现迭代应用程序和时间敏感型作业。要了解有关预初始化工作线程的更多信息，请参阅 使用 EMR Serverless 时配置应用程序。

EMR Studio

EMR Studio 是一个用户控制台，可用于管理 EMR Serverless 应用程序。如果在创建第一个 EMR Serverless 时，您的账户中没有 EMR Studio，我们会自动为您创建一个。从 Amazon EMR 控制台访问 EMR Studio，或者通过 IAM 或 IAM Identity Center 从身份提供者 (IdP) 开启联合访问。这样，用户就可以访问 Studio 并管理 EMR Serverless 应用程序，而无需直接访问 Amazon EMR 控制台。要了解有关 EMR Serverless 应用程序如何与 EMR Studio 配合使用的更多信息，请参阅 在 EMR Studio 控制台中创建 EMR Serverless 应用程序 和 从 EMR Studio 控制台运行作业。

开始使用 EMR Serverless 的先决条件

本节介绍了运行 EMR Serverless 的管理先决条件。其中包括账户配置和权限管理。

主题

注册获取 AWS 账户

如果您没有 AWS 账户，请完成以下步骤来创建一个。

要注册 AWS 账户

1. 打开https://portal.aws.amazon.com/billing/注册。
2. 按照屏幕上的说明操作。

   在注册时，将接到电话或收到短信，要求使用电话键盘输入一个验证码。

   当您注册时 AWS 账户，就会创建AWS 账户根用户一个。根用户有权访问该账户中的所有 AWS 服务 和资源。作为最佳安全实践，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

AWS 注册过程完成后会向您发送一封确认电子邮件。您可以随时前往 https://aws.amazon.com/并选择“我的账户”，查看您当前的账户活动并管理您的账户。

创建具有管理访问权限的用户

注册后，请保护您的安全 AWS 账户 AWS 账户根用户 AWS IAM Identity Center，启用并创建管理用户，这样您就不会使用 root 用户执行日常任务。

保护你的 AWS 账户根用户

1. 选择 Root 用户并输入您的 AWS 账户 电子邮件地址，以账户所有者的身份登录。AWS 管理控制台在下一页上，输入您的密码。

   要获取使用根用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的 Signing in as the root user。
2. 为您的根用户启用多重身份验证（MFA）。

   有关说明，请参阅 I A M 用户指南中的为 AWS 账户 根用户启用虚拟 MFA 设备（控制台）。

创建具有管理访问权限的用户

1. 启用 IAM Identity Center。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的启用 AWS IAM Identity Center。
2. 在 IAM Identity Center 中，为用户授予管理访问权限。

   有关使用 IAM Identity Center 目录 作为身份源的教程，请参阅《用户指南》 IAM Identity Center 目录中的使用默认设置配置AWS IAM Identity Center 用户访问权限。

以具有管理访问权限的用户身份登录

- 要使用您的 IAM Identity Center 用户身份登录，请使用您在创建 IAM Identity Center 用户时发送到您的电子邮件地址的登录 URL。

  有关使用 IAM Identity Center 用户登录的帮助，请参阅AWS 登录 用户指南中的登录 AWS 访问门户。

将访问权限分配给其他用户

1. 在 IAM Identity Center 中，创建一个权限集，该权限集遵循应用最低权限的最佳做法。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Create a permission set。
2. 将用户分配到一个组，然后为该组分配单点登录访问权限。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的 Add groups。

授予权限

在生产环境中，我们建议您使用更精细的策略。有关此类策略的示例，请参阅 EMR Serverless 的用户访问策略示例。要了解有关访问管理的更多信息，请参阅 IAM 用户指南中的AWS 资源访问管理。

对于需要在沙盒环境中开始使用 EMR Serverless 的用户，请使用类似下面的策略：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "EMRStudioCreate",
      "Effect": "Allow",
      "Action": [
        "elasticmapreduce:CreateStudioPresignedUrl",
        "elasticmapreduce:DescribeStudio",
        "elasticmapreduce:CreateStudio",
        "elasticmapreduce:ListStudios"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "EMRServerlessFullAccess",
      "Effect": "Allow",
      "Action": [
        "emr-serverless:*"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "AllowEC2ENICreationWithEMRTags",
      "Effect": "Allow",
      "Action": [
        "ec2:CreateNetworkInterface"
      ],
      "Resource": [
```

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

要提供访问权限，请为您的用户、组或角色添加权限：

- 中的用户和群组 AWS IAM Identity Center：

  创建权限集合。按照《AWS IAM Identity Center 用户指南》中创建权限集的说明进行操作。

- 通过身份提供者在 IAM 中托管的用户：

  创建适用于身份联合验证的角色。按照《IAM 用户指南》中针对第三方身份提供者创建角色（联合身份验证）的说明进行操作。

- IAM 用户：
  - 创建您的用户可以担任的角色。按照《IAM 用户指南》中为 IAM 用户创建角色的说明进行操作。
  - （不推荐使用）将策略直接附加到用户或将用户添加到用户组。按照《IAM 用户指南》中向用户添加权限（控制台）中的说明进行操作。

授予编程式访问权限

如果用户想在 AWS 外部进行交互，则需要编程访问权限 AWS 管理控制台。授予编程访问权限的方式取决于正在访问的用户类型 AWS。

要向用户授予编程式访问权限，请选择以下选项之一。

哪个用户需要编程式访问权限？	目的	方式
IAM	（推荐）使用控制台凭证作为临时凭证，签署对 AWS CLI AWS SDKs、或的编程请求 AWS APIs。	按照您希望使用的界面的说明进行操作。 • 有关的 AWS CLI，请参阅《AWS Command Line Interface 用户指南》中的“登录 AWS 本地开发”。 • 有关信息 AWS SDKs，请参阅《工具参考指南》AWS SDKs 和《工具参考指南》中的登录进行 AWS 本地开发。
人力身份 （在 IAM Identity Center 中管理的用户）	使用临时证书签署向 AWS CLI AWS SDKs、或发出的编程请求 AWS APIs。	按照您希望使用的界面的说明进行操作。 • 有关的 AWS CLI，请参阅《AWS Command Line Interface 用户指南》 AWS IAM Identity Center中的 “配置 AWS CLI 要使用”。 • 有关工具和 AWS SDKs AWS APIs，请参阅《工具参考指南》中的 IAM 身份中心身份验证AWSSDKs 和工具参考指南。
IAM	使用临时证书签署向 AWS CLI AWS SDKs、或发出的编程请求 AWS APIs。	按照 IAM 用户指南中的将临时证书与 AWS 资源配合使用中的说明进行操作。
IAM	（不推荐使用）	按照您希望使用的界面的说明进行操作。

哪个用户需要编程式访问权限？	目的	方式
	使用长期凭证签署向 AWS CLI AWS SDKs、或发出的编程请求 AWS APIs。	• 有关信息 AWS CLI，请参阅用户指南中的使用 IAM 用户证书进行身份验证。AWS Command Line Interface • 有关 AWS SDKs 和工具，请参阅《工具参考指南》AWS SDKs 和《工具参考指南》中的使用长期凭证进行身份验证。 • 有关信息 AWS APIs，请参阅 IAM 用户指南中的管理 IAM 用户的访问密钥。

安装和配置 AWS CLI

如果要使用 EMR Serverless APIs，请安装最新版本的 ()。 AWS Command Line Interface AWS CLI你不需要从 EMR Studio 控制台中使用 EMR Serverless，也可以按照中的步骤在没有 CLI 的情况下开始使用。 AWS CLI 从控制台开始使用 EMR Serverless

要设置 AWS CLI

1. 要安装 AWS CLI 适用于 macOS、Linux 或 Windows 的最新版本，请参阅安装或更新最新版本的。 AWS CLI
2. 要配置您的访问权限 AWS CLI 和安全设置（包括 EMR Serverless），请参阅使用进行快速配置。 AWS 服务`aws configure`
3. 要验证设置，请在 DataBrew 命令提示符下输入以下命令。

   ```
   aws emr-serverless help
   ```

   AWS CLI 命令使用配置 AWS 区域 中的默认值，除非您使用参数或配置文件进行设置。要使用参数 AWS 区域 进行设置，请将该`--region`参数添加到每个命令中。

要使用配置文件 AWS 区域 进行设置，请先在~/.aws/config文件或文件中添加命名的配置%UserProfile%/.aws/config文件（适用于 Microsoft Windows）。按 AWS CLI的命名配置文件中的步骤执行操作。接下来，使用与以下示例类似的命令来设置您的 AWS 区域 和其他设置。

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

打开 控制台

本节中大部分面向控制台的主题都是从 Amazon EMR 控制台开始。如果您尚未登录自己的 AWS 账户，请登录，然后打开 Amazon EMR 控制台并继续进入下一部分，继续开始使用 Amazon EMR。

开始使用 Amazon EMR Serverless

本教程可帮助您在部署示例 Spark 或 Hive 工作负载时开始使用 EMR Serverless。您将创建、运行和调试自己的应用程序。我们将在本教程的大部分内容中展示默认选项。

在启动 EMR Serverless 应用程序之前，请完成以下任务。

主题

授予使用 EMR Serverless 的权限

要使用 EMR Serverless，您需要具有附加策略的用户或 IAM 角色，可授予使用 EMR Serverless 的权限。要创建用户并向该用户附加适当的策略，请按照 授予权限 中的说明操作。

准备 EMR Serverless 存储

在本教程中，您将使用 S3 存储桶来存储示例 Spark 或 Hive 工作负载（使用 EMR Serverless 应用程序运行）的输出文件和日志。要创建存储桶，请按照《Amazon Simple Storage Service 控制台用户指南》中创建存储桶的说明操作。将对 *amzn-s3-demo-bucket* 的进一步引用替换为新建存储桶的名称。

创建 EMR Studio 运行交互式工作负载

如果要使用 EMR Serverless 通过托管在 EMR Studio 中的 Notebook 执行交互式查询，则需要指定一个 S3 存储桶，并为 EMR Serverless 指定一个最低服务角色，以创建 Workspace。有关设置步骤，请参阅《Amazon EMR 管理指南》中的设置 EMR Studio。有关交互式工作负载的更多信息，请参阅 通过 EMR Studio 使用 EMR Serverless 运行交互式工作负载。

创建作业运行时角色

在 EMR Serverless 中运行的作业使用运行时角色，该角色在运行时为特定资源 AWS 服务 和资源提供精细权限。在本教程中，数据和脚本存放在公有 S3 存储桶中。存储桶 *amzn-s3-demo-bucket* 存储输出。

要设置作业运行时角色，先创建一个具有信任策略的运行时角色，以便 EMR Serverless 可以使用新角色。然后，将所需的 S3 访问策略附加到该角色。以下步骤将指导您完成此过程。

Console

1. 导航到 https://console.aws.amazon.com/iam/ 的 IAM 控制台。
2. 在左侧导航窗格中，选择 Policies（策略）。
3. 选择创建策略。
4. 创建策略页面将在新选项卡中打开。选择策略编辑器为 JSON，并在下面粘贴策略 JSON。

> **Important**
>
> 将以下策略中的 *amzn-s3-demo-bucket* 替换为 准备 EMR Serverless 存储 中创建的实际存储桶名称。这是 S3 访问的基本策略。有关更多作业运行时角色示例，请参阅 Amazon EMR Serverless 的作业运行时角色。

JSON

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

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

5. 选择下一步为您的策略输入名称（如 EMRServerlessS3AndGlueAccessPolicy），然后选择创建策略
6. 在 IAM 控制台的左侧导航窗格中，选择角色。

7. 选择创建角色。
8. 对于角色类型，选择自定义信任策略并粘贴以下信任策略。这样，提交到您的 Amazon EMR 无服务器应用程序的任务就可以 AWS 服务 代表您访问其他应用程序。

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole"
      ],
      "Resource": "arn:aws:iam::123456789012:role/
EMRServerlessExecutionRole",
      "Sid": "AllowSTSAssumerole"
    }
  ]
}
```

9. 选择 “下一步” 导航至 “添加权限” 页面，然后选择 EMRServerlessS3 AndGlueAccessPolicy。
10. 在命名、查看和创建页面上，在角色名称中输入角色的名称，例如 `EMRServerlessS3RuntimeRole`。要创建此 IAM 角色，请选择创建角色。

CLI

1. 创建一个名为 `emr-serverless-trust-policy.json` 的文件，其中包含要用于 IAM 角色的信任策略。该文件应包含以下策略。

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "EMRServerlessTrustPolicy",
      "Action": [
        "sts:AssumeRole"
```

```
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:iam::123456789012:role/
EMRServerlessExecutionRole"
    }
  ]
}
```

2. 创建命名为 EMRServerlessS3RuntimeRole 的 IAM 角色。使用您在上一步中创建的信任策略。

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

记下输出中的 ARN。在作业提交时，您可以使用新角色的 ARN，后面称为 *job-role-arn*。

3. 创建一个名为 emr-sample-access-policy.json 的文件，来定义工作负载的 IAM 策略。这提供了对公有 S3 存储桶中存储的脚本和数据的读取访问权限以及对 *amzn-s3-demo-bucket* 的读写访问权限。

> **Important**
>
> 将以下策略中的 *amzn-s3-demo-bucket* 替换为 准备 EMR Serverless 存储 中创建的实际存储桶名称。这是 Glue 和 AWS S3 访问的基本策略。有关更多作业运行时角色示例，请参阅 Amazon EMR Serverless 的作业运行时角色。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "ReadAccessForEMRSamples",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
```

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

4. 使用您在步骤 3 中创建的策略文件创建一个名为 EMRServerlessS3AndGlueAccessPolicy 的 IAM 策略。记下输出中的 ARN，因为下一步将使用新策略的 ARN。

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

记下输出中新策略的 ARN。下一步将用 *policy-arn* 来替代。

5. 将 IAM 策略 EMRServerlessS3AndGlueAccessPolicy 附加到作业运行时角色 EMRServerlessS3RuntimeRole。

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

从控制台开始使用 EMR Serverless

本节介绍了如何使用 EMR Serverless，包括创建 EMR Studio。还介绍了如何提交作业运行和查看日志。

完成步骤

- 步骤 1：创建 EMR Serverless 应用程序
- 步骤 2：提交作业运行或交互式工作负载
- 步骤 3：查看应用程序 UI 和日志
- 步骤 4：清除

步骤 1：创建 EMR Serverless 应用程序

使用 EMR Serverless 创建新应用程序，如下所示。

1. 登录 AWS 管理控制台 并打开亚马逊 EMR 控制台，网址为 /emr。https://console.aws.amazon.com
2. 在左侧导航窗格中，选择 EMR Serverless 以导航到 EMR Serverless 登录页面。
3. 要创建或管理 EMR Serverless 应用程序，需使用 EMR Studio UI。

- 如果您要创建应用程序 AWS 区域 的地方已经有 EMR Studio，请选择 “管理应用程序” 以导航到您的 EMR Studio，或者选择要使用的工作室。
- 如果您要创建应用程序 AWS 区域 的地方没有 EMR Studio，请选择 “开始”，然后选择 “创建” 并启动 Studio。EMR Serverless 会为您创建一个 EMR Studio，用来创建和管理应用程序。

4. 在新选项卡中打开的创建 Studio UI 中，输入应用程序的名称、类型和发布版本。如果只想运行批处理作业，请选择仅对批处理作业使用默认设置。对于交互式工作负载，请选择对交互式工作负载使用默认设置。您还可以使用此选项在交互式应用程序上运行批处理作业。如果需要，您可以在以后更改这些设置。

   有关更多信息，请参阅创建 Studio。
5. 选择创建应用程序，创建您的第一个应用程序。

继续下一节 步骤 2：提交作业运行或交互式工作负载，提交作业运行或交互式工作负载。

步骤 2：提交作业运行或交互式工作负载

Spark job run

在本教程中，我们使用 PySpark 脚本来计算多个文本文件中唯一单词的出现次数。公有的只读 S3 存储桶可同时存储脚本和数据集。

运行 Spark 作业

1. 使用以下命令将示例脚本 `wordcount.py` 上传到新存储桶。

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

2. 完成 步骤 1：创建 EMR Serverless 应用程序 后，您将进入 EMR Studio 中的应用程序详情页面。在此页面上，选择提交作业选项。
3. 在提交作业页面上，完成以下操作。
   - 在名称字段中，输入要调用的作业运行名称。
   - 在运行时角色字段中，输入您在 创建作业运行时角色 中创建的角色名称。
   - 在脚本位置字段中，输入 `s3://amzn-s3-demo-bucket/scripts/wordcount.py` 作为 S3 URI。

- 在脚本参数字段中，输入 ["s3://*amzn-s3-demo-bucket*/emr-serverless-spark/output"]。
- 在 Spark 属性部分，选择以文本形式编辑，然后输入以下配置。

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

4. 要开始作业运行，请选择提交作业。
5. 在作业运行选项卡中，您应该看到状态为正在运行的新作业运行。

Hive job run

在本教程的这一部分，我们将创建一个表，插入几条记录，然后运行计数聚合查询。要运行 Hive 作业，先创建一个包含所有要作为单个作业的一部分运行的 Hive 查询的文件，将该文件上传到 S3，然后在启动 Hive 作业时指定此 S3 路径。

运行 Hive 作业

1. 创建一个名为 hive-query.ql 的文件，其中包含您要在 Hive 作业中运行的所有查询。

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

2. 使用以下命令将 hive-query.ql 上传到 S3 存储桶。

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

3. 完成 步骤 1：创建 EMR Serverless 应用程序 后，您将进入 EMR Studio 中的应用程序详情页面。在此页面上，选择提交作业选项。
4. 在提交作业页面上，完成以下操作。
   - 在名称字段中，输入要调用的作业运行名称。
   - 在运行时角色字段中，输入您在 创建作业运行时角色 中创建的角色名称。

- 在脚本位置字段中，输入 `s3://amzn-s3-demo-bucket/emr-serverless-hive/query/hive-query.ql` 作为 S3 URI。
- 在 Hive 属性部分，选择以文本形式编辑，然后输入以下配置。

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

- 在作业属性部分，选择以文本形式编辑，然后输入以下 JSON。

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

5. 要开始运行作业，请选择提交作业。
6. 在作业运行选项卡中，您应该看到状态为正在运行的新作业运行。

Interactive workload

在 Amazon EMR 6.14.0 及更高版本中，您可以使用 EMR Studio 中托管的 Notebook 在 EMR Serverless 中运行 Spark 的交互式工作负载。有关权限和先决条件等更多信息，请参阅 通过 EMR Studio 使用 EMR Serverless 运行交互式工作负载。

创建应用程序并设置所需权限后，请按以下步骤使用 EMR Studio 运行交互式 Notebook：

1. 导航到 EMR Studio 中的 Workspace。如果仍需要配置 Amazon S3 存储位置和 EMR Studio 服务角色，请选择屏幕顶部横幅中的配置 Studio 按钮。
2. 要访问 Notebook，请选择一个 Workspace 或创建一个新 Workspace。使用快速启动在新选项卡中打开 Workspace。

3. 转到新打开的选项卡。从左侧导航栏中选择计算图标。选择 EMR Serverless 作为计算类型。
4. 选择您在上一节中创建的交互式应用程序。
5. 在运行时角色字段中，输入 EMR Serverless 应用程序在运行作业时可代入的 IAM 角色名称。要了解有关运行时角色的更多信息，请参阅《Amazon EMR Serverless 用户指南》中的作业运行时角色。
6. 选择附加。该过程可能需要一分钟。附加后，页面将会刷新。
7. 选择内核并启动 Notebook。您还可以浏览 EMR Serverless 上的示例 Notebook，将其复制到 Workspace。要访问示例 Notebook，请导航到左侧导航栏中的 **{...}** 菜单，然后浏览 Notebook文件名中包含 `serverless` 的 Notebook。
8. 在 Notebook 中，您可以访问驱动程序日志链接和指向 Apache Spark UI 链接，Apache Spark UI 是一个实时界面，提供了监控作业的指标。有关更多信息，请参阅《Amazon EMR Serverless 用户指南》中的监控 EMR Serverless 应用程序和作业。

将应用程序附加到 Studio Workspace 时，如果应用程序尚未运行，则会自动触发启动。您也可以预先启动应用程序，在将其附加到 Workspace 之前准备就绪。

步骤 3：查看应用程序 UI 和日志

要查看应用程序 UI，先确定作业运行情况。该作业运行的第一行选项中提供了 Spark UI 或 Hive Tez UI 选项（取决于作业类型）。选择相应的选项。

如果选择 Spark UI，请选择执行程序选项卡以查看驱动程序和执行程序日志。如果选择 Hive Tez UI，请选择所有任务选项卡以查看日志。

一旦作业运行状态显示为成功，就可以在 S3 存储桶中查看作业的输出。

步骤 4：清除

虽然您创建的应用程序应在 15 分钟不活动后自动停止，但我们仍然建议您释放不打算再次使用的资源。

要删除应用程序，请导航到列出应用程序页面。选择您创建的应用程序，然后选择操作→停止以停止应用程序。应用程序处于 `STOPPED` 状态后，选择同一应用程序，然后选择操作→删除。

有关运行 Spark 和 Hive 作业的更多示例，请参阅 运行 EMR Serverless 作业时使用 Spark 配置 和 运行 EMR Serverless 作业时使用 Hive 配置。

从这里开始 AWS CLI

使用创建应用程序、运行作业、检查作业运行输出和删除资源的命令开始 AWS CLI 使用 EMR Serverless。

步骤 1：创建 EMR Serverless 应用程序

使用 `emr-serverless create-application` 命令创建您的第一个 EMR Serverless 应用程序。您需要指定应用程序类型，以及与要使用的应用程序版本关联的 Amazon EMR 发行版标签。应用程序的名称为可选。

Spark

要创建 Spark 应用程序，请运行以下命令。

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

Hive

要创建 Hive 应用程序，请运行以下命令。

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

记下输出中返回的应用程序 ID。您将在启动应用程序和提交作业时使用该 ID，后面称为 *application-id*。

在继续学习 CREATED 之前，请确保您的应用程序已使用 `get-application` API 达到 步骤 2：将作业运行提交到 EMR Serverless 应用程序 状态。

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

EMR Serverless 会创建工作线程来容纳您请求的作业。默认情况下，这些是按需创建的，但您也可以在创建应用程序时通过设置 `initialCapacity` 参数来指定预初始化容量。您还可以使用

maximumCapacity 参数来限制应用程序可以使用的最大总容量。要了解有关这些选项的更多信息，请参阅 使用 EMR Serverless 时配置应用程序。

步骤 2：将作业运行提交到 EMR Serverless 应用程序

现在，您的 EMR Serverless 应用程序已准备好运行作业。

Spark

在此步骤中，我们使用 PySpark 脚本来计算多个文本文件中唯一单词的出现次数。公有的只读 S3 存储桶可同时存储脚本和数据集。应用程序将 Spark 运行时中的输出文件和日志数据发送到您创建的 S3 存储桶中的 /output 和 /logs 目录。

运行 Spark 作业

1. 使用以下命令将我们要运行的示例脚本复制到您的新存储桶。

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

2. 在以下命令中，将 *application-id* 替换为您的应用程序 ID。将 *job-role-arn* 替换为您在 创建作业运行时角色 中创建的运行时角色 ARN。将 *job-run-name* 替换为您要调用的作业运行名称。将所有 *amzn-s3-demo-bucket* 字符串替换为您创建的 Amazon S3 存储桶，然后将 /output 添加到路径中。这将在您的存储桶中创建一个新文件夹，EMR Serverless 可以在其中复制应用程序的输出文件。

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

3. 记下输出中返回的作业运行 ID。在以下步骤中将 *job-run-id* 替换为此 ID。

Hive

在本教程中，我们将创建一个表，插入几条记录，然后运行计数聚合查询。要运行 Hive 作业，先创建一个包含所有要作为单个作业的一部分运行的 Hive 查询的文件，将该文件上传到 S3，然后在启动 Hive 作业时指定此 S3 路径。

运行 Hive 作业

1. 创建一个名为 `hive-query.ql` 的文件，其中包含您要在 Hive 作业中运行的所有查询。

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

2. 使用以下命令将 `hive-query.ql` 上传到 S3 存储桶。

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

3. 在以下命令中，将 *application-id* 替换为您自己的应用程序 ID。将 *job-role-arn* 替换为您在 创建作业运行时角色 中创建的运行时角色 ARN。将所有 *amzn-s3-demo-bucket* 字符串替换为您创建的 Amazon S3 存储桶，然后将 `/output` 和 `/logs` 添加到路径中。这将在您的存储桶中创建的新文件夹，EMR Serverless 可以在其中复制应用程序的输出和日志文件。

```
aws emr-serverless start-job-run \
    --application-id application-id \
    --execution-role-arn job-role-arn \
    --job-driver '{
        "hive": {
          "query": "s3://amzn-s3-demo-bucket/emr-serverless-hive/query/hive-
query.ql",
          "parameters": "--hiveconf hive.log.explain.output=false"
        }
    }' \
    --configuration-overrides '{
      "applicationConfiguration": [{
        "classification": "hive-site",
          "properties": {
```

```
            "hive.exec.scratchdir": "s3://amzn-s3-demo-bucket/emr-serverless-
hive/hive/scratch",
            "hive.metastore.warehouse.dir": "s3://amzn-s3-demo-bucket/emr-
serverless-hive/hive/warehouse",
            "hive.driver.cores": "2",
            "hive.driver.memory": "4g",
            "hive.tez.container.size": "4096",
            "hive.tez.cpu.vcores": "1"
            }
        }],
        "monitoringConfiguration": {
          "s3MonitoringConfiguration": {
            "logUri": "s3://amzn-s3-demo-bucket/emr-serverless-hive/logs"
           }
        }
    }'
```

4. 记下输出中返回的作业运行 ID。在以下步骤中将 *job-run-id* 替换为此 ID。

步骤 3：查看作业运行的输出

作业运行通常需要 3-5 分钟才能完成。

Spark

您可以使用以下命令检查 Spark 作业的状态。

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

将日志目标设置为 s3://*amzn-s3-demo-bucket*/emr-serverless-spark/logs，可在 s3://*amzn-s3-demo-bucket*/emr-serverless-spark/logs/applications/*application-id*/jobs/*job-run-id* 下方找到特定作业的日志。

对于 Spark 应用程序，EMR Serverless 每 30 秒将事件日志推送到 S3 日志目标中的 sparklogs 文件夹。作业完成后，驱动程序和执行程序的 Spark 运行时日志会上传到按工作线程类型适当命名的文件夹，例如 driver 或 executor。 PySpark 作业的输出将上传到。s3://*amzn-s3-demo-bucket*/output/

Hive

您可以使用以下命令检查 Hive 作业的状态。

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

将日志目标设置为 s3://*amzn-s3-demo-bucket*/emr-serverless-hive/logs，可在 s3://*amzn-s3-demo-bucket*/emr-serverless-hive/logs/applications/*application-id*/jobs/*job-run-id* 下方找到特定作业的日志。

对于 Hive 应用程序，EMR Serverless 会持续将 Hive 驱动程序上传到 S3 日志目标的 HIVE_DRIVER 文件夹，并将 Tez 任务日志上传到 TEZ_TASK 文件夹。作业运行达到 SUCCEEDED 状态后，Hive 查询的输出将出现在您在 configurationOverrides 的 monitoringConfiguration 字段中指定的 Amazon S3 位置。

步骤 4：清除

本教程学习结束后，请考虑删除创建的资源。建议您释放不打算再次使用的资源。

删除应用程序

要删除应用程序，请使用以下命令。

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

删除 S3 日志存储桶

要删除 S3 日志记录和输出存储桶，请使用以下命令。将 *amzn-s3-demo-bucket* 替换为您在 准备 EMR Serverless 存储 中创建的 S3 存储桶的实际名称。

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

删除作业运行时角色

要删除运行时角色，请将策略与角色分离。然后，就可以删除角色和策略。

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

要删除角色，请使用以下命令。

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

要删除附加到角色的策略，请使用以下命令。

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

有关运行 Spark 和 Hive 作业的更多示例，请参阅 运行 EMR Serverless 作业时使用 Spark 配置 和 运行 EMR Serverless 作业时使用 Hive 配置。

与 EMR Serverless 应用程序交互并进行配置

本节介绍如何通过 AWS CLI与 Amazon EMR Serverless 应用程序交互。它还介绍如何配置应用程序、执行自定义以及 Spark 和 Hive 引擎的默认设置。

主题

应用程序状态

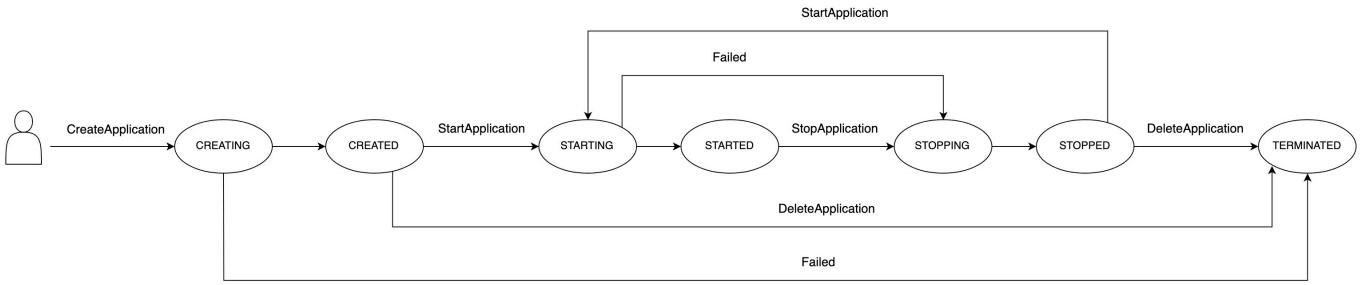
使用 EMR Serverless 创建应用程序时，应用程序运行将进入 `CREATING` 状态。随后，它将经历以下状态，直至成功（退出并返回代码 `0`）或失败（退出并返回非零代码）。

应用程序可能处于以下状态：

州	说明
Creating	应用程序正在准备中，还不能使用。
Created	应用程序已创建，但尚未预置容量。您可以修改应用程序以更改其初始容量配置。
Starting	应用程序正在启动且正在预置容量。
已启动	应用程序已准备好接受新作业。只有在这种状态下，应用程序才接受作业。
Stopping	所有作业都已完成，应用程序正在释放其容量。

州	说明
Stopped	应用程序已停止，并且该用程序上没有资源在运行。您可以修改应用程序以更改其初始容量配置。
已终止	应用程序已终止，未出现在应用程序列表中。

下图展示了 EMR Serverless 应用程序状态的轨迹。



在 EMR Studio 控制台中创建 EMR Serverless 应用程序

在 EMR Studio 控制台中，创建、访问和管理 EMR Serverless 应用程序。要导航到 EMR Studio 控制台，请按照控制台入门中的说明操作。

创建 应用程序

在创建应用程序页面中，按照以下步骤创建 EMR Serverless 应用程序。

1. 在名称字段中，输入要调用的应用程序名称。
2. 在类型字段中，选择 Spark 或 Hive 作为应用程序的类型。
3. 在发行版字段中，选择 EMR 版本号。
4. 在架构选项中，选择要使用的指令集架构。有关更多信息，请参阅Amazon EMR Serverless 架构选项。
   - arm64：64 位 ARM 架构；使用 Graviton 处理器
   - x86_64：64 位 x86 架构；使用基于 x86 的处理器
5. 其余字段有两个应用程序设置选项：默认设置和自定义设置。这些字段均为可选字段。

默认设置：通过默认设置，您可以快速创建具有预初始化容量的应用程序。这包括 Spark 的一个驱动程序和一个执行程序，以及 Hive 的一个驱动程序和一个 Tez 任务。默认设置不允许与您的网络连接 VPCs。应用程序配置为闲置 15 分钟即停止，在提交作业时自动启动。

自定义设置：自定义设置允许您修改以下属性。

- 预初始化容量：驱动程序和执行程序或 Hive Tez 任务工作线程的数量，以及每个工作线程的大小。
- 应用程序限制：应用程序的最大容量。
- 应用程序行为：应用程序的自动启动和自动停止行为。
- 网络连接：与 VPC 资源的网络连接。
- 标签 — 分配给应用程序的自定义标签。

有关预初始化容量、应用程序限制和应用程序行为的更多信息，请参阅 使用 EMR Serverless 时配置应用程序。有关网络连接的更多信息，请参阅 为 EMR Serverless 应用程序配置 VPC 访问权限以连接数据。

6. 要创建应用程序，请选择创建应用程序。

在 EMR Studio 控制台中列出应用程序

您可以在列出应用程序页面上访问所有现有 EMR Serverless 应用程序。您可以选择应用程序的名称，导航到该应用程序的详细信息页面。

在 EMR Studio 控制台中管理应用程序

您可以在列出应用程序页面或特定应用程序的详细信息页面对应用程序执行以下操作。

启动应用程序

选择此选项以手动启动应用程序。

停止应用程序

选择此选项以手动停止应用程序。应用程序必须没有正在运行的作业才能停止。要了解有关应用程序状态转换的更多信息，请参阅应用程序状态。

配置应用程序

在配置应用程序页面上编辑应用程序的可选设置。您可以更改大部分应用程序设置。例如，更改应用程序的发行版标签，将其升级到其他版本的 Amazon EMR，或者将架构从 x86_64 切换到 arm64。其他可选设置与创建应用程序页面上自定义设置部分中的设置相同。有关应用程序设置的更多信息，请参阅 创建 应用程序。

删除应用程序

选择此选项以手动删除应用程序。必须停止应用程序才能将其删除。要了解有关应用程序状态转换的更多信息，请参阅应用程序状态。

在上与您的 EMR 无服务器应用程序交互 AWS CLI

从中创建 AWS CLI、描述和删除各个应用程序。您还可以列出所有应用程序，以便直观地访问。本节介绍如何执行这些操作。有关更多应用程序操作，如启动、停止和更新应用程序，请参阅 EMR Serverless API 参考。有关如何使用使用 EMR Serverless API 的示例 适用于 Java 的 AWS SDK，请参阅我们存储库中的 Java 示例。 GitHub 有关如何使用使用 EMR Serverless API 的示例 AWS SDK for Python (Boto)，请参阅我们存储库中的 Pyth on 示例。 GitHub

要创建应用程序，请使用 `create-application`。必须将 `SPARK` 或 `HIVE` 指定为应用程序 `type`。该命令将返回应用程序的 ARN、名称和 ID。

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

要描述应用程序，请使用 `get-application` 并提供 `application-id`。该命令将返回应用程序的状态和容量相关配置。

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

要列出所有应用程序，请调用 `list-applications`。该命令将返回与 `get-application` 相同的属性，但包括所有应用程序。

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

要删除应用程序，请调用 delete-application 并提供 application-id。

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

使用 EMR Serverless 时配置应用程序

借助 EMR Serverless，配置所使用的应用程序。例如，设置应用程序可以扩展的最大容量，配置预初始化容量以使驱动程序和工作线程随时响应，在应用程序级别指定一组通用的运行时和监控配置。以下页面介绍了使用 EMR Serverless 时如何配置应用程序。

主题

了解 EMR Serverless 中的应用程序行为

本节介绍了 EMR Serverless 的作业提交行为、扩展容量配置以及工作线程配置设置。

默认应用程序行为

自动启动：默认情况下，应用程序配置为在提交作业时自动启动。您可以关闭此功能。

自动停止：默认情况下，应用程序配置为在空闲 15 分钟后自动停止。当应用程序变为 STOPPED 状态时，会释放已配置的预初始化容量。您可以修改应用程序自动停止前的空闲时间，也可以关闭此功能。

最大容量

您可以配置应用程序可扩展的最大容量。指定 CPU、内存（GB）和磁盘（GB）的最大容量。

> **Note**
>
> 最佳做法是将最大容量配置为与支持的工作线程大小成正比，方法是将工作线程数量乘以其大小。例如，如果要将应用程序限制为 50 个工作线程，2 v CPUs、16 GB 内存和 20 GB 磁盘容量，请将最大容量设置为 100 v CPUs、800 GB 内存和 1000 GB 磁盘容量。

支持的工作线程配置

下表显示了可为 EMR Serverless 指定的受支持工作线程配置和大小。根据工作负载的需要，为驱动程序和执行程序配置不同的大小。

工作线程配置和大小

CPU	内存	默认临时存储
1 个 vCPU	最小 2GB，最大 8GB，以 1GB 为增量	20GB - 200GB
2 个 vCPU	最小 4GB，最大 16GB，以 1GB 为增量	20GB - 200GB
4 个 vCPU	最小 8GB，最大 30GB，以 1GB 为增量	20GB - 200GB
8 个 vCPU	最小 16GB，最大 60GB，以 4GB 为增量	20GB - 200GB
16 个 vCPU	最小 32GB，最大 120GB，以 8GB 为增量	20GB - 200GB

CPU — 每个工作器可以有 1、2、4、8 或 16 v CPUs。

内存：每个工作线程的内存（GB）都在上表所列限制范围内。Spark 作业有内存开销，这意味着其使用的内存超过了指定的容器大小。此开销由属性 `spark.driver.memoryOverhead` 和 `spark.executor.memoryOverhead` 指定。开销的默认值为容器内存的 10%，最小为 384MB。在选择工作线程大小时，应考虑此开销。

例如，如果您为工作实例选择 4 vCPUs ，预先初始化的存储容量为 30 GB，则将值设置为大约 27 GB 作为 Spark 作业的执行器内存。这样可以最大限度地提高预初始化容量的利用率。可用内存为 27 GB，再加上 27 GB的 10%（2.7 GB），共计 29.7 GB。

磁盘：您可以为每个工作线程配置临时存储磁盘容量，最小 20GB，最大 200GB。您只需为每个工作线程配置的超出 20GB 的额外存储空间付费。

在 EMR Serverless 中使用应用程序的预初始化容量

EMR Serverless 提供了一项可选功能，可使驱动程序和工作线程保持预初始化状态，并在几秒钟内做出响应。这将为应用程序创建一个热工作线程池。此功能称为预初始化容量。要配置此功能，请将应用程序的 `initialCapacity` 参数设置为要预初始化的工作线程数。通过预初始化的工作线程容量，作业可立即启动。如果要执行迭代应用程序和时间敏感型作业，这是理想的选择。

预初始化容量可保持一个热工作线程池，随时准备在几秒钟内启动作业和会话。即使应用程序处于空闲状态，您也需要为预置的预初始化工作线程付费，因此我们建议为受益于快速启动时间的用例启用，并调整大小以实现资源的最佳利用。EMR Serverless 应用程序在空闲时自动关闭。我们建议在使用预初始化工作线程时保留此功能，以免产生意外费用。

提交作业时，如果来自 `initialCapacity` 的工作线程可用，作业将使用这些资源开始运行。如果这些工作线程已被其他作业使用，或者如果作业需要的资源多于 `initialCapacity` 中的可用资源，则应用程序会请求并获取额外的工作线程，直到达到为应用程序设置的最大资源限制。作业运行结束时，将会释放其使用的工作线程，应用程序可用的资源数将恢复为 `initialCapacity`。即使在 `initialCapacity` 作业完成运行后，应用程序仍会保留资源。当作业不再需要资源运行时，应用程序会释放 `initialCapacity` 以外的多余资源。

预初始化容量在应用程序启动后即可使用。应用程序停止时，预初始化容量将变为非活动状态。只有在请求的预初始化容量已创建并可供使用时，应用程序才会进入 `STARTED` 状态。在应用程序处于 `STARTED` 状态期间，EMR Serverless 会保留预初始化容量，供作业或交互式工作负载使用。该功能可恢复已释放或故障容器的容量。这样就能保持 `InitialCapacity` 参数指定的工作线程数。没有预初始化容量的应用程序状态可立即从 `CREATED` 变为 `STARTED`。

您可以对应用程序进行配置，使其在一段时间内未使用时释放预初始化容量，默认值为 15 分钟。提交新作业时，停止的应用程序会自动启动。您可以在创建应用程序时设置这些自动启动和停止配置，也可以在应用程序处于 `CREATED` 或 `STOPPED` 状态时对其进行更改。

您可以更改 `InitialCapacity` 计数，并为每个工作线程指定计算配置，如 CPU、内存和磁盘。由于无法进行部分修改，请在更改值时指定所有计算配置。只有当应用程序处于 `CREATED` 或 `STOPPED` 状态时，才能更改配置。

> **Note**
>
> 为了优化应用程序对资源的使用，我们建议将容器大小与预初始化容量工作线程大小保持一致。例如，如果您将 Spark 执行器大小配置为 2CPUs ，将内存配置为 8 GB，但预先初始化的容量工作器大小为 4CPUs ，内存为 16 GB，那么 Spark 执行器在分配给此作业时仅使用一半的工作器资源。

为 Spark 和 Hive 自定义预初始化容量

您可以为在特定大数据框架上运行的工作负载进一步自定义预初始化容量。例如，当工作负载在 Apache Spark 上运行时，请指定有多少个工作线程作为驱动程序启动，有多少个工作线程作为执行程序启动。同样，在使用 Apache Hive 时，请指定有多少个工作线程作为 Hive 驱动程序启动，有多少个工作线程运行 Tez 任务。

为运行 Apache Hive 的应用程序配置预初始化容量

以下 API 请求创建了一个基于 Amazon EMR 发行版 emr-6.6.0 运行 Apache Hive 的应用程序。应用程序开始时有 5 个预初始化的 Hive 驱动程序（每个驱动程序有 2 个 vCPU 和 4GB 内存）和 50 个预初始化的 Tez 任务工作线程（每个工作线程有 4 个 vCPU 和 8GB 内存）。当 Hive 查询在此应用程序上运行时，先使用预初始化的工作线程并立即开始执行。如果所有预初始化的工作程序都处于忙碌状态，并且提交了更多的 Hive 作业，则应用程序可以扩展到总计 400 个 vCPU 和 1024GB 内存。您可以选择忽略 `DRIVER` 或 `TEZ_TASK` 工作线程的容量。

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

为运行 Apache Spark 的应用程序配置预初始化容量

以下 API 请求创建了一个基于 Amazon EMR 发行版 6.6.0 运行 Apache Spark 3.2.0 的应用程序。应用程序开始时有 5 个预初始化的 Spark 驱动程序（每个驱动程序有 2 个 vCPU 和 4GB 内存）和 50 个预初始化的执行程序（每个执行程序有 4 个 vCPU 和 8GB 内存）。当 Spark 作业在此应用程序上运行时，先使用预初始化的工作线程并立即开始执行。如果所有预初始化的工作程序都处于忙碌状态，并且提交了更多的 Spark 作业，则应用程序可以扩展到总计 400 个 vCPU 和 1024GB 内存。您可以选择忽略 DRIVER 或 EXECUTOR 的容量。

> **Note**
>
> Spark 在驱动程序和执行程序请求的内存中添加了可配置的内存开销（默认值为 10%）。对于使用预初始化工作线程的作业，初始容量内存配置应大于作业和开销请求的内存。

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

EMR Serverless 的默认应用程序配置

您可以在应用程序级别为同一应用程序下提交的所有作业指定一组通用的运行时和监控配置。这减少了因需要为每个作业提交相同配置而产生的额外开销。

您可以在以下时间点修改配置：

- 在作业提交时声明应用程序级配置。
- 在作业运行期间覆盖默认配置。

以下部分提供了更多详细信息和更多上下文的示例。

在应用程序级别声明配置

您可以为在应用程序下提交的作业指定应用程序级日志记录和运行时配置属性。

`monitoringConfiguration`

要为随应用程序提交的作业指定日志配置，请使用 `monitoringConfiguration` 字段。有关 EMR Serverless 日志记录的更多信息，请参阅 存储日志。

`runtimeConfiguration`

要指定运行时配置属性（如 `spark-defaults`），请在 `runtimeConfiguration` 字段中提供配置对象。这将影响您随应用程序提交的所有作业的默认配置。有关更多信息，请参阅 Hive 配置覆盖参数 和 Spark 配置覆盖参数。

可用的配置分类因特定的 EMR Serverless 发行版而异。例如，自定义 Log4j `spark-driver-log4j2` 和 `spark-executor-log4j2` 的分类仅适用于 6.8.0 及更高版本。有关特定于应用程序的属性列表，请参阅 Spark 作业属性 和 Hive 作业属性。

您还可以在应用程序级别配置 Apache Log4j2 属性、AWS Secrets Manager 数据保护和 Java 17 运行时。

要在应用程序级别传递 Secrets Manager 密钥，请将以下策略附加到需要使用密钥创建或更新 EMR Serverless 应用程序的用户和角色。

JSON

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

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

有关为密钥创建自定义策略的更多信息，请参阅《AWS Secrets Manager 用户指南》中的 AWS Secrets Manager权限策略示例。

> **Note**
>
> 您在应用程序级别指定的 `runtimeConfiguration` 映射到 `StartJobRun` API 中的 `applicationConfiguration`。

示例声明

以下示例展示了如何使用 `create-application` 声明默认配置。

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

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

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

任务运行期间覆盖配置

您可以使用 StartJobRun API 为应用程序配置和监控配置指定配置覆盖。然后，EMR Serverless 合并您在应用程序级别和作业级别指定的配置，以确定作业执行的配置。

合并时的粒度级别如下：

- **ApplicationConfiguration**：分类类型，例如 spark-defaults。
- **MonitoringConfiguration**：配置类型，例如 s3MonitoringConfiguration。

> **Note**
>
> 您在 StartJobRun 设置的配置优先级将取代您在应用程序级别设置的配置。

有关优先级排名的更多信息，请参阅 Hive 配置覆盖参数 和 Spark 配置覆盖参数。

启动作业时，如果未指定特定配置，则会从应用程序继承。如果在作业级别声明配置，可以执行以下操作：

- 覆盖现有配置：在 StartJobRun 请求中提供相同的配置参数和覆盖值。
- 添加其他配置：在 StartJobRun 请求中添加新的配置参数，并指定所需的值。
- 删除现有配置：要删除应用程序运行时配置，请提供要删除的配置的密钥，然并为该配置传递一个空声明 {}。我们不建议删除任何包含作业运行所需参数的分类。例如，如果您尝试删除 Hive 作业所需的属性，作业将会失败。

  要删除应用程序监控配置，请针对相关配置类型使用适当的方法：

  - **cloudWatchLoggingConfiguration**：要删除 cloudWatchLogging，请将启用标志传递为 false。
  - **managedPersistenceMonitoringConfiguration**：要删除托管持久性设置并回退到默认启用状态，请为配置传递一个空声明 {}。
  - **s3MonitoringConfiguration**：要删除 s3MonitoringConfiguration，请为配置传递一个空声明 {}。

示例覆盖

以下示例显示了在 start-job-run 提交作业期间可以执行的不同操作。

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

```
        "sparkSubmit": {
            "entryPoint": "s3://us-east-1.elasticmapreduce/emr-containers/samples/
wordcount/scripts/wordcount.py",
            "entryPointArguments": ["s3://amzn-s3-demo-destination-bucket1/
wordcount_output"]
        }
    }' \
    --configuration-overrides '{
        "applicationConfiguration": [
            {
                // Override existing configuration for spark-defaults in the
 application
                "classification": "spark-defaults",
                "properties": {
                    "spark.driver.cores": "2",
                    "spark.executor.cores": "1",
                    "spark.driver.memory": "4G",
                    "spark.executor.memory": "4G"
                }
            },
            {
                // Add configuration for spark-executor-log4j2
                "classification": "spark-executor-log4j2",
                "properties": {
                    "rootLogger.level": "error",
                    "logger.IdentifierForClass.name": "classpathForSettingLogger",
                    "logger.IdentifierForClass.level": "info"
                }
            },
            {
                // Remove existing configuration for spark-driver-log4j2 from the
 application
                "classification": "spark-driver-log4j2",
                "properties": {}
            }
        ],
        "monitoringConfiguration": {
            "managedPersistenceMonitoringConfiguration": {
                // Override existing configuration for managed persistence
                "enabled": true
            },
            "s3MonitoringConfiguration": {
                // Remove configuration of S3 monitoring
            },
```

```
            "cloudWatchLoggingConfiguration": {
                // Add configuration for CloudWatch logging
                "enabled": true
            }
        }
    }'
```

作业执行时，将根据 Hive 配置覆盖参数 和 Spark 配置覆盖参数 中描述的优先级覆盖排名应用以下分类和配置。

- 分类 `spark-defaults` 将根据作业级别指定的属性进行更新。此分类仅考虑 `StartJobRun` 中包含的属性。
- 分类 `spark-executor-log4j2` 将添加到现有分类列表中。
- 分类 `spark-driver-log4j2` 将被删除。
- `managedPersistenceMonitoringConfiguration` 配置将根据作业级别的配置进行更新。
- `s3MonitoringConfiguration` 的配置将被删除。
- `cloudWatchLoggingConfiguration` 的配置将添加到现有监控配置中。

自定义 EMR Serverless 映像

从 Amazon EMR 6.9.0 开始，请使用自定义映像将应用程序依赖项和运行时环境打包到 Amazon EMR Serverless 的单个容器中。这简化了管理工作负载依赖项的方式，并使软件包更具可移植性。自定义 EMR Serverless 映像具有以下优势：

- 安装并配置针对工作负载优化的软件包。这些软件包无法在 Amazon EMR 运行时环境的公开分发中广泛使用。
- 将 EMR Serverless 与组织内当前确立的构建、测试和部署流程（包括本地开发和测试）集成。
- 应用确立的安全流程（如映像扫描），满足组织内的合规性和监管要求。
- 允许您在应用程序中使用自己的 JDK 和 Python 版本。

EMR Serverless 提供的映像可在您创建自己的映像时作为基础。基础映像提供了映像与 EMR Serverless 交互所需的必要 jar 文件、配置和库。您可以在 Amazon ECR 公开映像浏览馆中找到基础映像。使用与您的应用程序类型（Spark 或 Hive）和发行版匹配的映像。例如，如果您在 Amazon EMR 发行版 6.9.0 上创建应用程序，请使用以下映像。

Type	Image
Spark	`public.ecr.aws/emr-serverless/spark/emr-6.9.0:latest`
Hive	`public.ecr.aws/emr-serverless/hive/emr-6.9.0:latest`

先决条件

在创建 EMR Serverless 自定义映像之前，请完成以下先决条件。

1. 创建与启动 EMR 无服务器应用程序相同的 AWS 区域 存储库 Amazon ECR 存储库。要创建 Amazon ECR 私有存储库，请参阅创建私有存储库。
2. 要授予用户访问 Amazon ECR 存储库的权限，请向使用该存储库中的映像创建或更新 EMR Serverless 应用程序的用户和角色添加以下策略。

   JSON

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

   有关 Amazon ECR 基于身份的策略的更多示例，请参阅 Amazon Elastic Container Registry 基于身份的策略示例。

步骤 1：根据 EMR Serverless 基础映像创建自定义映像

首先，创建一个 Dockerfile，该文件以 FROM 指令开头，要求使用首选基础映像。在 FROM 指令之后，包含要对映像进行的任何修改。基础映像会自动将 USER 设置为 hadoop。此设置不具备您所包含的所有修改的权限。解决方法是，将 USER 设置为 root，修改映像，然后将 USER 重新设置为 hadoop:hadoop。要参考常见使用案例的示例，请参阅在 EMR Serverless 中使用自定义映像。

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

USER root
# MODIFICATIONS GO HERE

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

创建 Dockerfile 之后，使用以下命令构建映像。

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

步骤 2：在本地验证映像

EMR Serverless 提供了一个离线工具，可静态检查自定义映像，用来验证基本文件、环境变量和正确的映像配置。有关如何安装和运行该工具的信息，请参阅 Amazon EMR Serverless Image CLI。GitHub

安装工具后，运行以下命令验证映像：

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

输出类似于以下内容。

```
Amazon EMR Serverless - Image CLI
Version: 0.0.1
... Checking if docker cli is installed
```

```
... Checking Image Manifest
[INFO] Image ID: 9e2f4359cf5beb466a8a2ed047ab61c9d37786c555655fc122272758f761b41a
[INFO] Created On: 2022-12-02T07:46:42.586249984Z
[INFO] Default User Set to hadoop:hadoop : PASS
[INFO] Working Directory Set to  : PASS
[INFO] Entrypoint Set to /usr/bin/entrypoint.sh : PASS
[INFO] HADOOP_HOME is set with value: /usr/lib/hadoop : PASS
[INFO] HADOOP_LIBEXEC_DIR is set with value: /usr/lib/hadoop/libexec : PASS
[INFO] HADOOP_USER_HOME is set with value: /home/hadoop : PASS
[INFO] HADOOP_YARN_HOME is set with value: /usr/lib/hadoop-yarn : PASS
[INFO] HIVE_HOME is set with value: /usr/lib/hive : PASS
[INFO] JAVA_HOME is set with value: /etc/alternatives/jre : PASS
[INFO] TEZ_HOME is set with value: /usr/lib/tez : PASS
[INFO] YARN_HOME is set with value: /usr/lib/hadoop-yarn : PASS
[INFO] File Structure Test for hadoop-files in /usr/lib/hadoop: PASS
[INFO] File Structure Test for hadoop-jars in /usr/lib/hadoop/lib: PASS
[INFO] File Structure Test for hadoop-yarn-jars in /usr/lib/hadoop-yarn: PASS
[INFO] File Structure Test for hive-bin-files in /usr/bin: PASS
[INFO] File Structure Test for hive-jars in /usr/lib/hive/lib: PASS
[INFO] File Structure Test for java-bin in /etc/alternatives/jre/bin: PASS
[INFO] File Structure Test for tez-jars in /usr/lib/tez: PASS
----------------------------------------------------------------
Overall Custom Image Validation Succeeded.
----------------------------------------------------------------
```

步骤 3：将映像上传到 Amazon ECR 存储库

使用以下命令将 Amazon ECR 映像推送到 Amazon ECR 存储库。确保您拥有正确的 IAM 权限，可将映像推送到存储库。有关更多信息，请参阅《Amazon ECR 用户指南》中的推送映像。

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

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

步骤 4：使用自定义映像创建或更新应用程序

根据您想要的 AWS 管理控制台 应用程序启动方式选择 AWS CLI 选项卡或选项卡，然后完成以下步骤。

Console

1. 通过 /emr 登录 EMR Studio 控制台。https://console.aws.amazon.com导航到您的应用程序，或按照创建应用程序中的说明创建新的应用程序。
2. 要在创建或更新 EMR Serverless 应用程序时指定自定义映像，请在应用程序设置选项中选择自定义设置。
3. 在自定义映像设置部分，选择在此应用程序中使用自定义映像复选框。
4. 将 Amazon ECR 映像 URI 粘贴到映像 URI 字段。EMR Serverless 会将此映像用于应用程序的所有工作线程类型。或者，您可以选择不同的自定义图像，然后 URIs 为每种工作人员类型粘贴不同的 Amazon ECR 图像。

CLI

- 使用 `image-configuration` 参数创建应用程序。EMR Serverless 会将此设置应用于所有工作线程类型。

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

要为每种工作线程类型创建具有不同映像设置的应用程序，请使用 `worker-type-specifications` 参数。

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

```
            "imageUri": "aws-account-id.dkr.ecr.region.amazonaws.com/my-
repository:tag/@digest"
        }
    }
}'
```

要更新应用程序，请使用 `image-configuration` 参数。EMR Serverless 会将此设置应用于所有工作线程类型。

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

步骤 5：允许 EMR Serverless 访问自定义映像存储库

将以下资源策略添加到 Amazon ECR 存储库，以允许 EMR Serverless 服务主体使用来自该存储库的 `get`、`describe` 和 `download` 请求。

JSON

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

```
      }
    }
  ]
}
```

最佳安全实践是，将 aws:SourceArn 条件键添加到存储库策略。IAM 全局条件键 aws:SourceArn 可确保 EMR Serverless 仅将存储库用于应用程序 ARN。有关 Amazon ECR 存储库策略的更多信息，请参阅创建私有存储库。

注意事项和限制

在使用自定义映像时，请注意以下几点：

- 使用与应用程序类型（Spark 或 Hive）和发行版标签（例如 emr-6.9.0）相匹配的正确基础映像。
- EMR Serverless 会忽略 Docker 文件中的 [CMD] 或 [ENTRYPOINT] 指令。使用 Docker 文件中的常用指令，如 [COPY]、[RUN] 和 [WORKDIR]。
- 创建自定义映像时，请不要修改环境变量 JAVA_HOME、SPARK_HOME、HIVE_HOME 和 TEZ_HOME。
- 自定义图像的大小不能超过 10 GB。
- 如果修改 Amazon EMR 基础映像中的二进制文件或 jar 文件，这可能会导致应用程序或作业启动失败。
- Amazon ECR 存储库必须与您用于启动 EMR 无服务器应用程序的存储库相同 AWS 区域 。

为 EMR Serverless 应用程序配置 VPC 访问权限以连接数据

您可以配置 EMR Serverless 应用程序以连接到 VPC 内的数据存储，例如 Amazon Redshift 集群、Amazon RDS 数据库或具有 VPC 端点的 Amazon S3 存储桶。EMR Serverless 应用程序具有到 VPC 内数据存储的出站连接。默认情况下，EMR Serverless 会阻止对应用程序的入站访问和出站互联网访问，以增强安全性。

> **Note**
>
> 如果要对应用程序使用外部 Hive 元存储数据库，则必须配置 VPC 访问。有关如何配置外部 Hive 元存储的信息，请参阅元存储配置。

创建应用程序

在创建应用程序页面上，选择自定义设置，并指定 EMR Serverless 应用程序可以使用的 VPC、子网和安全组。

VPCs

选择包含数据存储的虚拟私有云（VPC）名称。创建应用程序页面列出了您选择的所有 VPCs 应用程序 AWS 区域。

子网

选择包含数据存储的 VPC 内的子网。创建应用程序页面列出了 VPC 内数据存储的所有子网。支持公有和私有子网。您可以将私有子网或公有子网传递给您的应用程序。选择使用公有子网还是私有子网需要注意一些相关事项。

对于私有子网：

- 关联的路由表不能有互联网网关。
- 对于与互联网的出站连接，如果需要，请使用 NAT 网关配置出站路由。要配置 NAT 网关，请参阅 NAT 网关。
- 要实现 Amazon S3 连接，请配置 NAT 网关或 VPC 端点。要配置 S3 VPC 端点，请参阅创建网关端点。
- 如果您配置了 S3 VPC 端点并附加了端点策略来控制访问，请按照使用托管存储的 EMR Serverless 日志记录中的说明为 EMR Serverless 提供存储和提供应用程序日志的权限。
- 要连接到 VPC AWS 服务 外部的其他人，例如与 Amazon DynamoDB 的连接，请配置 VPC 终端节点或 NAT 网关。要为 AWS 服务配置 VPC 端点，请参阅使用 VPC 端点。

> **Note**
>
> 当您在私有子网中设置 Amazon EMR Serverless 应用程序时，我们建议您也为 Amazon S3 设置 VPC 端点。如果您的 EMR Serverless 应用程序位于没有适用于 Amazon S3 的 VPC 端点的私有子网中，则会产生与 S3 流量关联的额外 NAT 网关费用。这是因为如果未配置 VPC 端点，则您的 EMR 应用程序和 Amazon S3 之间的流量不会停留在您的 VPC 中。

对于公有子网：

- 它们具有通向互联网网关的路由。
- 您必须确保正确配置安全组以控制出站流量。

工作线程可通过出站流量连接到 VPC 内的数据存储。默认情况下，EMR Serverless 会阻止对工作线程的入站访问。这是为了提高安全性。

当你使用时 AWS Config，EMR Serverless 会为每个工作人员创建一个弹性网络接口项目记录。为避免与该资源相关的成本，请考虑关闭接`AWS::EC2::NetworkInterface`入 AWS Config。

> **Note**
>
> 建议您在多个可用区中选择多个子网。这是因为您选择的子网决定了可供 EMR Serverless 应用程序启动的可用区。每个工作线程都在其启动的子网上使用一个 IP 地址。请确保指定的子网具有足够的 IP 地址，以容纳您计划启动的工作线程数量。有关子网规划的更多信息，请参阅 the section called “子网规划最佳实践”。

子网的注意事项和限制

- 带有公共子网的 EMR Serverless 不支持 Lake Formation。 AWS
- 公有子网不支持入站流量。

安全组

选择一个或多个可与数据存储通信的安全组。创建应用程序页面列出了 VPC 内的所有安全组。EMR Serverless 会将这些安全组与附加到 VPC 子网的弹性网络接口关联。

> **Note**
>
> 建议您为 EMR Serverless 应用程序创建单独的安全组。如果安全组具有向 0.0.0.0/0 或 ::/0 范围内的公共互联网开放的端口，则 EMR Serverless 不允许您创建/更新/启动应用程序。这提供增强安全、隔离，并使管理网络规则更加高效。例如，这会阻止向具有公有 IP 地址的工作线程发送的意外流量。例如，要与 Amazon Redshift 集群通信，请在 Redshift 和 EMR Serverless 安全组之间定义流量规则，如下例所示。

Example示例：与 Amazon Redshift 集群通信

1. 为从其中一个 EMR Serverless 安全组到 Amazon Redshift 安全组的入站流量添加规则。

Type	协议	端口范围	来源
所有 TCP	TCP	5439	`emr-serverless-security-group`

2. 为来自其中一个 EMR Serverless 安全组的出站流量添加规则。通过两种方式之一来执行此操作。首先，向所有端口开放出站流量。

Type	协议	端口范围	目标位置
所有流量	TCP	ALL	0.0.0.0/0

或者，您可以限制到 Amazon Redshift 集群的出站流量。仅当应用程序必须与 Amazon Redshift 集群通信而无其他情况时，这才有用。

Type	协议	端口范围	来源
所有 TCP	TCP	5439	`redshift-security-group`

配置应用程序

您可以在配置应用程序页面上更改现有 EMR Serverless 应用程序的网络配置。

访问作业运行详细信息

在作业运行详细信息页面上，访问作业在特定运行中使用的子网。请注意，作业只能在从指定子网中选择的一个子网中运行。

子网规划最佳实践

AWS 资源是在子网中创建的，子网是 Amazon VPC 中可用 IP 地址的子集。例如，网络掩码为 /16 的 VPC 最多有 65536 个可用 IP 地址，可使用子网掩码将其分解为多个较小的网络。例如，您可以将此范围拆分为两个子网，每个子网使用 /17 掩码和 32768 个可用 IP 地址。子网位于可用区内，不能跨可用区。

在设计子网时，应考虑 EMR Serverless 应用程序扩展限制。例如，如果您的应用程序需要 4 个 vCpu 工作线程，并可以纵向扩展到 4,000 个 vCpu，则最多需要 1,000 个工作线程，共计 1,000 个网络接口。建议您在多个可用区创建子网。这样，EMR Serverless 在可用区发生故障时可以重试作业，或在其他可用区中配置预初始化容量（这种情况不太可能发生）。因此，至少两个可用区中的每个子网应具有超过 1000 个可用 IP 地址。

您需要掩码小于或等于 22 的子网才能预置 1000 个网络接口。任何大于 22 的掩码都不符合要求。例如，子网掩码 /23 提供 512 个 IP 地址，而掩码 /22 提供 1024 个 IP 地址，掩码 /21 提供 2048 个 IP 地址。以下是网络掩码为 /16 的 VPC 中掩码为 /22 的 4 个子网的示例，这些子网可以分配到不同的可用区。可用和可用 IP 地址之间存在五个差异，因为每个子网中的前四个 IP 地址和最后一个 IP 地址都由保留 AWS。

子网 ID	子网地址	子网掩码	IP 地址范围	可用 IP 地址	可用 IP 地址
1	10.0.0.0	255.255.252.0/22	10.0.0.0 - 10.0.3.255	1024	1,019
2	10.0.4.0	255.255.252.0/22	10.0.4.0 - 10.0.7.255	1024	1,019
3	10.0.8.0	255.255.252.0/22	10.0.8.0 - 10.0.11.255	1024	1,019
4	10.0.12.0	255.255.252.0/22	10.0.12.0 - 10.0.15.255	1024	1,019

您应该评估工作负载是否适合较大的工作线程。使用较大的工作线程需要的网络接口较少。例如，使用 16 个 vCpu 工作线程且应用程序扩展限制为 4,000 vCpu，最多需要 250 个工作线程，共计 250 个可用 IP 地址来预置网络接口。您需要多个可用区中掩码小于或等于 24 的子网才能预置 250 个网络接口。任何大于 24 的掩码都能提供少于 250 个 IP 地址。

如果多个应用程序共享子网，则每个子网的设计应考虑到所有应用程序的集体扩展限制。例如，如果您有 3 个应用程序请求 4 个 vCPU 工作线程，每个可以纵向扩展到 4000 个 vCPU，并且具有 12,000 个 vCPU 账户级服务配额，则每个子网需要 3000 个可用 IP 地址。如果要使用的 VPC 没有足够数量的 IP 地址，请尝试增加可用 IP 地址的数量。您可以通过关联其他无类别域间路由（CIDR）块与 VPC 来执行此操作。有关更多信息，请参阅 Amazon VPC 用户指南中的将更多 IPv4 CIDR 块与您的 VPC 关联起来。

您可以使用在线工具快速生成子网定义，并查看可用的 IP 地址范围。

Amazon EMR Serverless 架构选项

Amazon EMR Serverless 应用程序的指令集架构决定了应用程序用来运行作业的处理器类型。Amazon EMR 为应用程序提供了两种架构选项：x86_64 和 arm64。EMR Serverless 会在最新的实例可用时自动更新，因此您的应用程序始终可以使用新的实例，而无需进行额外的操作。

主题

- 使用 x86_64 架构
- 使用 arm64 架构（Graviton）
- 启动支持 Graviton 的新应用程序
- 将现有应用程序配置为使用 Graviton
- 使用 Graviton 的注意事项

使用 x86_64 架构

x86_64 架构也称为 x86 64 位或 x64。x86_64 是 EMR Serverless 应用程序的默认选项。该架构使用基于 x86 的处理器，并与大多数第三方工具和库兼容。

大多数应用程序都与 x86 硬件平台兼容，可在默认的 x86_64 架构上成功运行。但是，如果您的应用程序与 64 位 ARM 兼容，请切换到 arm64 以使用 Graviton 处理器来提高性能、计算能力和内存。与在 x86 架构上运行同等大小的实例相比，在 arm64 架构上运行实例的成本更低。

使用 arm64 架构（Graviton）

AWS Graviton 处理器由 64 位 ARM Neoverse 内核 AWS 进行定制设计，利用了 arm64 架构（也称为 Arch64 或 64 位 ARM）。EMR Serverless AWS 上提供的 Graviton 系列处理器包括 Graviton3 和 Graviton2 处理器。与在 x86_64 架构上运行的同等工作负载相比，这些处理器为 Spark 和 Hive 工作

负载提供了卓越的性价比。EMR Serverless 会在最新的处理器可用时自动更新，因此无需额外操作即可升级到最新的处理器。

启动支持 Graviton 的新应用程序

使用以下任一方法启动采用 arm64 架构的应用程序。

AWS CLI

要使用来自的 Graviton 处理器启动应用程序 AWS CLI，请在 ARM64 API 中指定为`architecture`参数。`create-application`在其他参数中为您的应用程序提供适当的值。

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

EMR Studio

要从 EMR Studio 使用 Graviton 处理器启动应用程序，请在创建或更新应用程序时选择 arm64 作为架构选项。

将现有应用程序配置为使用 Graviton

您可以将现有的 Amazon EMR 无服务器应用程序配置为使用 Graviton (arm64) 架构和 SDK 或 EMR Studio。 AWS CLI

将现有应用程序从 x86 转换为 arm64

1. 确认您使用的是支持 `architecture` 参数的 AWS CLI/SDK 的最新主版本。
2. 确认没有作业正在运行，然后停止应用程序。

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

3. 要更新应用程序以使用 Graviton，请在 `update-application` API 中为 `architecture` 参数指定 ARM64。

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

4. 要验证应用程序的 CPU 架构是否为现在 ARM64，请使用 `get-application` API。

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

5. 准备就绪后，重启应用程序。

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

使用 Graviton 的注意事项

在使用 arm64 启动支持 Graviton 的 EMR Serverless 应用程序之前,确认以下内容。

库兼容性

当您选择 Graviton（arm64）作为架构选项时，请确保第三方软件包和库与 64 位 ARM 架构兼容。有关如何将 Python 库打包到与所选架构兼容的 Python 虚拟环境中的信息，请参阅 将 Python 库与 EMR Serverless 结合使用。

要了解更多信息，请参阅上的 AWS Graviton 入门知识库。 GitHub该存储库包含一些基本资源，有助于您开始使用基于 ARM 的 Graviton。

EMR Serverless 应用程序的作业并发和排队

从 Amazon EMR 7.0.0 及更高版本开始，请为应用程序指定作业运行队列超时和并发配置。指定此配置后，Amazon EMR Serverless 会根据应用程序的并发利用率排队等候作业并开始执行。例如，如果作业运行并发数为 10，则应用程序一次只能运行 10 个作业。剩余的作业排队，直到其中一个正在运行的作业终止。如果队列超时时间提前到达，则作业超时。有关更多信息，请参阅作业运行状态。

并发和排队的主要好处

当需要提交大量作业时，作业并发和排队具有以下好处：

- 有助于控制并发执行的作业，从而有效利用应用程序级别的容量限制。
- 队列可能包含突发的作业提交，具有可配置的超时设置。

并发和排队入门

以下过程展示了实现并发和排队的几种不同方法。

使用 AWS CLI

1. 创建具有队列超时和并发作业运行的 Amazon EMR Serverless 应用程序：

```
aws emr-serverless create-application \
--release-label emr-7.0.0 \
--type SPARK \
--scheduler-configuration '{"maxConcurrentRuns": 1, "queueTimeoutMinutes": 30}'
```

2. 更新应用程序以更改作业队列超时和并发：

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

> **Note**
>
> 您可以更新现有应用程序以启用作业并发和排队。为此，应用程序的发行版标签必须为 emr-7.0.0 或更高版本。

使用 AWS 管理控制台

以下步骤展示了如何使用 AWS 管理控制台开始作业并发和排队：

1. 前往 EMR Studio，选择创建一个发行版标签为 EMR-7.0.0 或更高版本的应用程序。
2. 在应用程序设置选项下，选择使用自定义设置选项。
3. 在其他配置下，有一个作业运行设置部分。选择启用作业并发选项以启用该功能。

4. 选择后，请选择并发作业运行和队列超时，分别配置并发作业运行的数量和队列超时。如果没有为这些设置输入值，则使用默认值。
5. 选择创建应用程序，将在启用此功能的情况下创建应用程序。要进行验证，请转到控制面板，选择应用程序，然后在“属性”选项卡下检查，以确定该功能是否已启用。

配置后，请在启用此功能的情况下提交作业。

并发和排队注意事项

在实现并发和排队时，请注意以下几点：

- Amazon EMR 7.0.0 及更高版本支持作业并发和排队。
- Amazon EMR 7.3.0 及更高版本默认启用作业并发和排队。
- 您无法更新处于已启动状态的应用程序并发。
- `maxConcurrentRuns` 的有效范围为 1 到 1000，`queueTimeoutMinutes` 的有效范围为 15 到 720。
- 一个账户最多可以有 2000 个作业处于已排队状态。
- 并发和排队适用于批处理和流处理作业。不能用于交互式作业。有关更多信息，请参阅通过 EMR Studio 使用 EMR Serverless 运行交互式工作负载。

利用 EMR Serverless 将数据导入 S3 Express One Zone

对于 Amazon EMR 7.2.0 及更高版本，请将 EMR Serverless 与 Amazon S3 Express One Zone 存储类别结合使用，以提高运行作业和工作负载时的性能。S3 Express One Zone 是一种高性能的单区 Amazon S3 存储类，可为大多数延迟敏感型应用程序提供一致的个位数毫秒级数据访问。S3 Express One Zone 在其发布时，提供了 Amazon S3 中延迟最低、性能最高的云对象存储。

先决条件

- S3 Express One Zone 权限：当 S3 Express One Zone 最初在 S3 对象上调用 `GET`、`LIST` 或 PUT 等操作时，存储类会代表您调用 `CreateSession`。您的 IAM policy 必须允许 `s3express:CreateSession` 权限，S3A 连接器才能调用 `CreateSession` API。有关使用此权限的示例策略，请参阅开始使用 S3 Express One Zone。
- S3A 连接器：要将 Spark 配置为从使用 S3 Express One Zone 存储类的 Amazon S3 存储桶中访问数据，请使用 Apache Hadoop 连接器 S3A。要使用连接器，请确保所有 S3 都 URIs 使用该`s3a`方案。如果没有，请更改用于 `s3` 和 `s3n` 方案的文件系统实施。

要更改 `s3` 方案，请指定以下集群配置：

```
[
  {
    "Classification": "core-site",
    "Properties": {
      "fs.s3.impl": "org.apache.hadoop.fs.s3a.S3AFileSystem",
      "fs.AbstractFileSystem.s3.impl": "org.apache.hadoop.fs.s3a.S3A"
    }
  }
]
```

要更改 `s3n` 方案，请指定以下集群配置：

```
[
  {
    "Classification": "core-site",
    "Properties": {
      "fs.s3n.impl": "org.apache.hadoop.fs.s3a.S3AFileSystem",
      "fs.AbstractFileSystem.s3n.impl": "org.apache.hadoop.fs.s3a.S3A"
    }
```

```
  }
]
```

开始使用 S3 Express One Zone

按照以下步骤开始使用 S3 Express One Zone。

1. 创建 VPC 端点。将端点 `com.amazonaws.us-west-2.s3express` 添加到 VPC 端点。
2. 按照开始使用 Amazon EMR Serverless，创建 Amazon EMR 7.2.0 或更高版本的应用程序。
3. 配置应用程序以使用新建的 VPC 端点、私有子网组和安全组。
4. 为作业执行角色添加 `CreateSession` 权限。

   JSON

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

5. 运行作业。请注意，请使用 S3A 方案访问 S3 Express One Zone 存储桶。

```
aws emr-serverless start-job-run \
--application-id <application-id> \
--execution-role-arn <job-role-arn> \
--name <job-run-name> \
--job-driver '{
 "sparkSubmit": {
```

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

运行任务

预置应用程序后，请向应用程序提交作业。本节介绍如何使用 AWS CLI 来运行这些作业。本节还确定了 EMR Serverless 上可用的每种应用程序类型的默认值。

主题

任务运行状态

当您将作业运行提交到 Amazon EMR Serverless 作业队列时，作业运行将进入 SUBMITTED 状态。作业状态从 SUBMITTED 变为 RUNNING，直至达到 FAILED、SUCCESS 或 CANCELLING。

任务运行可具有以下状态：

州	说明
已提交	将作业运行提交到 EMR Serverless 时的初始作业状态。作业等待为应用程序安排。EMR

州	说明
	Serverless 开始确定作业运行的优先级并安排作业运行。
已排队	当应用程序级作业运行并发被完全占用时，作业运行将在此状态下等待。有关排队和并发的更多信息，请参阅 EMR Serverless 应用程序的作业并发和排队。
待处理	调度程序评估作业运行，确定应用程序运行的优先级和计划。
已安排	EMR Serverless 已安排应用程序的作业运行，并分配资源来执行作业。
正在运行	EMR Serverless 已分配作业最初需要的资源，作业正在应用程序中运行。在 Spark 应用程序中，这意味着 Spark 驱动程序进程处于 `running` 状态。
已失败	EMR Serverless 未能将作业运行提交到应用程序，或者未成功完成。有关此作业失败的其他信息，请参阅 `StateDetails` 。
成功	作业运行已成功完成。
正在取消	`CancelJobRun` API 已请求取消作业运行，或者作业运行已超时。EMR Serverless 正在尝试取消应用程序中的作业并释放资源。
已取消	作业运行已成功取消，使用的资源也已释放。

EMR Serverless 作业运行取消宽限期

在数据处理系统中，突然终止可能会导致资源浪费、操作不完整和潜在数据不一致。Amazon EMR Serverless 允许您在取消作业运行时指定宽限期。此功能允许在作业终止之前有时间进行适当的清理和完成正在进行的工作。

取消作业运行时，请使用参数 shutdownGracePeriodInSeconds 指定一个宽限期（以秒为单位），在此期间作业可以执行清理操作，然后再最终终止。批处理作业和流式处理作业的行为和默认设置有所不同。

批处理作业的宽限期

对于批处理作业，EMR Serverless 允许您实施在宽限期内执行的自定义清理操作。您可以将这些清理操作注册为应用程序代码中 JVM 关闭钩子的一部分。

默认行为

关闭的默认行为没有宽限期。它包含以下两个操作：

- 立即终止
- 资源立即释放

配置选项

您可以指定导致正常关闭的设置：

- 关闭宽限期有效范围：15-1800 秒（可选）
- 立即终止（没有任何宽限期）：0 秒

启用正常关闭

要实现批处理作业的正常关闭，请按照以下步骤进行操作：

1. 在应用程序代码中添加包含自定义关闭逻辑的关闭钩子。

   Example in Scala

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

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

   使用 ShutdownHookManager

Example in PySpark

```
import atexit

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

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

2. 指定取消作业时的宽限期，以便有时间执行之前添加的钩子

示例

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

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

流式处理作业的宽限期

在 Spark Structured Streaming 中，计算涉及从外部数据来源读取数据或向其写入数据，突然关闭可能会导致不需要的结果。流式处理作业以微批次处理数据，而中途中断这些操作可能会导致后续尝试出现重复处理。当以前微批次中的最新检查点未写入时会发生这种情况，导致在流式处理作业重新启动时再次处理相同的数据。这种重复处理不仅浪费计算资源，还会影响业务运营，因此避免突然关闭至关重要。

EMR Serverless 通过流式查询侦听器提供内置正常关闭支持。这样可以确保在作业终止之前正确完成正在进行的微批次。该服务会自动管理流式处理应用程序的微批次之间的正常关闭，确保当前微批次完成处理，正确写入检查点，以及在关闭过程中无需提取新数据即可干净地终止流式处理上下文。

默认行为

- 默认启用 120 秒宽限期。

- 内置的流式查询侦听器管理正常关闭。

配置选项

- 关闭宽限期有效范围：15-1800 秒（可选）
- 立即终止：0 秒

启用正常关闭

要实现流式处理作业正常关闭，请执行以下操作：

指定取消作业时的宽限期，以便有时间完成正在进行的微批次。

示例

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

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

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

添加自定义关闭钩子（可选）

虽然默认情况下，EMR Serverless 通过其内置的流式查询侦听器管理正常关闭，但是您可以选择为单个流式查询实现自定义关闭逻辑。EMR Serverless 将其优雅关闭侦听器注册为优先级 60（使用）。ShutdownHookManager由于优先级较高的钩子会优先运行，您可以注册优先级大于 60 的自定义清理操作，从而确保它们在 EMR Serverless 的关闭过程开始之前执行。

要添加自定义钩子，请参考本主题中的第一个示例，该示例展示了如何在应用程序代码中添加关闭钩子。这里，优先级为 100，大于 60。因此，此类关闭钩子会首先运行。

> **Note**
>
> 自定义关闭钩子是可选的，并非正常关闭功能所必需，EMR Serverless 会自动处理该功能。

宽限期费用和批处理持续时间

如果使用默认的宽限期值（120 秒）：

- 如果您的批处理持续时间少于 120 秒，您只需支付完成批处理所需的实际时间费用。
- 如果您的批处理持续时间超过 120 秒，则将按最大宽限期（120 秒）向您收费，但查询可能不会正常关闭，因为它将被强制终止。

要优化成本并确保正常关闭，请执行以下操作：

- 对于持续时间超过 120 秒的批处理：考虑增加宽限期以匹配您的批处理持续时间。
- 对于持续时间少于 120 秒的批处理：无需调整宽限期，因为您只需支付实际处理时间的费用。

注意事项

宽限期行为

- 宽限期为您注册的关闭钩子完成提供了时间。
- 即使在宽限期之前，一旦关闭钩子完成，作业也会立即终止。
- 如果清理作业超过宽限期，则该作业将被强制终止。

服务行为

- 宽限期关闭仅适用于处于 RUNNING 状态的作业。
- CANCELLING 状态下的后续取消请求将被忽略。
- 如果 EMR Serverless 由于内部服务错误而无法启动宽限期关闭：
  - 服务将重试最多 2 分钟。
  - 如果重试不成功，则作业将被强制终止。

计费

作业将按作业完全关闭之前所使用的计算资源计费，包括在宽限期内花费的任何时间。

从 EMR Studio 控制台运行作业

您可以向 EMR Serverless 应用程序提交作业运行，从 EMR Studio 控制台访问作业。要在 EMR Studio 控制台上创建或导航到 EMR Serverless 应用程序，请按照控制台入门中的说明操作。

提交作业

在提交作业页面上，按如下方式向 EMR Serverless 应用程序提交作业。

Spark

1. 在名称字段中，输入作业运行的名称。
2. 在运行时角色字段中，输入 EMR Serverless 应用程序在运行作业时可代入的 IAM 角色名称。要了解有关运行时角色的更多信息，请参阅 Amazon EMR Serverless 的作业运行时角色。
3. 在脚本位置字段中，输入要运行的脚本或 JAR 的 Amazon S3 位置。对于 Spark 作业，脚本可以是 Python（`.py`）文件或 JAR（`.jar`）文件。
4. 如果脚本位置是 JAR 文件，请在主类字段中输入作为作业入口的类名。
5. （可选）输入其余字段的值。

   - 脚本参数：输入要传递给主 JAR 或 Python 脚本的参数。您的代码会读取这些参数。用逗号分隔数组中的每个参数。
   - Spark 属性：展开 Spark 属性部分，在此字段中输入任何 Spark 配置参数。

     > **Note**
     >
     > 如果指定 Spark 驱动程序和执行程序的大小，请考虑内存开销。在属性 `spark.driver.memoryOverhead` 和 `spark.executor.memoryOverhead` 中指定内存开销值。内存开销的默认值为容器内存的 10%，最小为 384MB。执行程序内存和内存开销之和不能超过工作线程内存。例如，30GB 工作线程的最大 `spark.executor.memory` 必须为 27GB。

   - 作业配置：在此字段中指定任何作业配置。您可以使用这些作业配置覆盖应用程序的默认配置。以下示例说明如何覆盖 Spark 的默认设置，例如执行程序和驱动程序内存。

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

- 其他设置：激活或停用作为元存储的 AWS Glue Data Catalog，并修改应用程序日志设置。要了解有关元存储配置的更多信息，请参阅EMR Serverless 的元存储配置。要了解有关应用程序日志记录选项的更多信息，请参阅存储日志。
- 标签：为应用程序分配自定义标签。

6. 选择提交作业。

Hive

1. 在名称字段中，输入作业运行的名称。
2. 在运行时角色字段中，输入 EMR Serverless 应用程序在运行作业时可代入的 IAM 角色名称。
3. 在脚本位置字段中，输入要运行的脚本或 JAR 的 Amazon S3 位置。对于 Hive 作业，脚本必须是 Hive（`.sql`）文件。
4. （可选）输入其余字段的值。

   - 初始化脚本位置：输入在 Hive 脚本运行之前初始化表的脚本位置。
   - Hive 属性：展开 Hive 属性部分，在此字段中输入任何 Hive 配置参数。
   - 作业配置：指定任何作业配置。您可以使用这些作业配置覆盖应用程序的默认配置。对于 Hive 作业，`hive.exec.scratchdir` 和 `hive.metastore.warehouse.dir` 是 `hive-site` 配置中的必需属性。

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

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

- 其他设置-激活或停用 AWS Glue 数据目录作为元数据仓并修改应用程序日志设置。要了解有关元存储配置的更多信息，请参阅EMR Serverless 的元存储配置。要了解有关应用程序日志记录选项的更多信息，请参阅存储日志。
- 标签：为应用程序分配任何自定义标签。

5. 选择提交作业。

访问作业运行

在应用程序详细信息页面上的作业运行选项卡中，访问作业运行并对作业运行执行以下操作。

取消作业：要取消处于 RUNNING 状态的作业运行，请选择此选项。要了解有关作业运行转换的更多信息，请参阅任务运行状态。

克隆作业：要克隆之前的运行作业并重新提交，请选择此选项。

正在运行来自的作业 AWS CLI

您可以在 AWS CLI上创建、描述和删除单个作业。您还可以列出所有作业，以便一目了然地访问它们。

要提交新作业，请使用 `start-job-run`。提供要运行的应用程序的 ID 以及特定于作业的属性。有关 Spark 示例，请参阅 运行 EMR Serverless 作业时使用 Spark 配置。有关 Hive 示例，请参阅 运行 EMR Serverless 作业时使用 Hive 配置。此命令将返回 `application-id`、ARN 和新的 `job-id`。

每个作业运行都设定了超时时间。如果作业运行超过此持续时间，EMR Serverless 会自动将其取消。默认超时时间为 12 小时。开始作业运行时，请将此超时设置配置为符合作业要求的值。使用 `executionTimeoutMinutes` 属性配置此值。

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

要描述作业，请使用 `get-job-run`。此命令将返回特定于作业的配置和新作业的设置容量。

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

要列出作业，请使用 `list-job-runs`。此命令将返回一组简短的属性，包括作业类型、状态和其他高级属性。如果您不想访问所有作业，请指定要访问的最大作业数，最多 50 个。以下示例指定您想访问最后两次作业运行情况。

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

要取消作业，请使用 `cancel-job-run`。提供要取消的作业的 `application-id` 和 `job-id`。

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

有关如何从中运行作业的更多信息 AWS CLI，请参阅《EMR Serverles s API 参考》。

执行 IAM 策略

在 EMR Serverless 上提交作业运行时，除了执行角色之外，您还可以指定执行 IAM 策略。作业运行所采用的权限是执行角色中的权限与指定的执行 IAM 策略中的权限的交集。

开始使用

使用执行 IAM 策略的步骤：

创建一个 `emr-serverless` 应用程序或使用现有应用程序，然后运行以下 aws cli 以启动带有内联 IAM 策略的作业运行：

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

CLI 命令示例

如果机器上的 `policy.json` 文件中存储了以下策略：

JSON

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

```
      "Sid": "AllowS3Getobject"
    }
  ]
}
```

然后，我们可以使用以下 AWS CLI 命令使用此策略启动作业：

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

您也可以同时使用客户托管策略 AWS 和客户托管策略，通过以下方式指定它们 ARNs：

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

也可以在同一个请求 ARNs 中同时指定内联 IAM 策略和托管策略：

```
aws emr-serverless start-job-run --region us-west-2 \
      --application-id application-id \
      --execution-role-arn execution-role-arn \
      --job-driver job-driver-options
      --execution-iam-policy '{
          "policy": '$(jq -c '. | @json' policy.json)',
          "policyArns": [
          "arn:aws:iam::aws:policy/AmazonS3FullAccess",
          "arn:aws:iam::aws:policy/CloudWatchLogsFullAccess"
          ]
      }'
```

重要提示

- `execution-role-policy` 的 `policy` 字段最多可包含 2048 个字符。
- 在 `execution-iam-policy` 的 `policy` 字段中指定的内联 IAM 策略字符串必须符合 json 字符串标准，不能像前面的示例那样对换行符和引号进行转义。
- ARNs 可以将最多 10 个托管策略的列表指定为 `execution-iam-policy's` `policyArns` 字段的值。
- 托管策略 ARNs 必须是有效 AWS 或客户托管策略 ARN 的列表，当指定客户托管策略 ARN 时，该策略必须属于 EMR-S 的同一个 AWS 账户。 JobRun
- 当同时使用内联 IAM 策略和托管策略时，用于内联策略和托管策略的明文组合不能超过 2,048 个字符。
- 由此产生的权限 JobRun 是执行角色和指定执行 IAM 策略中的权限的交集。

策略交集

作业运行所采用的权限是执行角色中的权限与指定的执行 IAM 策略中的权限的交集。这意味着必须在两个地方都指定任何所需的许可才能起作用。 JobRun 但是，对于您不打算更新或覆盖的任何权限，可以在内联策略中指定额外一揽子允许语句。

示例

给定以下执行 IAM 角色策略：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:*"
      ],
      "Resource": [
        "*"
      ],
      "Sid": "AllowS3"
    },
```

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

以及以下内联 IAM 策略：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::my-test-bucket/tenant1",
        "arn:aws:s3:::my-test-bucket/tenant1/*"
      ],
      "Sid": "AllowS3Getobject"
    },
```

```
    {
      "Effect": "Allow",
      "Action": [
        "logs:*",
        "dynamodb:*"
      ],
      "Resource": [
        "*"
      ],
      "Sid": "AllowLOGS"
    }
  ]
}
```

由此产生的权限 JobRun 为::

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::my-test-bucket/tenant1",
        "arn:aws:s3:::my-test-bucket/tenant1/*"
      ],
      "Sid": "AllowS3Getobject"
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogGroups"
      ],
      "Resource": [
        "arn:aws:logs:us-west-2:123456789012:log-group::log-stream"
      ],
      "Sid": "AllowLOGSDescribeloggroups"
```

```
    },
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:DescribeTable"
      ],
      "Resource": [
        "arn:aws:dynamodb:*:*:table/MyCompany1table"
      ],
      "Sid": "AllowDYNAMODBDescribetable"
    }
  ]
}
```

使用随机排序优化的磁盘

在 Amazon EMR 7.1.0 及更高版本中，在运行 Apache Spark 或 Hive 作业时使用经过洗牌优化的磁盘（以提高每秒操作的性能），以加快数据移动速度并减少洗牌I/O-intensive workloads. Compared to standard disks, shuffle-optimized disks provide higher IOPS (I/O操作期间的延迟。随机排序优化的磁盘允许为每个工作线程附加高达 2TB 的磁盘大小，因此请根据工作负载要求配置适当的容量。

主要优势

随机排序优化的磁盘具有以下优势。

- 高 IOPS 性能：随机排序优化的磁盘具有比标准磁盘更高的 IOPS，可在 Spark 和 Hive 作业以及其他随机排序密集型工作负载期间实现更高效、更快速的数据随机排序。
- 磁盘容量更大：随机排序优化的磁盘支持每个工作线程 20GB 到 2TB 的磁盘大小，因此请根据工作负载选择合适的容量。

开始使用

要在工作流程中使用随机排序优化的磁盘，请参阅以下步骤。

Spark

1. 使用以下命令创建 EMR Serverless 7.1.0 版应用程序。

```
aws emr-serverless create-application \
```

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

2. 配置您的 Spark 作业，使其包含要在经过洗牌优化的磁盘上运行的参数`spark.emr-serverless.driver.disk.type` and/or `spark.emr-serverless.executor.disk.type`。您可以使用一个或两个参数，具体取决于您的用例。

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

有关更多信息，请参阅 Spark 作业属性。

Hive

1. 使用以下命令创建 EMR Serverless 7.1.0 版应用程序。

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

2. 配置您的 Hive 作业，使其包含要在经过洗牌`hive.driver.disk.type` and/or `hive.tez.disk.type`优化的磁盘上运行的参数。您可以使用一个或两个参数，具体取决于您的用例。

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

有关更多信息，请参阅 Hive 作业属性。

配置具有预初始化容量的应用程序

请参阅以下示例，创建 Amazon EMR 7.1.0 版应用程序。这些应用程序具有以下属性：

- 5 个预初始化的 Spark 驱动程序，每个都有 2 个 vCPU、4GB 内存和 50GB 随机排序优化的磁盘。
- 50 个预初始化的执行程序，每个都有 4 个 vCPU、8GB 内存和 500GB 随机排序优化的磁盘。

当应用程序运行 Spark 作业时，会先使用预初始化的工作线程，然后将按需工作线程扩展到最大容量 400 个 vCPU 和 1024GB 内存。您可以选择忽略 DRIVER 或 EXECUTOR 的容量。

Spark

```
aws emr-serverless create-application \
  --type "SPARK" \
  --name <my-application-name> \
  --release-label emr-7.1.0 \
  --initial-capacity '{
    "DRIVER": {
        "workerCount": 5,
        "workerConfiguration": {
            "cpu": "2vCPU",
            "memory": "4GB",
            "disk": "50GB",
            "diskType": "SHUFFLE_OPTIMIZED"
        }
    },
    "EXECUTOR": {
        "workerCount": 50,
        "workerConfiguration": {
            "cpu": "4vCPU",
            "memory": "8GB",
            "disk": "500GB",
            "diskType": "SHUFFLE_OPTIMIZED"
        }
    }
  }' \
  --maximum-capacity '{
    "cpu": "400vCPU",
    "memory": "1024GB"
  }'
```

Hive

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

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

为 Amazon EMR 使用无服务器存储 Serverless

在 Amazon EMR 7.12 及更高版本中，在运行 Apache Spark 任务时使用无服务器存储，以消除本地磁盘配置并降低数据处理成本，并防止由于磁盘容量限制而导致任务失败。Serverless 存储无需配置容量即可自动处理作业的洗牌、磁盘溢出和磁盘缓存操作，并免费存储中间数据。Amazon EMR Serverless 将中间数据存储在完全托管的无服务器存储中，该存储可根据工作负载需求自动扩展，并使 Spark 能够在空闲时立即释放计算工作人员，从而降低计算成本。

主要优势

EMR Serverless 的无服务器存储具有以下优势。

- 零配置存储 — 无服务器存储无需为每个应用程序或任务配置本地磁盘类型和大小。EMR Serverless 无需容量规划即可自动管理中间数据操作。
- 通过自动扩展防止任务失败 — 存储容量可根据工作负载需求自动扩展，防止因磁盘容量不足而导致任务失败。
- 降低数据处理成本 — 无服务器存储通过两种机制降低了处理成本。首先，中间数据存储是免费提供的，您只需为计算和内存资源付费。其次，将存储与 Spark 的动态资源分配分开后，Spark 能够在闲置时立即释放工作人员，而不是保留他们以将中间数据保留在本地磁盘上。这可以加快每个 Spark 阶段的横向扩展和缩小规模，从而降低后期阶段需要比初始阶段更少的工作人员的计算成本。

- 具有作业级隔离功能的加密存储 — 所有中间数据在传输过程中和静态数据都经过严格的作业级隔离。
- 精细访问控制支持 — 无服务器存储通过 AWS Lake Formation 集成支持精细的访问控制。

开始使用

要在 Spark 工作流程中使用适用于 EMR Serverless 的无服务器存储，请参阅以下步骤。

1. 创建 EMR 无服务器应用程序

   通过在 spark-defaults 分类中将 spark 属性`spark.aws.serverlessStorage.enabled`设置为 t rue，创建启用无服务器存储的 EMR Serverless 7.12（或更高版本）应用程序。

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

2. 开始一个 Spark 任务

   开始在您的应用程序上运行作业。EMR 的无服务器存储 Serverless 会自动处理中间数据操作，例如作业的随机播放。

```
aws emr-serverless start-job-run \
  --application-id <application-id> \
  --execution-role-arn <job-role-arn> \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "s3://<bucket>/script.py",
      "sparkSubmitParameters": "--conf spark.executor.cores=4
        --conf spark.executor.memory=20g
        --conf spark.driver.cores=4
        --conf spark.driver.memory=8g
        --conf spark.executor.instances=10"
```

```
    }
  }'
```

即使未在应用程序级别启用无服务器存储，您也可以在作业级别为 EMR Serverless 启用无服务器存储。这将启动启用了无服务器存储的工作节点来处理您的作业。您还可以通过将相同的 Spark 属性设置为 false **spark.aws.serverlessStorage.enabled** 来禁用特定作业的无服务器存储。

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

> **Note**
>
> 要继续使用传统的本地磁盘配置，请省略spark.aws.serverlessStorage.enabled配置或将其设置为 false。

注意事项和限制

- 发布版本 — Amazon EMR 7.12 及更高版本支持无服务器存储。
- 数据量限制 — 每个作业每次运行最多可以读取和写入 200 GB 的中间数据。超过此限制的任务将失败，并显示一条错误消息，表明已达到无服务器存储限制。
- 任务执行超时 — 无服务器存储支持执行超时长达 24 小时的作业。为更长的执行超时配置的作业将失败并显示错误消息。
- 预初始化容量 — 预初始化的容量工作人员不支持无服务器存储。配置预初始化容量时，只有在作业级别明确禁用无服务器存储的作业才会使用该容量。启用无服务器存储的作业将始终按需配置新的工作人员，并且无论应用程序级别的配置如何，都不会使用任何预先初始化的容量。
- 工作负载类型-流式处理和交互式作业不支持无服务器存储。

- 工作器配置 — 使用 1 或 2 v CPUs 的工作器不支持无服务器存储。

支持 AWS 区域

EMR Serverless 支持以下区域的无服务器存储：

- 美国东部（弗吉尼亚州北部）
- 美国西部（俄勒冈州）
- 欧洲地区（爱尔兰）

处理连续流数据的流处理作业

EMR Serverless 中的流处理作业是一种作业模式，可让您近乎实时地分析和处理流数据。这些长时间运行的作业会轮询流数据，并在数据到达时持续处理结果。流处理作业最适合需要实时数据处理的任务，如近实时分析、欺诈检测和推荐引擎。EMR Serverless 流处理作业提供了优化功能，如内置作业弹性、实时监控、增强的日志管理以及与流连接器的集成。

以下是流处理作业的一些用例：

- 近实时分析：Amazon EMR Serverless 中的流处理作业让您可以近乎实时地处理流数据，以便对日志数据、传感器数据或点击流数据等连续数据流执行实时分析，从而根据最新信息获得见解并及时做出决策。
- 欺诈检测：在分析数据流并发现可疑模式或异常情况时，可使用流处理作业对金融交易、信用卡业务或在线活动进行近乎实时的欺诈检测。
- 推荐引擎：流处理作业可以处理用户活动数据并更新推荐模型。这样可根据行为和偏好进行个性化的实时推荐。
- 社交媒体分析：流处理作业可以处理推文、评论和帖子等社交媒体数据，让组织近乎实时地监控趋势、分析情感和管理品牌声誉。
- 物联网 (IoT) 分析：流处理作业可以处理和分析来自物联网设备、传感器和连接机器的高速数据流，以便运行异常检测、预测性维护和其他物联网分析用例。
- 点击流分析：流处理作业可以处理和分析来自网站或移动应用程序的点击流数据。使用此类数据的企业可以进行分析，以深入了解用户行为、提供个性化用户体验、优化营销活动。
- 日志监控和分析：流处理作业还可以处理来自服务器、应用程序和网络设备的日志数据。为您提供异常检测、故障排除、系统运行状况和性能。

主要优势

EMR Serverless 中的流处理作业会自动提供作业弹性，这是以下因素组合的结果：

- 自动重试：EMR Serverless 会自动重试任何失败的作业，无需手动输入。
- 可用区（AZ）弹性：如果原始可用区出现问题，EMR Serverless 会自动将流处理作业切换到运行状况良好的可用区。
- 日志管理：
  - 日志轮换：为了更有效地管理磁盘存储，EMR Serverless 会定期轮换长时间流处理作业的日志。这样可防止可能占用磁盘空间的日志累积。
  - 日志压缩：帮助您有效管理和优化托管持久性中的日志文件。在使用托管 Spark History Server 时，压缩还可以改善调试体验。

支持的数据来源和数据接收器

EMR Serverless 可与多个输入数据来源和输出数据接收器配合使用：

- 支持的输入数据来源：Amazon Kinesis Data Streams、Amazon Managed Streaming for Apache Kafka 和自我管理的 Apache Kafka 集群。默认情况下，Amazon EMR 7.1.0 及更高版本包含 Amazon Kinesis Data Streams 连接器，因此您无需构建或下载任何其他软件包。
- 支持的输出数据接收器 — AWS Glue 数据目录表、亚马逊 S3、亚马逊 Redshift、MySQL、PostgreSQL 甲骨文、甲骨文、微软 SQL、Apache Iceberg、Delta Lake 和 Apache Hudi。

注意事项和限制

使用流处理作业时，请记住以下注意事项和限制。

- Amazon EMR 7.1.0 及更高版本支持流处理作业。
- EMR Serverless 期望流处理作业能长时间运行，因此无法设置执行超时来限制作业的运行时间。
- 流处理作业仅与 Spark 引擎兼容，该引擎构建在结构化流框架之上。
- EMR Serverless 会无限期重试流处理作业，您无法自定义最大尝试次数。如果失败的尝试次数超过了每小时窗口内的阈值，则会自动包含防抖动功能。默认阈值为一小时内 5 次失败尝试。您可以将此阈值配置为 1 到 10 次尝试。有关更多信息，请参阅 Job resiliency。

- 流处理作业具有检查点来保存运行时状态和进度，因此 EMR Serverless 可以从最新的检查点恢复流处理作业。有关更多信息，请参阅 Apache Spark 文档中的 Recovering from failures with Checkpointing。

开始使用流处理作业

请参阅以下说明，了解如何开始使用流处理作业。

1. 要创建应用程序，请参阅开始使用 Amazon EMR Serverless。请注意，您的应用程序必须运行 Amazon EMR 7.1.0 或更高版本。
2. 应用程序准备就绪后，将mode参数设置为STREAMING以提交流媒体作业，类似于以下 AWS CLI 示例。

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

支持的流处理连接器

流处理连接器有助于从流处理源读取数据，也可以将数据写入流处理接收器。

以下是支持的流处理连接器：

Amazon Kinesis Data Streams 连接器

适用于 Apache Spark 的 Amazon Kinesis Data Streams 连接器支持构建流处理应用程序和管道，这些应用程序和管道使用来自 Amazon Kinesis Data Streams 的数据并向其写入数据。该连接器支持增强

的扇出消耗，每个分片的专用读取吞吐率高达 2MB/秒。默认情况下，Amazon EMR Serverless 7.1.0 及更高版本包含该连接器，因此您无需构建或下载任何其他软件包。有关连接器的更多信息，请参阅上的 spark-sql-kinesis-connector页面 GitHub。

以下示例展示了如何使用 Kinesis Data Streams 连接器依赖项启动作业运行。

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

要连接到 Kinesis Data Streams，请将 EMR Serverless 应用程序配置为 VPC 访问，并使用 VPC 端点允许私有访问，或使用 NAT 网关进行公有访问。有关更多信息，请参阅配置 VPC 访问。您还必须确保作业运行时角色拥有访问所需数据流的必要读写权限。要了解有关如何配置作业运行时角色的更多信息，请参阅 Amazon EMR Serverless 的作业运行时角色。有关所有必需权限的完整列表，请参阅上的 spark-sql-kinesis-connector 页面 GitHub。

Apache Kafka 连接器

适用于 Spark 结构化流处理的 Apache Kafka 连接器是来自 Spark 社区的开源连接器，可在 Maven 存储库中使用。此连接器有助于 Spark 结构化流处理应用程序从自我管理的 Apache Kafka 和 Amazon Managed Streaming for Apache Kafka 读取数据并向其写入数据。有关连接器的更多信息，请参阅 Apache Spark 文档中的结构化流处理 + Kafka 集成指南。

以下示例演示了如何在作业运行请求中包含 Kafka 连接器。

```
aws emr-serverless start-job-run \
--application-id <APPLICATION_ID> \
```

```
--execution-role-arn <JOB_EXECUTION_ROLE> \
--mode 'STREAMING' \
--job-driver '{
    "sparkSubmit": {
        "entryPoint": "s3://<Kafka-streaming-script>",
        "entryPointArguments": ["s3://<DOC-EXAMPLE-BUCKET-OUTPUT>/output"],
        "sparkSubmitParameters": "--conf spark.executor.cores=4
                --conf spark.executor.memory=16g
                --conf spark.driver.cores=4
                --conf spark.driver.memory=16g
                --conf spark.executor.instances=3
                --packages org.apache.spark:spark-sql-
kafka-0-10_2.12:<KAFKA_CONNECTOR_VERSION>"
    }
}'
```

Apache Kafka 连接器版本取决于 EMR Serverless 发行版和相应的 Spark 版本。要查找正确的 Kafka 版本，请参阅结构化流处理 + Kafka 集成指南。

要将 Amazon Managed Streaming for Apache Kafka 与 IAM 身份验证结合使用，请包含另一个依赖项，以使 Kafka 连接器能够通过 IAM 连接到 Amazon MSK。有关更多信息，请参阅上的aws-msk-iam-auth 存储库 GitHub。您还必须确保作业运行时角色拥有必要的 IAM 权限。以下示例演示了如何将连接器与 IAM 身份验证结合使用。

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

要使用 Kafka 连接器和 Amazon MSK 中的 IAM 身份验证库，请为 EMR Serverless 应用程序配置 VPC 访问。子网必须能够访问互联网，并使用 NAT 网关来访问 Maven 依赖项。有关更多信息，请参阅配置 VPC 访问。子网必须连接网络才能访问 Kafka 集群。无论您的 Kafka 集群是自我管理，还是使用 Amazon Managed Streaming for Apache Kafka，都是如此。

流处理作业日志管理

流处理作业支持 Spark 应用程序日志和事件日志的日志轮换，以及 Spark 事件日志的日志压缩。这可以帮助您有效管理资源。

日志轮换

流处理作业支持 Spark 应用程序日志和事件日志的日志轮换。日志轮换可防止长时间流处理作业生成大型日志文件，占用可用磁盘空间。日志轮换可帮助您节省磁盘存储空间，并防止由于磁盘空间不足而导致作业失败。有关更多信息，请参阅轮换日志。

日志压缩

当托管日志可用时，流处理作业还支持对 Spark 事件日志进行日志压缩。有关托管日志记录的更多详细信息，请参阅使用托管存储进行日志记录。流处理可以长时间运行，事件数据量会随着时间的推移而增加，并显著增加日志文件大小。Spark History Server 会读取这些事件，将其加载到 Spark 应用程序 UI 的内存中。此过程可能会产生高延迟和高成本，尤其是当 Amazon S3 中存储的事件日志非常大时。

日志压缩可减小事件日志的大小，因此 Spark History Server 在任何时候都不必加载超过 1GB 的事件日志。有关更多信息，请参阅 Apache Spark 文档中的监控和仪表。

运行 EMR Serverless 作业时使用 Spark 配置

您可以在 `type` 参数设置为 `SPARK` 的情况下在应用程序上运行 Spark 作业。作业必须与适用于 Amazon EMR 发行版的 Spark 版本兼容。例如，当您在 Amazon EMR 发行版 6.6.0 上运行作业时，作业必须与 Apache Spark 3.2.0 兼容。有关每个发行版的应用程序版本信息，请参阅 Amazon EMR Serverless 发行版。

Spark 作业参数

使用 `StartJobRun API` 运行 Spark 作业时，请指定以下参数。

必填参数

- Spark 作业运行时角色

- Spark 作业驱动程序参数
- Spark 配置覆盖参数
- Spark 动态资源分配优化

Spark 作业运行时角色

使用 **executionRoleArn** 指定应用程序用来执行 Spark 作业的 IAM 角色 ARN。此角色必须具有以下权限：

- 从数据驻留的 S3 存储桶或其他数据来源读取数据
- 从 PySpark 脚本或 JAR 文件所在的 S3 存储桶或前缀中读取
- 写入要写入最终输出的 S3 存储桶
- 将日志写入 S3MonitoringConfiguration 指定的 S3 存储桶或前缀
- 访问 KMS 密钥（如果使用 KMS 密钥加密 S3 存储桶中的数据）
- 如果你使用 sparkS AWS QL，则可以访问 Glue 数据目录

如果 Spark 作业从/向其他数据来源读取/写入数据，请在此 IAM 角色中指定相应的权限。如果不向 IAM 角色提供这些权限，作业可能会失败。有关更多信息，请参阅 Amazon EMR Serverless 的作业运行时角色 和 存储日志。

Spark 作业驱动程序参数

使用 **jobDriver** 为作业提供输入。对于要运行的作业类型，作业驱动程序参数只接受一个值。对于 Spark 作业，参数值为 sparkSubmit。您可以使用此作业类型通过 Spark 提交来运行 Scala PySpark、Java 和任何其他支持的作业。Spark 作业具有以下参数：

- **sparkSubmitParameters**：这些是您希望发送给作业的其他 Spark 参数。使用此参数可覆盖默认 Spark 属性，例如驱动程序内存或执行程序数量，如 --conf 或 --class 参数中定义的属性。
- **entryPointArguments**：这是您希望传递给主 JAR 或 Python 文件的参数数组。您应该使用 Entrypoint 代码来处理读取这些参数。用逗号分隔数组中的每个参数。
- **entryPoint**：这是 Amazon S3 中对要运行的主 JAR 或 Python 文件的引用。如果您正在运行 Scala 或 Java JAR，请使用 --class 参数指定 SparkSubmitParameters 中的主入口类。

有关更多信息，请参阅使用 spark-submit 启动应用程序。

Spark 配置覆盖参数

使用 **configurationOverrides** 覆盖监控级别和应用程序级别配置属性。该参数接受包含以下两个字段的 JSON 对象：

- **monitoringConfiguration**：使用该字段指定希望 EMR Serverless 作业存储 Spark 作业日志的 Amazon S3 URL（s3MonitoringConfiguration）。请确保您创建此存储桶时使用的存储桶与托管应用程序的存储桶相同，且运行任务的 AWS 区域 位置相同。 AWS 账户
- **applicationConfiguration**：您可以在此字段中提供一个配置对象，来覆盖应用程序的默认配置。您可以使用简写语法提供配置，或可引用 JSON 文件中的配置对象。配置对象包含分类、属性和可选的嵌套配置。属性由您希望在该文件中覆盖的设置组成。您可以在一个 JSON 对象中为多个应用程序指定多个分类。

> **Note**
>
> 可用的配置分类因特定的 EMR Serverless 发行版而异。例如，自定义 Log4j `spark-driver-log4j2` 和 `spark-executor-log4j2` 的分类仅适用于 6.8.0 及更高版本。

如果在应用程序覆盖和 Spark 提交参数中使用相同的配置，则 Spark 提交参数优先。配置按优先级从高到低排列如下：

- EMR Serverless 在创建 `SparkSession` 时提供的配置。
- 使用 `--conf` 参数作为 `sparkSubmitParameters` 的一部分提供的配置。
- 启动作业时，作为应用程序覆盖的一部分提供的配置。
- 创建应用程序时，作为 `runtimeConfiguration` 的一部分提供的配置。
- Amazon EMR 对发行版使用的优化配置。
- 应用程序的默认开源配置。

有关在应用程序级别声明配置以及在作业运行期间覆盖配置的更多信息，请参阅 EMR Serverless 的默认应用程序配置。

Spark 动态资源分配优化

使用 `dynamicAllocationOptimization` 优化 EMR Serverless 中的资源使用情况。在 Spark 配置分类中将此属性设置为 `true` 表示 EMR Serverless 需要优化执行程序资源分配，以便 Spark 请求

和取消执行程序的速率与 EMR Serverless 创建和释放工作线程的速率更好地保持一致。这样，EMR Serverless 可以更好地跨阶段重用工作线程，从而在运行多阶段作业时降低成本，同时性能保持不变。

此属性适用于所有 Amazon EMR 发行版。

以下是使用 `dynamicAllocationOptimization` 进行配置分类的示例。

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

如果使用动态分配优化，请考虑以下几点：

- 此优化适用于您为其启用了动态资源分配的 Spark 作业。
- 为实现最佳成本效益，我们建议根据工作负载情况，使用作业级别设置 `spark.dynamicAllocation.maxExecutors` 或应用程序级最大容量设置来配置工作线程的扩展上限。
- 在简单的作业中，您可能看不到成本改善。例如，如果您的作业在一个小数据集上运行或在一个阶段内完成运行，Spark 可能不需要大量的执行程序或多个扩展事件。
- 如果作业顺序是大阶段、小阶段然后又是大阶段，则作业运行时可能会出现回归。由于 EMR Serverless 能有效使用资源，可能会导致大阶段的可用工作线程减少，从而延长运行时间。

Spark 作业属性

下表列出了提交 Spark 作业时可以覆盖的可选 Spark 属性及其默认值。

可选 Spark 属性和默认值

Key	说明	默认 值
`spark.archives`	以逗号分隔的存档列表，Spark 将其提取到每个执行程序的工作目录中。支持的文件类型包括 `.jar`、 `.tar.gz`、`.tgz`	NULL

Key	说明	默认 值
	和 .zip。若要指定要提取的目录名，请在要提取的文件名后添加 #。例如 file.zip#directory 。	
spark.authenticate	开启 Spark 内部连接身份验证的选项。	TRUE
spark.driver.cores	驱动程序使用的核心数。	4
spark.driver.extraJavaOptions	Spark 驱动程序的额外 Java 选项。	NULL
spark.driver.memory	驱动程序使用的内存量。	14G
spark.dynamicAllocation.enabled	开启动态资源分配的选项。此选项可根据工作负载扩展或缩减在应用程序中注册的执行程序数量。	TRUE
spark.dynamicAllocation.executorIdleTimeout	在 Spark 将其删除之前，执行程序可保持空闲状态的时间长度。这仅适用于开启动态分配的情况。	60s
spark.dynamicAllocation.initialExecutors	开启动态分配时要运行的初始执行程序数量。	3
spark.dynamicAllocation.maxExecutors	如果开启动态分配，则表示执行程序数量的上限。	对于 6.10.0 及更高版本，infinity 对于 6.9.0 及更低版本，100
spark.dynamicAllocation.minExecutors	如果开启动态分配，则表示执行程序数量的下限。	0

Key	说明	默认 值
spark.emr-serverless.allocation.batch.size	在每个执行程序分配周期中请求的容器数。每个分配周期之间有一秒的间隔。	20
spark.emr-serverless.driver.disk	Spark 驱动程序磁盘。	20G
spark.emr-serverless.driverEnv. *[KEY]*	为 Spark 驱动程序添加环境变量的选项。	NULL
spark.emr-serverless.executor.disk	Spark 执行程序磁盘。	20G
spark.emr-serverless.memoryOverheadFactor	设置要添加到驱动程序和执行程序容器内存的内存开销。	0.1
spark.emr-serverless.driver.disk.type	附加到 Spark 驱动程序的磁盘类型。	标准
spark.emr-serverless.executor.disk.type	附加到 Spark 执行程序的磁盘类型。	标准
spark.executor.cores	每个执行程序使用的核心数。	4
spark.executor.extraJavaOptions	Spark 执行程序的额外 Java 选项。	NULL
spark.executor.instances	要分配的 Spark 执行程序容器的数量。	3
spark.executor.memory	每个执行程序使用的内存量。	14G
spark.executorEnv. *[KEY]*	向 Spark 执行程序添加环境变量的选项。	NULL

Key	说明	默认 值
spark.files	以逗号分隔的文件列表，这些文件放置在每个执行程序的工作目录中。您可以使用 SparkFiles.get(*fileName*) 在执行程序中访问这些文件的路径。	NULL
spark.hadoop.hive.metastore.client.factory.class	Hive 元存储实现类。	NULL
spark.jars	添加到驱动程序和执行程序的运行时类路径中的其他 jar 文件。	NULL
spark.network.crypto.enabled	开启基于 AES 的 RPC 加密的选项。这包括 Spark 2.2.0 中添加的身份验证协议。	FALSE
spark.sql.warehouse.dir	托管数据库和表的默认位置。	$PWD/spark-warehouse 的值
spark.submit.pyFiles	以逗号分隔的 .zip、.egg 或 .py 文件列表，这些文件放置在 Python 应用程序的 PYTHONPATH 中。	NULL

下表列出了默认的 Spark 提交参数。

默认 Spark 提交参数

Key	说明	默认 值
archives	以逗号分隔的存档列表，Spark 将其提取到每个执行程序的工作目录中。	NULL

Key	说明	默认 值
class	应用程序的主类（适用于 Java 和 Scala 应用程序）。	NULL
conf	任意 Spark 配置属性。	NULL
driver-cores	驱动程序使用的核心数。	4
driver-memory	驱动程序使用的内存量。	14G
executor-cores	每个执行程序使用的核心数。	4
executor-memory	执行程序使用的内存量。	14G
files	以逗号分隔的文件列表，这些文件放置在每个执行程序的工作目录中。您可以使用 SparkFiles.get(*fileName*) 在执行程序中访问这些文件的路径。	NULL
jars	以逗号分隔的 jar 文件列表，这些文件包含在驱动程序和执行程序的类路径上。	NULL
num-executors	要启动的执行程序数。	3
py-files	以逗号分隔的 .zip、.egg 或 .py 文件列表，这些文件放置在 Python 应用程序的 PYTHONPATH 中。	NULL
verbose	开启其他调试输出的选项。	NULL

资源配置最佳实践

通过 API 配置驱动程序和执行器资源 StartJobRun

> **Note**
>
> Spark 驱动程序和执行器内核以及内存属性（如果已指定）必须在 StartJobRun API 请求中直接指定。

通过这种方式配置资源可确保 EMR Serverless 在运行作业之前分配正确的资源。这与用户脚本中提供的设置（例如 .py 或 .jar 文件）形成对比，这些设置评估得太晚，因为驱动程序和执行程序工作线程有时会在脚本执行开始之前预置。在提交作业时，可通过两种受支持的方式配置这些资源：

选项 1：使用 sparkSubmitParameters

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

选项 2：使用 configurationOverrides 进行 spark-defaults 分类

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

Spark 示例

下面的示例展示了如何使用 StartJobRun API 运行 Python 脚本。有关使用此示例的 end-to-end教程，请参阅开始使用 Amazon EMR Serverless。您可以在 EMR Serverles s 示例存储库中找到有关如何运行 PySpark 作业和添加 Python 依赖关系的其他示例。 GitHub

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

下面的示例展示了如何使用 StartJobRun API 运行 Spark JAR。

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

运行 EMR Serverless 作业时使用 Hive 配置

您可以在 type 参数设置为 HIVE 的情况下在应用程序上运行 Hive 作业。作业必须与适用于 Amazon EMR 发行版的 Hive 版本兼容。例如，当您在 Amazon EMR 发行版 6.6.0 的应用程序上运行作业时，

作业必须与 Apache Hive 3.1.2 兼容。有关每个发行版的应用程序版本信息，请参阅 Amazon EMR Serverless 发行版。

Hive 作业参数

使用 `StartJobRun API` 运行 Hive 作业时，请指定以下参数。

必填参数

- Hive 作业运行时角色
- Hive 作业驱动程序参数
- Hive 配置覆盖参数

Hive 作业运行时角色

使用 **`executionRoleArn`** 指定应用程序用来执行 Hive 作业的 IAM 角色 ARN。此角色必须具有以下权限：

- 从数据驻留的 S3 存储桶或其他数据来源读取数据
- 从 Hive 查询文件和 init 查询文件驻留的 S3 存储桶或前缀读取数据
- 读取和写入 Hive Scratch 目录和 Hive Metastore 仓库目录驻留的 S3 存储桶
- 写入要写入最终输出的 S3 存储桶
- 将日志写入 `S3MonitoringConfiguration` 指定的 S3 存储桶或前缀
- 访问 KMS 密钥（如果使用 KMS 密钥加密 S3 存储桶中的数据）
- 访问 AWS Glue 数据目录

如果 Hive 作业从/向其他数据来源读取/写入数据，请在此 IAM 角色中指定相应的权限。如果不向 IAM 角色提供这些权限，作业可能会失败。有关更多信息，请参阅Amazon EMR Serverless 的作业运行时角色。

Hive 作业驱动程序参数

使用 **`jobDriver`** 为作业提供输入。对于要运行的作业类型，作业驱动程序参数只接受一个值。当您指定 `hive` 为作业类型时，EMR Serverless 会将 Hive 查询传递给 `jobDriver` 参数。Hive 作业具有以下参数：

- **`query`**：这是 Amazon S3 中对要运行的 Hive 查询文件的引用。

- **parameters**：这些是要覆盖的其他 Hive 配置属性。要覆盖属性，请将其作为 `--hiveconf` *property=value* 传递给此参数。要覆盖变量，请将其作为 `--hivevar` *key=value* 传递给此参数。
- **initQueryFile**：这是初始化 Hive 查询文件。Hive 会在查询之前运行该文件，并用其初始化表。

Hive 配置覆盖参数

使用 **configurationOverrides** 覆盖监控级别和应用程序级别配置属性。该参数接受包含以下两个字段的 JSON 对象：

- **monitoringConfiguration**：使用该字段指定希望 EMR Serverless 作业存储 Hive 作业日志的 Amazon S3 URL（`s3MonitoringConfiguration`）。请务必使用托管应用程序的存储桶以及运行任务的相同 AWS 区域 位置创建此存储桶。 AWS 账户
- **applicationConfiguration**：您可以在此字段中提供一个配置对象，来覆盖应用程序的默认配置。您可以使用简写语法提供配置，或可引用 JSON 文件中的配置对象。配置对象包含分类、属性和可选的嵌套配置。属性由您希望在该文件中覆盖的设置组成。您可以在一个 JSON 对象中为多个应用程序指定多个分类。

> **Note**
>
> 可用的配置分类因特定的 EMR Serverless 发行版而异。例如，自定义 Log4j `spark-driver-log4j2` 和 `spark-executor-log4j2` 的分类仅适用于 6.8.0 及更高版本。

如果在应用程序覆盖和 Hive 参数中传递相同的配置，则 Hive 参数优先。以下列表按优先级从高到低对配置进行排序。

- 作为 `--hiveconf` *property=value* 中 Hive 参数的一部分提供的配置。
- 启动作业时，作为应用程序覆盖的一部分提供的配置。
- 创建应用程序时，作为 `runtimeConfiguration` 的一部分提供的配置。
- Amazon EMR 为发行版分配的优化配置。
- 应用程序的默认开源配置。

有关在应用程序级别声明配置以及在作业运行期间覆盖配置的更多信息，请参阅 EMR Serverless 的默认应用程序配置。

Hive 作业属性

下表列出了在提交 Hive 作业时配置的必需属性。

必需 Hive 作业属性

设置	说明
hive.exec.scratchdir	EMR Serverless 在 Hive 作业执行期间创建临时文件的 Amazon S3 位置。
hive.metastore.warehouse.dir	Hive 中托管表数据库的 Amazon S3 位置。

下表列出了提交 Hive 作业时可以覆盖的可选 Hive 属性及其默认值。

可选 Hive 属性和默认值

设置	说明	默认 值
fs.s3.customAWSCredentialsProvider	您要使用的 AWS 凭证提供商。	com.amazonaws.auth.DefaultAWSCredentials ProviderChain
fs.s3a.aws.credentials.provider	您要用于 S3A 文件系统的 AWS 凭证提供程序。	com.amazonaws.auth.DefaultAWSCredentials ProviderChain
hive.auto.convert.join	根据输入文件大小，将普通连接自动转换为 mapjoin 的选项。	TRUE
hive.auto.convert.join.noconditionaltask	当 Hive 根据输入文件大小将普通连接转换为 mapjoin 时，开启优化的选项。	TRUE
hive.auto.convert.join.noconditionaltask.size	低于此大小时，连接会直接转换为 mapjoin。	最优值是基于 Tez 任务内存计算的

设置	说明	默认 值
hive.cbo.enable	使用 Calcite 框架开启成本优化的选项。	TRUE
hive.cli.tez.session.async	在 Hive 查询编译时启动后台 Tez 会话的选项。设置为 false 时，Tez AM 将在 Hive 查询编译后启动。	TRUE
hive.compute.query.using.stats	激活 Hive 以使用元存储中存储的统计数据回答某些查询的选项。对于基本统计数据，将 hive.stats.autogather 设置为 TRUE。要获取更高级的查询集合，请运行 analyze table queries。	TRUE
hive.default.fileformat	CREATE TABLE 语句的默认文件格式。如果您在 CREATE TABLE 命令中指定 STORED AS [FORMAT]，则可以显式覆盖此设置。	TEXTFILE
hive.driver.cores	Hive 驱动程序进程使用的核心数。	2
hive.driver.disk	Hive 驱动程序的磁盘大小。	20G
hive.driver.disk.type	Hive 驱动程序的磁盘类型。	标准
hive.tez.disk.type	Tez 工作线程的磁盘大小。	标准
hive.driver.memory	每个 Hive 驱动程序进程使用的内存量。Hive CLI 和 Tez Application Master 均摊此内存，并保留 20% 的余量。	6G

设置	说明	默认 值
hive.emr-serverless.launch.env.[*KEY*]	在 Hive 特定进程（如 Hive 驱动程序、Tez AM 和 Tez 任务）中设置 *KEY* 环境变量的选项。	
hive.exec.dynamic.partition	在 DML/DDL 中开启动态分区的选项。	TRUE
hive.exec.dynamic.partition.mode	指定是要使用严格模式还是非严格模式的选项。在严格模式下，至少指定一个静态分区，以免意外覆盖所有分区。在非严格模式下，允许所有分区都是动态的。	strict
hive.exec.max.dynamic.partitions	Hive 创建的最大动态分区数。	1000
hive.exec.max.dynamic.partitions.pernode	Hive 在每个 mapper 和 reducer 节点中创建的最大动态分区数。	100
hive.exec.orc.split.strategy	预计为下列值之一：BI、ETL 或 HYBRID。这不是用户级配置。BI 指定您希望在拆分生成上花费的时间比在查询执行上花费的时间更少。ETL 指定您希望在拆分生成上花费更多时间。HYBRID 指定根据启发式方法从上述策略中选择。	HYBRID
hive.exec.reducers.bytes.per.reducer	每个 reducer 的大小。默认值为 256MB。如果输入大小为 1G，作业将使用 4 个 reducer。	256000000

设置	说明	默认 值
hive.exec.reducers.max	最大 reducer 数。	256
hive.exec.stagingdir	存储临时文件的目录的名称，Hive 在表位置和 hive.exec.scratchdir 属性中指定的暂存目录位置中创建这些临时文件。	.hive-staging
hive.fetch.task.conversion	预计为下列值之一：NONE、MINIMAL 或 MORE。Hive 可以将选定查询转换为单个 FETCH 任务。这样可以最大限度地减少延迟。	MORE
hive.groupby.position.alias	使 Hive 在 GROUP BY 语句中使用列位置别名的选项。	FALSE
hive.input.format	默认输入格式。如果在使用 CombineHiveInputFormat 时遇到问题，请设置为 HiveInputFormat 。	org.apache.hadoop.hive.ql.io.CombineHiveInputFormat
hive.log.explain.output	为 Hive 日志中的任何查询开启扩展输出解释的选项。	FALSE
hive.log.level	Hive 日志记录级别。	INFO
hive.mapred.reduce.tasks.speculative.execution	为 reducer 开启推测性启动的选项。仅 Amazon EMR 6.10.x 及更低版本支持。	TRUE
hive.max-task-containers	最大并发容器数。将配置的 mapper 内存乘以该值，确定拆分计算和任务抢占使用的可用内存。	1000

设置	说明	默认 值
hive.merge.mapfiles	使小文件在仅映射作业结束时合并的选项。	TRUE
hive.merge.size.per.task	作业结束时合并文件的大小。	256000000
hive.merge.tezfiles	在 Tez DAG 结束时开启小文件合并的选项。	FALSE
hive.metastore.client.factory.class	生成对象以实现 IMetaStoreClient 接口的工厂类名称。	com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveClientFactory
hive.metastore.glue.catalogid	如果 AWS Glue Data Catalog 充当元存储但运行在与作业 AWS 账户 不同的位置，则为作业运行 AWS 账户 位置的 ID。	NULL
hive.metastore.uris	元存储客户端用于连接到远程元存储的 thrift URI。	NULL
hive.optimize.ppd	开启谓词下推的选项。	TRUE
hive.optimize.ppd.storage	开启对存储处理程序谓词下推的选项。	TRUE
hive.orderby.position.alias	使 Hive 在 ORDER BY 语句中使用列位置别名的选项。	TRUE
hive.prewarm.enabled	为 Tez 开启容器预热的选项。	FALSE
hive.prewarm.numcontainers	为 Tez 预热的容器数量。	10

设置	说明	默认 值
hive.stats.autogather	使 Hive 在 INSERT OVERWRITE 命令执行期间自动收集基本统计数据的选项。	TRUE
hive.stats.fetch.column.stats	关闭从元数据仓获取列统计数据的选项。当列数很大时，获取列统计数据的开销会很大。	FALSE
hive.stats.gather.num.threads	partialscan 和 noscan analyze 命令用于分区表的线程数。这仅适用于实现 StatsProvidingRecordReader 的文件格式（如 ORC）。	10
hive.strict.checks.cartesian.product	开启严格笛卡尔连接检查的选项。这些检查不允许使用笛卡尔乘积（交叉连接）。	FALSE
hive.strict.checks.type.safety	开启严格类型安全检查，并关闭 bigint 与 string 和 double 比较的选项。	TRUE
hive.support.quoted.identifiers	预期值为 NONE 或 COLUMN。NONE 意味着标识符中只有字母数字和下划线字符有效。COLUMN 意味着列名可以包含任何字符。	COLUMN
hive.tez.auto.reducer.parallelism	开启 Tez auto-reducer 并行度功能的选项。Hive 仍会估计数据大小并设置并行度估计。Tez 会对源顶点的输出大小进行采样，并在运行时根据需要调整估计值。	TRUE

设置	说明	默认 值
hive.tez.container.size	每个 Tez 任务进程使用的内存量。	6144
hive.tez.cpu.vcores	每个 Tez 任务使用的核心数。	2
hive.tez.disk.size	每个任务容器的磁盘大小。	20G
hive.tez.input.format	Tez AM 中拆分生成的输入格式。	org.apache.hadoop.hive.ql.io.HiveInputFormat
hive.tez.min.partition.factor	Tez 在开启 auto-reducer 并行度时指定的 reducer 下限。	0.25
hive.vectorized.execution.enabled	开启查询执行的向量化模式的选项。	TRUE
hive.vectorized.execution.reduce.enabled	开启查询执行 reduce-side 的向量化模式的选项。	TRUE
javax.jdo.option.ConnectionDriverName	JDBC 元存储的驱动程序类名称。	org.apache.derby.jdbc.EmbeddedDriver
javax.jdo.option.ConnectionPassword	与元存储数据库关联的密码。	NULL
javax.jdo.option.ConnectionURL	JDBC 元存储的 JDBC 连接字符串。	jdbc:derby:;databaseName=metastore_db;create=true
javax.jdo.option.ConnectionUserName	与元存储数据库关联的用户名。	NULL

设置	说明	默认 值
mapreduce.input.fileinputformat.split.maxsize	当输入格式为 org.apache.hadoop.hive.ql.io.CombineHiveInputFormat 时，拆分计算时拆分的最大大小。值为 0 表示没有限制。	0
tez.am.dag.cleanup.on.completion	在 DAG 完成时开启随机排序数据清理的选项。	TRUE
tez.am.emr-serverless.launch.env.[*KEY*]	在 Tez AM 进程中设置 *KEY* 环境变量的选项。对于 Tez AM，此值将覆盖 hive.emr-serverless.launch.env.[*KEY*] 值。	
tez.am.log.level	EMR Serverless 传递给 Tez 应用程序主进程的根日志记录级别。	INFO
tez.am.sleep.time.before.exit.millis	EMR Serverless 应在 AM 关闭请求发出后的这段时间后推送 ATS 事件。	0
tez.am.speculation.enabled	使较慢任务推测性启动的选项。当某些任务因机器故障或速度缓慢而运行较慢时，这有助于减少作业延迟。仅 Amazon EMR 6.10.x 及更低版本支持。	FALSE
tez.am.task.max.failed.attempts	在任务失败之前，特定任务可以失败的最大尝试次数。此数字不计入手动终止的尝试次数。	3

设置	说明	默认 值
tez.am.vertex.cleanup.height	如果所有从属顶点都已完成，Tez AM 将删除顶点随机排序数据的距离。值为 0 时，此功能关闭。Amazon EMR 6.8.0 及更高版本支持此功能。	0
tez.client.asynchronous-stop	使 EMR Serverless 在结束 Hive 驱动程序之前推送 ATS 事件的选项。	FALSE
tez.grouping.max-size	分组拆分的大小上限（字节）。此限制可防止过大的拆分。	1073741824
tez.grouping.min-size	分组拆分的大小下限（字节）。此限制可防止过小的拆分。	16777216
tez.runtime.io.sort.mb	Tez 对输出排序时软缓冲区的大小。	最优值是基于 Tez 任务内存计算的
tez.runtime.unordered.output.buffer.size-mb	Tez 不直接写入磁盘时使用的缓冲区大小。	最优值是基于 Tez 任务内存计算的
tez.shuffle-vertex-manager.max-src-fraction	在 EMR Serverless 为当前顶点安排所有任务之前必须完成的源任务的比例（在 ScatterGather 连接的情况下）。当前顶点上准备安排的任务数量在 min-fraction 和 max-fraction 之间线性缩放。这将为默认值或 tez.shuffle-vertex-manager.min-src-fraction ，以较大者为准。	0.75

设置	说明	默认 值
tez.shuffle-vertex-manager.min-src-fraction	在 EMR Serverless 为当前顶点安排任务之前必须完成的源任务的比例（在 ScatterGather 连接的情况下）。	0.25
tez.task.emr-serverless.launch.env.[*KEY*]	在 Tez 任务进程中设置 *KEY* 环境变量的选项。对于 Tez 任务，此值将覆盖 hive.emr-serverless.launch.env.[*KEY*] 值。	
tez.task.log.level	EMR Serverless 传递给 Tez 任务的根日志记录级别。	INFO
tez.yarn.ats.event.flush.timeout.millis	AM 在关闭之前等待事件刷新的最长时间。	300000

Hive 作业示例

下面的代码示例展示了如何使用 StartJobRun API 运行 Hive 查询。

```
aws emr-serverless start-job-run \
    --application-id application-id \
    --execution-role-arn job-role-arn \
    --job-driver '{
        "hive": {
            "query": "s3://amzn-s3-demo-bucket/emr-serverless-hive/query/hive-
query.ql",
            "parameters": "--hiveconf hive.log.explain.output=false"
        }
    }' \
    --configuration-overrides '{
        "applicationConfiguration": [{
            "classification": "hive-site",
            "properties": {
```

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

您可以在 EMR Serverless GitHub 示例存储库中找到有关如何运行 Hive 作业的其他示例。

EMR Serverless 作业弹性

EMR Serverless 7.1.0 及更高版本包含对作业弹性的支持，可自动重试任何失败的作业，而无需任何手动输入。作业弹性的另一个好处是，如果一个可用区（AZ）出现任何问题，EMR Serverless 会将作业运行转移到其他可用区（AZ）。

要为作业启用作业弹性，请设置作业的重试策略。重试策略可确保 EMR Serverless 在作业失败时自动重启作业。批处理和流处理作业均支持重试策略，因此请根据自己的使用案例自定义作业弹性。下表对比了批处理和流处理作业在作业弹性方面的行为和差异。

	批处理作业	流处理作业
默认 行为	不重新运行作业。	始终重试运行作业，因为应用程序会在运行作业时创建检查点。
重试点	批处理作业没有检查点，因此 EMR Serverless 总是从头开始重新运行作业。	流处理作业支持检查点，因此请配置流查询，将运行时状态和进度保存到 Amazon S3 中的检查点位置。EMR Serverless 从检查点恢复作业运行。有关更多信息，请参阅 Apache Spark 文档中的使用检查点从故障中恢复。

	批处理作业	流处理作业
最大重试次数	最多允许重试 10 次。	流处理作业具有内置的防抖动控制，因此如果作业在一小时后继续失败，应用程序将停止重试作业。一小时内的默认重试次数为 5 次。您可以将重试次数配置为 1 或 10 之间。无法自定义最大尝试次数。值为 1 表示不重试。

当 EMR Serverless 尝试重新运行作业时，还会使用尝试编号对该作业编制索引，以便跟踪作业尝试期间的生命周期。

使用 EMR Serverless API 操作或 AWS CLI 更改工作弹性或访问与工作弹性相关的信息。有关更多信息，请参阅 EMR Serverless API 指南。

默认情况下，EMR Serverless 不会重新运行批处理作业。要启用批处理作业重试，请在开始批处理作业运行时配置 `maxAttempts` 参数。`maxAttempts` 参数仅适用于批处理作业。默认值为 1，表示不重新运行作业。可接受的值为 1 到 10（含）。

以下示例演示了如何在启动作业运行时指定最多尝试 10 次。

```
aws emr-serverless start-job-run
 --application-id <APPLICATION_ID> \
 --execution-role-arn <JOB_EXECUTION_ROLE> \
 --mode 'BATCH' \
 --retry-policy '{
    "maxAttempts": 10
 }' \
 --job-driver '{
    "sparkSubmit": {
         "entryPoint": "/usr/lib/spark/examples/jars/spark-examples-does-not-
exist.jar",
         "entryPointArguments": ["1"],
         "sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi"
     }
}'
```

如果流处理作业失败，EMR Serverless 会无限期重试。要防止因重复出现无法恢复的故障而导致中断，可使用 maxFailedAttemptsPerHour 为流处理作业重试配置防抖动控制。通过该参数，您可以指定在 EMR Serverless 停止重试前一小时内允许的最大失败尝试次数。默认为 5 次。可接受的值为 1 到 10（含）。

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

您还可以使用其他作业运行 API 操作来获取有关作业的信息。例如，将 attempt 参数与 GetJobRun 操作一起使用，以获取有关特定作业尝试的详细信息。如果不包含 attempt 参数，该操作将返回有关最新尝试的信息。

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

ListJobRunAttempts 操作将返回与作业运行相关的所有尝试的信息。

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

该GetDashboardForJobRun操作创建并返回一个 URL， UIs 用于访问应用程序以运行作业。attempt 参数允许您获取特定尝试的 URL。如果不包含 attempt 参数，该操作将返回有关最新尝试的信息。

```
aws emr-serverless get-dashboard-for-job-run \
```

```
    --application-id application-id \
    --job-run-id job-run-id \
    --attempt 1
```

使用重试策略监控作业

作业弹性支持还添加了新事件 EMR Serverless 作业运行重试。EMR Serverless 在每次重试作业时都会发布此事件。您可以使用此通知跟踪作业的重试次数。有关事件的更多信息，请参阅 Amazon EventBridge 事件。

使用重试策略记录日志

每次 EMR Serverless 重试作业时，都会生成自己的日志集。为了确保 EMR Serverless 能够在不覆盖任何日志 CloudWatch 的情况下成功将这些日志传送到 Amazon S3 和 Amazon，EMR Serverless 在 S3 日志路径和 CloudWatch 日志流名称的格式中添加了一个前缀，以包含任务的尝试次数。

下面是该格式的一个示例。

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

这种格式可确保 EMR Serverless 将每次尝试执行任务的所有日志发布到自己在 Amazon S3 中的指定位置和。 CloudWatch有关更多详细信息，请参阅存储日志。

> **Note**
>
> EMR Serverless 仅对所有流处理作业和任何启用重试的批处理作业使用此前缀格式。

EMR Serverless 的元存储配置

Hive 元存储是一个集中位置，用于存储表的结构信息，包括架构、分区名称和数据类型。通过 EMR Serverless，请将此表元数据保存在有权访问您的作业的元存储中。

对于 Hive 元存储，您有两个选项：

- AWS Glue 数据目录
- 外部 Apache Hive 元存储

使用 AWS Glue 数据目录作为元数据库

你可以将 Spark 和 Hive 作业配置为使用 Glue AWS 数据目录作为其元数据库。如果需要持久元存储，或不同应用程序、服务或 AWS 账户应用程序共享元存储，建议使用此配置。有关数据目录的更多信息，请参阅填充 AWS Glue 数据目录。有关 AWS Glue 定价的信息，请参阅 AWS Glue 定价。

您可以将 EMR Serverless 作业配置为在与您的应用程序 AWS 账户 相同或不同的应用程序中使用 AWS Glue 数据目录。 AWS 账户

配置 AWS Glue 数据目录

要配置数据目录，请选择要使用的 EMR Serverless 应用程序类型。

Spark

当你使用 EMR Studio 通过 EMR Serverless Spark 应用程序运行作业时，Glue 数据目录是默认的 AWS 元数据库。

使用 SDKs 或时 AWS CLI，在作业运行的`sparkSubmit`参数`com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveClientFactory`中将`spark.hadoop.hive.metastore.client.factory.class`配置设置为。下面的示例显示了如何使用 AWS CLI配置数据目录。

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

或者，您可以在 Spark 代码中创建新 `SparkSession` 时设置此配置。

```
from pyspark.sql import SparkSession

spark = (
```

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

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

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

Hive

对于 EMR Serverless Hive 应用程序，数据目录是默认的元存储。也就是说，当你在 EMR Serverless Hive 应用程序上运行作业时，Hive 在数据目录中以与应用程序相同的形式记录元存储信息。 AWS 账户 您不需要虚拟私有云（VPC）即可将数据目录用作元存储。

要访问 Hive 元数据仓表，请添加为 G AWS lue 设置 IAM 权限中概述的必需 G AWS lue 策略。

为 EMR Serverless AWS 和 Glue 数据目录配置跨账户访问权限

要为 EMR Serverless 设置跨账户访问权限，请先登录以下网站： AWS 账户

- `AccountA`— 这是您创建了 EMR 无服务器应用程序 AWS 账户 的地方。
- `AccountB`— 包 AWS 账户 含 AWS Glue 数据目录，你想让 EMR Serverless 作业运行访问该目录。

1. 确保 `AccountB` 中的管理员或其他授权身份将资源策略附加到 `AccountB` 中的数据目录。此策略授予 `AccountA` 特定的跨账户权限，以便对 `AccountB` 目录中的资源执行操作。

   JSON

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

```
      "Effect": "Allow",
      "Action": [
        "glue:GetDatabase",
        "glue:CreateDatabase",
        "glue:GetDataBases",
        "glue:CreateTable",
        "glue:GetTable",
        "glue:UpdateTable",
        "glue:DeleteTable",
        "glue:GetTables",
        "glue:GetPartition",
        "glue:GetPartitions",
        "glue:CreatePartition",
        "glue:BatchCreatePartition",
        "glue:GetUserDefinedFunctions"
      ],
      "Resource": [
        "arn:aws:glue:*:123456789012:catalog"
      ],
      "Sid": "AllowGLUEGetdatabase"
    }
  ]
}
```

2. 将 IAM 策略添加到 AccountA 中的 EMR Serverless 作业运行时角色，以便该角色可以访问 AccountB 中的数据目录资源。

   JSON

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

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

3. 启动作业运行。此步骤略有不同，具体取决于 AccountA 的 EMR Serverless 应用程序类型。

   Spark

   如以下示例所示传递 sparkSubmitParameters 中的 spark.hadoop.hive.metastore.glue.catalogid 属性。将 *AccountB-catalog-id* 替换为 AccountB 中数据目录的 ID。

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

Hive

如以下示例所示，在 hive-site 分类中设置 hive.metastore.glue.catalogid 属性。将 *AccountB-catalog-id* 替换为 AccountB 中数据目录的 ID。

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

使用 Glue 数据 AWS 目录时的注意事项

您可以在 Hive 脚本ADD JAR中 JARs 使用添加辅助工具。有关其他注意事项，请参阅使用 AWS Glue 数据目录时的注意事项。

使用外部 Hive 元存储

您可以配置 EMR Serverless Spark 和 Hive 作业，使其连接到外部 Hive 元存储，如 Amazon Aurora 或 Amazon RDS for MySQL。本节介绍了如何设置 Amazon RDS Hive 元存储、配置 VPC 和配置 EMR Serverless 作业以使用外部元存储。

创建外部 Hive 元存储

1. 按照创建 VPC 中的说明，创建具有私有子网的 Amazon Virtual Private Cloud（Amazon VPC）。

2. 使用新的 Amazon VPC 和私有子网创建 EMR Serverless 应用程序。当您使用 VPC 配置 EMR Serverless 应用程序时，首先会为您指定的每个子网预置一个弹性网络接口。然后，将您指定的安全组附加到该网络接口。这样就可以对应用程序进行访问控制。有关如何设置 VPC 的更多详细信息，请参阅 为 EMR Serverless 应用程序配置 VPC 访问权限以连接数据。
3. 在 Amazon VPC 的私有子网中创建 MySQL 或 Aurora PostgreSQL 数据库。有关如何创建 Amazon RDS 数据库的信息，请参阅创建 Amazon RDS 数据库实例。
4. 按照修改 Amazon RDS 数据库实例中的步骤修改 MySQL 或 Aurora 数据库的安全组，允许来自 EMR Serverless 安全组的 JDBC 连接。为从其中一个 EMR Serverless 安全组到 RDS 安全组的入站流量添加规则。

Type	协议	端口范围	来源
所有 TCP	TCP	3306	`emr-serverless-security-group`

配置 Spark 选项

使用 JDBC

要将 EMR Serverless Spark 应用程序配置为连接到基于 Amazon RDS for MySQL 或 Amazon Aurora MySQL 实例的 Hive 元存储，请使用 JDBC 连接。在作业运行的 `spark-submit` 参数中传递带有 `--jars` 的 `mariadb-connector-java.jar`。

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

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

以下代码示例是一个 Spark 入口点脚本，与 Amazon RDS 上的 Hive 元存储交互。

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

使用 thrift 服务

您可以将 EMR Serverless Hive 应用程序配置为连接到基于 Amazon RDS for MySQL 或 Amazon Aurora MySQL 实例的 Hive 元存储。为此，请在现有 Amazon EMR 集群的主节点上运行 thrift 服务器。如果您已经有一个 Amazon EMR 集群，其中包含用于简化 EMR Serverless 作业配置的 thrift 服务器，则此选项是理想之选。

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

以下代码示例是一个入口点脚本（`thriftscript.py`），使用 thrift 协议连接到 Hive 元存储。请注意，需要将 `hive.metastore.uris` 属性设置为从外部 Hive 元存储读取。

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

```
spark.stop()
```

配置 Hive 选项

使用 JDBC

如果要在 Amazon RDS MySQL 或 Amazon Aurora 实例上指定外部 Hive 数据库位置，可以覆盖默认的元存储配置。

> **Note**
>
> 在 Hive 中，您可以同时对元存储表执行多次写入。如果在两个作业之间共享元存储信息，请确保不要同时写入同一个元存储表，除非写入同一元存储表的不同分区。

在 `hive-site` 分类中设置以下配置以激活外部 Hive 元存储。

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

使用 thrift 服务器

你可以将你的 EMR Serverless Hive 应用程序配置为连接基于亚马逊 RDS for MySQL 或 Amazon Aurora My 的 Hive 元存储库。SQLinstance为此，请在现有 Amazon EMR 集群的主节点上运行 thrift 服务器。如果您已经有一个运行 thrift 服务器的 Amazon EMR 集群，希望使用 EMR Serverless 作业配置，则此选项是理想之选。

在 `hive-site` 分类中设置以下配置，以便 EMR Serverless 可以访问远程 thrift 元存储。请注意，必须将 `hive.metastore.uris` 属性设置为从外部 Hive 元存储读取。

```
{
    "classification": "hive-site",
```

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

在 EMR Serv AWS erless 上使用 Glue 多目录层次结构

您可以将 EMR 无服务器应用程序配置为使用 Glue 多目录层次结构 AWS 。以下示例展示了如何在 Glue 多目录层次结构中使用 EMR-S Spark。 AWS

要了解有关多目录层次结构的更多信息，请参阅 Amazon EMR 上的 G AWS lue 数据目录中使用 Spark 处理多目录层次结构。

使用带有 Iceberg 和 Glue 数据目录的 Redshift 托管存储 (RMS) AWS

以下内容演示了如何配置 Spark 以与 Iceberg 的 AWS Glue 数据目录集成：

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

集成后来自目录中的表的查询示例：

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

将 Redshift 托管存储 (RMS) 与 Iceberg REST API 和 Glue 数据目录配合使用 AWS

以下内容展示了如何将 Spark 配置为与 Iceberg REST 目录配合使用：

```
aws emr-serverless start-job-run \
--application-id application-id \
--execution-role-arn job-role-arn \
--job-driver '{
"sparkSubmit": {
"entryPoint": "s3://amzn-s3-demo-bucket/myscript.py",
    "sparkSubmitParameters": "
    --conf spark.sql.catalog.rms=org.apache.iceberg.spark.SparkCatalog
    --conf spark.sql.catalog.rms.type=rest
    --conf spark.sql.catalog.rms.warehouse=Glue RMS catalog ID
    --conf spark.sql.catalog.rms.uri=Glue endpoint URI/iceberg
    --conf spark.sql.catalog.rms.rest.sigv4-enabled=true
    --conf spark.sql.catalog.rms.rest.signing-name=glue
    --conf
 spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions"
    }
 }'
```

来自目录中的表的查询示例：

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

使用外部元存储时的注意事项

- 您可以将与 MariaDB JDBC 兼容的数据库配置为元存储。这些数据库的示例包括 RDS for MariaDB、MySQL 和 Amazon Aurora。
- 元存储不会自动初始化。如果元存储未使用适合 Hive 版本的架构初始化，请使用 Hive Schema Tool。
- EMR Serverless 不支持 Kerberos 身份验证。您不能将具有 Kerberos 身份验证的 thrift 元存储服务器与 EMR Serverless Spark 或 Hive 作业一起使用。
- 您必须配置 VPC 访问才能使用多目录层次结构。

从 EMR Serverless 访问另一个 AWS 账户中的 S3 数据

您可以从一个 AWS 账户运行 Amazon EMR 无服务器任务，并将其配置为访问属于另一个账户的 Amazon S3 存储桶中的数据。 AWS 本页介绍了如何配置从 EMR Serverless 对 S3 的跨账户访问。

在 EMR Serverless 上运行的任务可以使用 S3 存储桶策略或代入的角色从其他账户访问 Amazon S3 中的数据。 AWS

先决条件

要为 Amazon EMR Serverless 设置跨账户访问权限，请在登录两个账户的同时完成任务： AWS

- **`AccountA`**：这是您在其中创建 Amazon EMR Serverless 应用程序的 AWS 账户。在设置跨账户访问之前，请在此账户中做好以下准备：
  - Amazon EMR Serverless 应用程序，在其中运行作业。
  - 作业执行角色，拥有在应用程序中运行作业所需的权限。有关更多信息，请参阅Amazon EMR Serverless 的作业运行时角色。
- **`AccountB`**：该 AWS 账户包含您希望 Amazon EMR Serverless 作业访问的 S3 存储桶。

使用 S3 存储桶策略访问跨账户 S3 数据

要从 account A 访问 account B 中的 S3 存储桶，请将以下策略附加到 account B 中的 S3 存储桶。

JSON

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

```
      ]
    }
  ]
}
```

有关使用 S3 存储桶策略进行 S3 跨账户访问的更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的示例 2：存储桶所有者授予跨账户存储桶权限。

使用代入角色跨账户访问 S3 数据

为 Amazon EMR Serverless 设置跨账户访问权限的另一种方法是使用 () `AssumeRole` 中的 AWS Security Token Service 操作。AWS STS AWS STS 是一项全球 Web 服务，允许您为用户申请临时的、权限有限的证书。您可以使用通过 `AssumeRole` 创建的临时安全凭证对 EMR Serverless 和 Amazon S3 进行 API 调用。

以下步骤说明了如何使用代入角色从 EMR Serverless 跨账户访问 S3 数据：

1. 创建 Amazon S3 存储桶，即 `AccountB` 中的 *`cross-account-bucket`*。有关更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的创建存储桶。如果您希望跨账户访问 DynamoDB，还请在 `AccountB` 中创建 DynamoDB 表。有关更多信息，请参阅《Amazon DynamoDB 开发人员指南》中的创建 DynamoDB 表。
2. 在 `AccountB` 中创建 `Cross-Account-Role-B` IAM 角色，以便可以访问 *`cross-account-bucket`*。
   a. 登录 AWS 管理控制台 并打开 IAM 控制台，网址为https://console.aws.amazon.com/iam/。
   b. 选择 Roles (角色) 并创建新角色：`Cross-Account-Role-B`。有关如何创建 IAM 角色的更多信息，请参阅《IAM 用户指南》中的创建 IAM 角色。
   c. 创建 IAM policy 来指定针对 `Cross-Account-Role-B` 的权限以访问 *`cross-account-bucket`* S3 存储桶，如以下策略声明所示。然后将 IAM policy 附加到 `Cross-Account-Role-B`。有关更多信息，请参阅《IAM 用户指南》中的创建 IAM 策略。

   JSON

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

```
      ],
      "Resource": [
        "arn:aws:s3:::cross-account-bucket",
        "arn:aws:s3:::cross-account-bucket/*"
      ],
      "Sid": "AllowS3"
    }
  ]
}
```

如果需要访问 DynamoDB，请创建 IAM 策略，指定跨账户访问 DynamoDB 表的权限。然后将 IAM policy 附加到 `Cross-Account-Role-B`。有关更多信息，请参阅《IAM 用户指南》中的 Amazon DynamoDB：允许访问特定表。

以下是允许访问 DynamoDB 表 `CrossAccountTable` 的策略。

JSON

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

3. 编辑 `Cross-Account-Role-B` 角色的信任关系。
   a. 要配置角色的信任关系，请在 IAM 控制台中为您在步骤 2 中创建的 `Cross-Account-Role-B` 角色选择信任关系选项卡。
   b. 选择 Edit Trust Relationship (编辑信任关系)。
   c. 添加以下策略文档。这允许 `AccountA` 中的 `Job-Execution-Role-A` 代入 `Cross-Account-Role-B` 角色。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole"
      ],
      "Resource": "arn:aws:iam::123456789012:role/Job-Execution-Role-A",
      "Sid": "AllowSTSAssumerole"
    }
  ]
}
```

4. 授Job-Execution-Role-AAccountA予假设 AWS STS AssumeRole权限Cross-Account-Role-B。

   a. 在 AWS 账户的 IAM 控制台中AccountA，选择Job-Execution-Role-A。

   b. 添加以下 Job-Execution-Role-A 策略语句以便对 Cross-Account-Role-B 角色执行 AssumeRole 操作。

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole"
      ],
      "Resource": [
        "arn:aws:iam::123456789012:role/Cross-Account-Role-B"
      ],
      "Sid": "AllowSTSAssumerole"
    }
  ]
}
```

代入角色示例

使用单个代入角色访问账户中的所有 S3 资源，或者在 Amazon EMR 6.11 及更高版本中，配置多个 IAM 角色，以便在跨账户访问不同的 S3 存储桶时代入。

主题

使用单个代入角色访问 S3 资源

> **Note**
>
> 如果将作业配置为使用单个代入角色，整个作业中的所有 S3 资源都将使用该角色，包括 `entryPoint` 脚本。

如果您想使用单个代入角色访问账户 B 中的所有 S3 资源，请指定以下配置：

1. 将 EMRFS 配置 `fs.s3.customAWSCredentialsProvider` 指定为 `com.amazonaws.emr.AssumeRoleAWSCredentialsProvider`。
2. 对于 Spark，使用 `spark.emr-serverless.driverEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN` 和 `spark.executorEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN` 指定驱动程序和执行程序的环境变量。
3. 对于 Hive，使用 `hive.emr-serverless.launch.env.ASSUME_ROLE_CREDENTIALS_ROLE_ARN`、`tez.am.emr-serverless.launch.env.ASSUME_ROLE_CREDENTIALS_ROLE_ARN` 和 `tez.task.emr-serverless.launch.env.ASSUME_ROLE_CREDENTIALS_ROLE_ARN` 指定 Hive 驱动程序、Tez 应用程序主进程和 Tez 任务容器的环境变量。

以下示例展示了如何使用代入角色启动具有跨账户访问权限的 EMR Serverless 作业运行。

Spark

以下示例展示了如何使用代入角色启动有权跨账户访问 S3 的 EMR Serverless Spark 作业运行。

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

Hive

以下示例展示了如何使用代入角色启动有权跨账户访问 S3 的 EMR Serverless Hive 作业运行。

```
aws emr-serverless start-job-run \
    --application-id application-id \
    --execution-role-arn job-role-arn \
    --job-driver '{
        "hive": {
            "query": "query_location",
            "parameters": "hive_parameters"
        }
    }' \
    --configuration-overrides '{
        "applicationConfiguration": [{
            "classification": "hive-site",
            "properties": {
```

```
                "fs.s3.customAWSCredentialsProvider":
 "com.amazonaws.emr.serverless.credentialsprovider.AssumeRoleAWSCredentialsProvider",
                "hive.emr-serverless.launch.env.ASSUME_ROLE_CREDENTIALS_ROLE_ARN":
 "arn:aws:iam::AccountB:role/Cross-Account-Role-B",
                "tez.am.emr-serverless.launch.env.ASSUME_ROLE_CREDENTIALS_ROLE_ARN":
 "arn:aws:iam::AccountB:role/Cross-Account-Role-B",
                "tez.task.emr-
serverless.launch.env.ASSUME_ROLE_CREDENTIALS_ROLE_ARN":
 "arn:aws:iam::AccountB:role/Cross-Account-Role-B"
            }
        }]
    }'
```

使用多个代入角色访问 S3 资源

在 EMR Serverless 6.11.0 及更高版本中，配置多个 IAM 角色，以便在访问不同的跨账户存储桶时代入。如果要使用账户 B 中的不同代入角色访问不同的 S3 资源，请在启动作业运行时使用以下配置：

1. 将 EMRFS 配置 `fs.s3.customAWSCredentialsProvider` 指定为 `com.amazonaws.emr.serverless.credentialsprovider.BucketLevelAssumeRoleCredenti`
2. 指定 EMRFS 配置 `fs.s3.bucketLevelAssumeRoleMapping`，定义从 S3 存储桶名称到账户 B 中要代入的 IAM 角色的映射。该值的格式为 `bucket1->role1;bucket2->role2`。

例如，使用 `arn:aws:iam::AccountB:role/Cross-Account-Role-B-1` 访问存储桶 `bucket1`，使用 `arn:aws:iam::AccountB:role/Cross-Account-Role-B-2` 访问存储桶 `bucket2`。以下示例展示了如何通过多个代入角色启动具有跨账户访问权限的 EMR Serverless 作业运行。

Spark

以下示例展示了如何使用多个代入角色来创建 EMR Serverless Spark 作业运行。

```
aws emr-serverless start-job-run \
    --application-id application-id \
    --execution-role-arn job-role-arn \
    --job-driver '{
        "sparkSubmit": {
            "entryPoint": "entrypoint_location",
            "entryPointArguments": [":argument_1:", ":argument_2:"],
```

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

Hive

以下示例展示了如何使用多个代入角色来创建 EMR Serverless Hive 作业运行。

```
aws emr-serverless start-job-run \
    --application-id application-id \
    --execution-role-arn job-role-arn \
    --job-driver '{
        "hive": {
            "query": "query_location",
            "parameters": "hive_parameters"
        }
    }' \
    --configuration-overrides '{
        "applicationConfiguration": [{
            "classification": "hive-site",
            "properties": {
                "fs.s3.customAWSCredentialsProvider":
 "com.amazonaws.emr.serverless.credentialsprovider.AssumeRoleAWSCredentialsProvider",
                "fs.s3.bucketLevelAssumeRoleMapping": "bucket1-
>arn:aws:iam::AccountB:role/Cross-Account-Role-B-1;bucket2-
>arn:aws:iam::AccountB:role/Cross-Account-Role-B-2"
            }
        }]
    }'
```

排查 EMR Serverless 中的错误

使用以下信息来帮助诊断和修复在使用 Amazon EMR Serverless 时发生的常见问题。

主题

错误：作业失败，因为账户已达到其可同时使用的最大 vCPU 的服务限制。

此错误表明 EMR Serverless 无法提交作业，因为账户已超出最大容量。增加账户的最大容量。在 EMR Serverless 服务配额查看您的服务限制。

错误：作业失败，因为应用程序超过 maximumCapacity 设置。

此错误表示 EMR Serverless 无法提交作业，因为应用程序已超出配置的最大容量。增加应用程序的最大容量。

错误：由于应用程序已超过 maximumCapacity，无法分配工作线程，作业失败。

此错误表明作业无法完成。由于应用程序已超过 maximumCapacity 设置，无法分配工作线程。

错误：S3 访问被拒绝。请检查作业运行时角色对所需 S3 资源的 S3 访问权限。

此错误表示您的作业无法访问 S3 资源。验证作业运行时角色是否有权访问作业需要使用的 S3 资源。要了解有关运行时角色的更多信息，请参阅 Amazon EMR Serverless 的作业运行时角色。

错误: ModuleNotFoundError: 未命名模块<module>。请参阅用户指南，了解如何将 Python 库与 EMR Serverless 结合使用。

此错误表明 Python 模块不可用于 Spark 作业。检查依赖的 Python 库是否可用于该作业。有关如何打包 Python 库的更多信息，请参阅将 Python 库与 EMR Serverless 结合使用。

错误：无法代入执行角色 <role-name>，因为该角色不存在或未设置所需的信任关系。

此错误表示您为作业指定的作业运行时角色不存在，或者该角色与 EMR Serverless 权限不存在信任关系。要验证 IAM 角色是否存在，并验证您是否已正确设置该角色的信任策略，请参阅 Amazon EMR Serverless 的作业运行时角色 中的说明。

启用 Job 级别成本分配

任务级别的成本分配允许在单个任务运行层面为 EMR Serverless 进行精细的计费归因，而不是在应用程序级别汇总所有成本。启用后，您可以在 Cost Explorer 和 “ AWS 成本和使用情况报告” 中按特定任务运行 IDs 和与任务运行关联的标签筛选和跟踪成本，从而更好地了解已提交的作业运行的费用。

默认 行为

默认情况下，未启用任务级别的成本分配。

如何启用或禁用该功能

您可以在创建应用程序期间配置任务级别的成本分配，也可以针对现有应用程序进行更新。

在创建新应用程序时指定`jobLevelCostAllocation`参数：

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

# Disable job-level cost allocation:
```

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

更新现有应用程序的jobLevelCostAllocationConfiguration参数：

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

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

注意事项和限制

- 启用任务级成本分配不会追溯归因在启用该功能之前完成的任务运行的成本。启用该功能后启动的 Job 运行将具有精细的成本归因。
- 只有当应用程序处于 “已创建” 或 “已停止” 状态时，才能更新任务级别的成本分配参数。
- 启用作业级别的成本分配后，成本将归因于单个作业运行而不是应用程序。要查看应用程序级别的汇总成本，您必须对该应用程序内运行的所有作业应用一致的标签（例如应用程序名称或应用程序 ID），并在 Cost Explorer 或 “成本和使用率报告” 中按这些标签进行筛选。

通过 EMR Studio 使用 EMR Serverless 运行交互式工作负载

借助 EMR Serverless 交互式应用程序，可使用 EMR Studio 中托管的 Notebook 通过 EMR Serverless 运行 Spark 交互式工作负载。

概述

交互式应用程序是启用了交互式功能的 EMR Serverless 应用程序。借助 Amazon EMR Serverless 交互式应用程序，您可以使用 Amazon EMR Studio 中管理的 Jupyter Notebook 执行交互式工作负载。这有助于数据工程师、数据科学家和数据分析师使用 EMR Studio 对 Amazon S3 和 Amazon DynamoDB 等数据存储中的数据集进行交互式分析。

EMR Serverless 中的交互式应用程序的用例包括：

- 数据工程师使用 EMR Studio 中的 IDE 体验来创建 ETL 脚本。该脚本从本地摄取数据，转换之后进行分析，并将数据存储在 Amazon S3 中。
- 数据科学家使用 Notebook 探索数据集，训练机器学习（ML）模型来检测数据集中的异常。
- 数据分析师探索数据集，创建脚本，生成日常报告，以更新业务控制面板等应用程序。

先决条件

要在 EMR Serverless 中使用交互式工作负载，应满足以下要求：

- Amazon EMR 6.14.0 及更高版本支持 EMR Serverless 交互式应用程序。
- 要访问交互式应用程序、执行提交的工作负载以及从 EMR Studio 运行交互式 Notebook，您需要特定的权限和角色。有关更多信息，请参阅交互式工作负载所需的权限。

交互式工作负载所需的权限

除了访问 EMR Serverless 所需的基本权限之外，请为您的 IAM 身份或角色配置其他权限：

访问交互式应用程序

为 EMR Studio 设置用户和 Workspace 权限。有关更多信息，请参阅《Amazon EMR 管理指南》中的配置 EMR Studio 用户权限。

执行您通过 EMR Serverless 提交的工作负载

设置作业运行时角色。有关更多信息，请参阅创建作业运行时角色。

从 EMR Studio 运行交互式 Notebook

将以下附加权限添加到 Studio 用户的 IAM 策略：

- **`emr-serverless:AccessInteractiveEndpoints`**：授予访问和连接您指定为 `Resource` 的交互式应用程序的权限。从 EMR Studio Workspace 附加到 EMR Serverless 应用程序需要此权限。
- **`iam:PassRole`**：授予访问您计划在附加到应用程序时使用的 IAM 执行角色的权限。从 EMR Studio Workspace 附加到 EMR Serverless 应用程序需要相应的 `PassRole` 权限。

JSON

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

```
  ]
}
```

配置交互式应用程序

使用以下高级步骤，从 AWS 管理控制台中的 Amazon EMR Studio 创建具有交互功能的 EMR Serverless 应用程序。

1. 按照 开始使用 Amazon EMR Serverless 中的步骤创建应用程序。
2. 然后，从 EMR Studio 启动一个 Workspace，将其作为计算选项附加到 EMR Serverless 应用程序。有关更多信息，请参阅 EMR Serverless 入门文档步骤 2 中的交互式工作负载选项卡。

将应用程序附加到 Studio Workspace 时，如果应用程序尚未运行，则会自动触发启动。您也可以预先启动应用程序，在将其附加到 Workspace 之前准备就绪。

交互式应用程序的注意事项

- Amazon EMR 6.14.0 及更高版本支持 EMR Serverless 交互式应用程序。
- EMR Studio 是唯一与 EMR Serverless 交互式应用程序集成的客户端。EMR Serverless 交互式应用程序不支持以下 EMR Studio 功能：Workspace 协作、SQL Explorer 和 Notebook 的编程执行。
- 只有 Spark 引擎支持交互式应用程序。
- 交互式应用程序支持 Python 3 PySpark 和 Spark Scala 内核。
- 您可以在单个交互式应用程序上运行多达 25 个并发 Notebook。
- 没有端点或 API 接口支持具有交互式应用程序的自托管 Jupyter Notebook。
- 为了获得最佳的启动体验，建议您为驱动程序和执行程序配置预初始化容量，并预先启动应用程序。预先启动应用程序时，确保其在附加到 Workspace 时已经准备就绪。

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

- 默认情况下，已为应用程序启用 `autoStopConfig`。这将在空闲 30 分钟后关闭应用程序。您可以将此配置作为 `create-application` 或 `update-application` 请求的一部分进行更改。
- 使用交互式应用程序时，建议您配置内核、驱动程序和执行程序的预初始化容量来运行笔记本。每个 Spark 交互式会话都需要一个内核和一个驱动程序，因此 EMR Serverless 会为每个预初始化的驱动

程序维护预初始化的内核工作线程。默认情况下，即使不为驱动程序指定任何预初始化容量，EMR Serverless 也会在整个应用程序中保留一个内核工作线程的预初始化容量。每个内核工作程序使用 4 个 vCPU 和 16GB 内存。有关当前定价信息，请参阅 Amazon EMR 定价页面。

- 您的中必须有足够的 vCPU 服务配额 AWS 账户 才能运行交互式工作负载。如果不运行支持 Lake Formation 的工作负载，建议至少使用 24 个 vCPU。如果这样做，我们建议至少使用 28 个 vCPU。
- 如果 Notebook 内核空闲时间超过 60 分钟，EMR Serverless 会自动终止内核。EMR Serverless 会根据 Notebook 会话期间完成的最后一项活动计算内核空闲时间。目前无法修改内核空闲超时设置。
- 要在交互式工作负载中启用 Lake Formation，请在创建 EMR Serverless 应用程序时将 `runtime-configuration` 对象中 `spark-defaults` 分类下的配置 `spark.emr-serverless.lakeformation.enabled` 设置为 `true`。要了解更多信息，请参阅 Enabling Lake Formation in Amazon EMR。

通过 Apache Livy 端点在 EMR Serverless 中运行交互式工作负载

在 Amazon EMR 6.14.0 及更高版本中，请在创建 EMR Serverless 应用程序时创建并启用 Apache Livy 端点，并通过自托管笔记本或自定义客户端运行交互式工作负载。Apache Livy 端点具有以下优势：

- 您可以通过 Jupyter Notebook 安全连接到 Apache Livy 端点，然后使用 Apache Livy 的 REST 接口管理 Apache Spark 工作负载。
- 对于使用 Apache Spark 工作负载数据的交互式 Web 应用程序，请使用 Apache Livy REST API 操作。

先决条件

要在 EMR Serverless 中使用 Apache Livy 端点，应满足以下要求：

- 完成开始使用 Amazon EMR Serverless 中的步骤。
- 要通过 Apache Livy 端点运行交互式工作负载，需要特定的权限和角色。有关更多信息，请参阅交互式工作负载所需的权限。

所需的权限

除了访问 EMR Serverless 所需的权限外，还请向 IAM 角色添加以下权限，以访问 Apache Livy 端点和运行应用程序：

- `emr-serverless:AccessLivyEndpoints`：授予访问和连接您指定为 `Resource` 并启用 Livy 的交互式应用程序的权限。您需要此权限才能运行 Apache Livy 端点提供的 REST API 操作。
- `iam:PassRole`：授予在创建 Apache Livy 会话时访问 IAM 执行角色的权限。EMR Serverless 将使用此角色来执行您的工作负载。
- `emr-serverless:GetDashboardForJobRun`：授予生成 Spark Live UI 和驱动程序日志链接的权限，并作为 Apache Livy 会话结果的一部分提供对日志的访问。

JSON

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

```
      "Action": [
        "emr-serverless:GetDashboardForJobRun"
      ],
      "Resource": [
        "arn:aws:emr-serverless:*:123456789012:/applications/*"
      ]
    }
  ]
}
```

开始使用

要创建并运行支持 Apache Livy 的应用程序，请按照以下步骤操作。

1. 要创建支持 Apache Livy 的应用程序，请运行以下命令。

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

2. 在 EMR Serverless 创建应用程序后，启动应用程序，使 Apache Livy 端点可用。

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

使用以下命令检查应用程序的状态。状态变为 STARTED 之后，请访问 Apache Livy 端点。

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

3. 使用以下 URL 访问端点：

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

端点准备就绪后，请根据使用案例提交工作负载。您必须使用 SIGv4 协议对发往端点的每个请求进行签名，并传入授权标头。您可以使用以下方法运行工作负载：

- HTTP 客户端：使用自定义 HTTP 客户端提交 Apache Livy 端点 API 操作。
- Sparkmagic 内核：在本地运行 sparkmagic 内核，使用 Jupyter Notebook 提交交互式查询。

HTTP 客户端

要创建 Apache Livy 会话，请在请求正文的 `conf` 参数中提交 `emr-serverless.session.executionRoleArn`。下面是一个 `POST /sessions` 请求示例。

```
{
    "kind": "pyspark",
    "heartbeatTimeoutInSecond": 60,
    "conf": {
        "emr-serverless.session.executionRoleArn": "<executionRoleArn>"
    }
}
```

下表列出了所有可用的 Apache Livy API 操作。

API 操作	说明
GET /sessions	返回所有活动交互式会话的列表。
POST /sessions	通过 Spark 或 pyspark 创建新的交互式会话。
GET /sessions/ < > *sessionId*	返回会话信息。
GET /sessions/ < >/state *sessionId*	返回会话状态。
删除 /sessions/ < > *sessionId*	停止并删除会话。
GET /sessions/ < >/语句 *sessionId*	返回会话中的所有语句。
POST /sessions/ < >/声明 *sessionId*	在会话中运行语句。
GET /sessions/ < >/statements/< > *sessionId* *statementId*	返回会话中指定语句的详细信息。
POST /sessions/ < >/statements/< >/取消 *sessionId* *statementId*	取消会话中的指定语句。

向 Apache Livy 端点发送请求

您也可以从 HTTP 客户端直接向 Apache Livy 端点发送请求。这样，您就可以在 Notebook 之外远程运行用例的代码。

在开始向端点发送请求之前，确保已安装以下库：

```
pip3 install botocore awscrt requests
```

下面是一个直接向端点发送 HTTP 请求的 Python 脚本示例：

```
from botocore import crt
import requests
from botocore.awsrequest import AWSRequest
from botocore.credentials import Credentials
import botocore.session
import json, pprint, textwrap

endpoint = 'https://<application_id>.livy.emr-serverless-
services.<AWS_REGION>.amazonaws.com'
headers = {'Content-Type': 'application/json'}

session = botocore.session.Session()
signer = crt.auth.CrtS3SigV4Auth(session.get_credentials(), 'emr-serverless',
 '<AWS_REGION>')


### Create session request

data = {'kind': 'pyspark', 'heartbeatTimeoutInSecond': 60, 'conf': { 'emr-
serverless.session.executionRoleArn': 'arn:aws:iam::123456789012:role/role1'}}

request = AWSRequest(method='POST', url=endpoint + "/sessions", data=json.dumps(data),
 headers=headers)

request.context["payload_signing_enabled"] = False

signer.add_auth(request)

prepped = request.prepare()

r = requests.post(prepped.url, headers=prepped.headers, data=json.dumps(data))
```

```
pprint.pprint(r.json())


### List Sessions Request

request = AWSRequest(method='GET', url=endpoint + "/sessions", headers=headers)

request.context["payload_signing_enabled"] = False

signer.add_auth(request)

prepped = request.prepare()

r2 = requests.get(prepped.url, headers=prepped.headers)
pprint.pprint(r2.json())


### Get session state

session_url = endpoint + r.headers['location']

request = AWSRequest(method='GET', url=session_url, headers=headers)

request.context["payload_signing_enabled"] = False

signer.add_auth(request)

prepped = request.prepare()

r3 = requests.get(prepped.url, headers=prepped.headers)

pprint.pprint(r3.json())


### Submit Statement

data = {
      'code': "1 + 1"
}

statements_url = endpoint + r.headers['location'] + "/statements"

request = AWSRequest(method='POST', url=statements_url, data=json.dumps(data),
 headers=headers)
```

```
request.context["payload_signing_enabled"] = False

signer.add_auth(request)

prepped = request.prepare()

r4 = requests.post(prepped.url, headers=prepped.headers, data=json.dumps(data))

pprint.pprint(r4.json())

### Check statements results

specific_statement_url = endpoint + r4.headers['location']

request = AWSRequest(method='GET', url=specific_statement_url, headers=headers)

request.context["payload_signing_enabled"] = False

signer.add_auth(request)

prepped = request.prepare()

r5 = requests.get(prepped.url, headers=prepped.headers)

pprint.pprint(r5.json())


### Delete session


session_url = endpoint + r.headers['location']

request = AWSRequest(method='DELETE', url=session_url, headers=headers)

request.context["payload_signing_enabled"] = False

signer.add_auth(request)

prepped = request.prepare()

r6 = requests.delete(prepped.url, headers=prepped.headers)
```

```
pprint.pprint(r6.json())
```

Sparkmagic 内核

在安装 sparkmagic 之前，请确保已在要安装 sparkmagic 的实例中配置了 AWS 凭据

1. 按照安装步骤安装 Sparkmagic。请注意，您只需执行前四个步骤。
2. sparkmagic 内核支持自定义身份验证器，因此您可以将身份验证器与 sparkmagic 内核集成，以便对每个请求进行签名。 SIGv4
3. 安装 EMR Serverless 自定义身份验证器。

   ```
   pip install emr-serverless-customauth
   ```
4. 现在，在 Sparkmagic 配置 json 文件中提供自定义身份验证器的路径和 Apache Livy 端点 URL。使用以下命令打开配置文件。

   ```
   vim ~/.sparkmagic/config.json
   ```

   以下为示例 `config.json` 文件。

   ```
   {
   "kernel_python_credentials" : {
       "username": "",
       "password": "",
       "url": "https://<application-id>.livy.emr-serverless-
   services.<AWS_REGION>.amazonaws.com",
       "auth": "Custom_Auth"
     },

     "kernel_scala_credentials" : {
       "username": "",
       "password": "",
       "url": "https://<application-id>.livy.emr-serverless-
   services.<AWS_REGION>.amazonaws.com",
       "auth": "Custom_Auth"
     },
     "authenticators": {
       "None": "sparkmagic.auth.customauth.Authenticator",
       "Basic_Access": "sparkmagic.auth.basic.Basic",
       "Custom_Auth":
    "emr_serverless_customauth.customauthenticator.EMRServerlessCustomSigV4Signer"
   ```

```
  },
  "livy_session_startup_timeout_seconds": 600,
  "ignore_ssl_errors": false
}
```

5. 启动 Jupyter 实验室。应使用您在上一步中设置的自定义身份验证。
6. 然后，您可以运行以下 Notebook 命令和代码开始使用。

```
%%info //Returns the information about the current sessions.
```

```
%%configure -f //Configure information specific to a session. We supply
 executionRoleArn in this example. Change it for your use case.
{
    "driverMemory": "4g",
    "conf": {
          "emr-serverless.session.executionRoleArn":
 "arn:aws:iam::123456789012:role/JobExecutionRole"
    }
}
```

```
<your code>//Run your code to start the session
```

在内部，每条指令都通过配置的 Apache Livy 端点 URL 调用每个 Apache Livy API 操作。然后，您可以根据自己的用例编写指令。

注意事项

通过 Apache Livy 端点运行交互式工作负载时，请考虑以下注意事项。

- EMR Serverless 使用调用方主体维护会话级隔离。创建会话的调用方主体是唯一可以访问该会话的主体。要进行更精细的隔离，请在代入凭证时配置源身份。在这种情况下，EMR Serverless 会基于调用方主体和源身份强制执行会话级隔离。有关源身份的更多信息，请参阅监控和控制使用代入角色执行的操作。
- EMR Serverless 6.14.0 及更高版本支持 Apache Livy 端点。
- 只有 Apache Spark 引擎支持 Apache Livy 端点。
- Apache Livy 端点支持 Scala Spark 和。 PySpark

- 默认情况下，应用程序中已启用 autoStopConfig。这意味着应用程序将在空闲 15 分钟后关闭。您可以将此配置作为 create-application 或 update-application 请求的一部分进行更改。
- 在一个支持 Apache Livy 端点的应用程序上，最多可以运行 25 个并发会话。
- 为获得最佳启动体验，建议您为驱动程序和执行程序配置预初始化容量。
- 在连接到 Apache Livy 端点之前，必须手动启动应用程序。
- 您必须有足够的 vCPU 服务配额才能使用 Apache Liv AWS 账户 y 终端节点运行交互式工作负载。我们建议至少使用 24 个 vCPU。
- 默认的 Apache Livy 会话超时为 1 小时。如果一小时没有运行语句，Apache Livy 将删除会话并释放驱动程序和执行程序。从 emr-7.8.0 版本开始，可以通过在 Livy /sessions POST 请求中指定 ttl 参数来设置此值，例如 2h（小时）、120m（分钟）、7200s（秒）、7200000ms（毫秒）。

> **Note**
>
> 在 emr-7.8.0 版本之前，此配置无法更改。下面是一个 POST /sessions 请求正文示例。

```
{
    "kind": "pyspark",
    "heartbeatTimeoutInSecond": 60,
    "conf": {
        "emr-serverless.session.executionRoleArn": "executionRoleArn"
    },
    "ttl": "2h"
}
```

- 从 Amazon EMR 版本 emr-7.8.0 开始，适用于通过 LakeFormation 启用精细访问控制的应用程序，可以按会话禁用该设置。有关为 EMR Serverless 应用程序启用精细访问控制的更多信息，请参阅 Methods for fine-grained access control。

> **Note**
>
> 如果没有为应用程序启用 Lake Formation，则无法为会话启用它。下面是一个 POST /sessions 请求正文示例。

```
{
    "kind": "pyspark",
```

```
    "heartbeatTimeoutInSecond": 60,
    "conf": {
        "emr-serverless.session.executionRoleArn": "executionRoleArn"
    },
    "spark.emr-serverless.lakeformation.enabled" : "false"
}
```

- 只有活动会话才能与 Apache Livy 端点交互。一旦会话结束、取消或终止，就无法通过 Apache Livy 端点访问会话。

日志记录和监控

监控是维护 EMR Serverless 应用程序和作业的可靠性、可用性和性能的一个重要环节。您应该收集 EMR Serverless 解决方案所有部分的监控数据，以便在发生多点故障时更轻松地进行调试。

主题

存储日志

要在 EMR Serverless 上监控作业进度并排除作业故障，请选择 EMR Serverless 存储和提供应用程序日志的方式。提交任务运行时，请将托管存储、Amazon S3 和 Amazon 指定 CloudWatch 为日志选项。

使用 CloudWatch，指定要使用的日志类型和日志位置，或者接受默认的类型和位置。有关 CloudWatch 日志的更多信息，请参阅the section called “Amazon CloudWatch”。对于托管存储和 S3 日志记录，下表列出了选择托管存储、Amazon S3 存储桶或两者时预期的日志位置和 UI 可用性。

Option	事件日志	容器日志	应用程序 UI
托管存储	存储在托管存储中	存储在托管存储中	支持
托管存储和 S3 存储桶	存放在两个位置	存储在 S3 存储桶中	支持
亚马逊 S3 存储桶	存储在 S3 存储桶中	存储在 S3 存储桶中	不支持[1]

[1] 建议您保持选中托管存储。否则，您将无法使用内置应用程序 UIs。

使用托管存储的 EMR Serverless 日志记录

默认情况下，EMR Serverless 将应用程序日志安全存储在 Amazon EMR 托管存储中，最长 30 天。

> **Note**
>
> 如果关闭默认选项，Amazon EMR 将无法代表您对作业进行故障排除。示例：您无法从 EMR Serverless 控制台访问 Spark-UI。

要从 EMR Studio 关闭此选项，请取消选中 “提交作业” 页面的 “其他设置” 部分中的 “允许 AWS 将日志保留 30 天” 复选框。

要从中关闭此选项 AWS CLI，请在提交作业运行时使用该`managedPersistenceMonitoringConfiguration`配置。

```
{
    "monitoringConfiguration": {
        "managedPersistenceMonitoringConfiguration": {
            "enabled": false
        }
    }
}
```

如果您的 EMR Serverless 应用程序位于具有适用于 Amazon S3 的 VPC 端点的私有子网中，并且您附加了端点策略来控制访问，请为 EMR Serverless 添加以下权限以存储和提供应用程序日志。请将 `Resource` 替换为 Sample policies for private subnets that access Amazon S3 中可用区域表中的 `AppInfo` 存储桶。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "EMRServerlessManagedLogging",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:PutObjectAcl"
      ],
      "Resource": [
        "arn:aws:s3:::prod.us-east-1.appinfo.src",
        "arn:aws:s3:::prod.us-east-1.appinfo.src/*"
```

```
      ],
      "Condition": {
        "StringEquals": {
          "aws:PrincipalServiceName": "emr-serverless.amazonaws.com",
          "aws:SourceVpc": "vpc-12345678"
        }
      }
    }
  ]
}
```

此外，使用 aws:SourceVpc 条件键确保请求通过 VPC 端点所附加的 VPC 进行传输。

使用 Amazon S3 存储桶的 EMR Serverless 日志记录

在作业将日志数据发送到 Amazon S3 之前，请在作业运行时角色的权限策略中包含以下权限。将 *amzn-s3-demo-logging-bucket* 替换为日志记录存储桶的名称。

JSON

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

要设置 Amazon S3 存储桶来存储来自的日志 AWS CLI，请在开始运行任务时使用s3MonitoringConfiguration配置。为此，请在配置中提供以下 --configuration-overrides。

```
{
    "monitoringConfiguration": {
        "s3MonitoringConfiguration": {
            "logUri": "s3://amzn-s3-demo-logging-bucket/logs/"
        }
    }
}
```

对于未启用重试的批处理作业，EMR Serverless 会将日志发送到以下路径：

```
'/applications/<applicationId>/jobs/<jobId>'
```

EMR Serverless 将 Spark 驱动程序日志存储在以下路径中

```
'/applications/<applicationId>/jobs/<jobId>/SPARK_DRIVER/'
```

EMR Serverless 将 Spark 执行程序日志存储在以下路径中

```
'/applications/<applicationId>/jobs/<jobId>/SPARK_EXECUTOR/<EXECUTOR-ID>'
```

<EXECUTOR-ID> 是一个整数。

EMR Serverless 7.1.0 及更高版本支持流处理作业和批处理作业重试。如果您在启用重试的情况下运行作业，EMR Serverless 会自动在日志路径前缀中添加重试编号，以便更好地区分和跟踪日志。

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

使用 Amazon 登录 EMR 无服务器 CloudWatch

当您向 EMR Serverless 应用程序提交任务时，请选择 Amazon CloudWatch 作为存储应用程序日志的选项。这允许您使用 CloudWatch 日志分析功能，例如 Logs Insights 和 Live Tail。 CloudWatch 您也可以将日志从其他系统流式传输 CloudWatch 到其他系统，例如 OpenSearch 进行进一步分析。

EMR Serverless 可实时记录驱动程序日志。您可以使用实时跟踪功能或通过 CloudWatch CLI tail 命令 CloudWatch 实时访问日志。

默认情况下，EMR Ser CloudWatch verless 的日志记录处于禁用状态。若要启用，请使用 AWS CLI 中的配置。

> **Note**
>
> Amazon 实时 CloudWatch 发布日志，因此它会从工作人员那里获得更多资源。如果您选择较小的工作线程容量，可能会增加作业运行延迟。如果您启用 CloudWatch 日志记录，我们建议您选择更大的工作人员容量。如果每秒事务数（TPS）速率对于 `PutLogEvents` 来说太低，日志发布也可能会受到限制。 CloudWatch 限制配置适用于所有服务，包括 EMR Serverless。有关更多信息，请参阅如何确定日志中的限制？ CloudWatch 在 re AWS : post 上。

使用登录所需的权限 CloudWatch

在您的任务可以向 Amazon 发送日志数据之前 CloudWatch，请在任务运行时角色的权限策略中包含以下权限。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogGroups"
      ],
      "Resource": [
        "arn:aws:logs:*:123456789012:*"
      ],
      "Sid": "AllowLOGSDescribeloggroups"
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:PutLogEvents",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:DescribeLogStreams"
      ],
      "Resource": [
        "arn:aws:logs:*:123456789012:log-group:my-log-group-name:*"
      ],
      "Sid": "AllowLOGSPutlogevents"
```

```
    }
  ]
}
```

AWS CLI

要将 Amazon 设置为存储 CloudWatch 来自 EMR Serverless 的日志 AWS CLI，请在开始运行任务时使用`cloudWatchLoggingConfiguration`配置。为此，请提供以下配置覆盖。或者，还请提供日志组名称、日志流前缀名称、日志类型和加密密钥 ARN。

如果您未指定可选值，则使用默认日志流将日志 CloudWatch 发布到默认日志组`/aws/emr-serverless/applications/`*`applicationId`*`/jobs/`*`jobId`*`/`*`worker-type`*。

EMR Serverless 7.1.0 及更高版本支持流处理作业和批处理作业重试。如果作业启用了重试，EMR Serverless 会自动在日志路径前缀中添加重试编号，以便更好地区分和跟踪日志。

```
'/applications/<applicationId>/jobs/<jobId>/attempts/<attemptNumber>/worker-type'
```

以下内容演示了使用 EMR Serverless 的默认设置开启 Amazon CloudWatch 日志记录所需的最低配置：

```
{
    "monitoringConfiguration": {
        "cloudWatchLoggingConfiguration": {
            "enabled": true
         }
     }
}
```

以下示例显示了所有必需和可选配置，这些配置指定何时为 EMR Serverless 启用 Amazon CloudWatch 日志功能。示例下方还列出了支持的 `logTypes` 值。

```
{
    "monitoringConfiguration": {
        "cloudWatchLoggingConfiguration": {
            "enabled": true, // Required
            "logGroupName": "Example_logGroup", // Optional
            "logStreamNamePrefix": "Example_logStream", // Optional
            "encryptionKeyArn": "key-arn", // Optional
            "logTypes": {
```

```
                "SPARK_DRIVER": ["stdout", "stderr"] //List of values
            }
        }
    }
}
```

默认情况下，EMR Serverless 仅向发布驱动程序 stdout 和 stderr 日志。 CloudWatch如果需要其他日志，请在 `logTypes` 字段中指定容器角色和相应的日志类型。

以下列表显示了为 `logTypes` 配置指定的受支持工作线程类型：

Spark

- `SPARK_DRIVER : ["STDERR", "STDOUT"]`
- `SPARK_EXECUTOR : ["STDERR", "STDOUT"]`

Hive

- `HIVE_DRIVER : ["STDERR", "STDOUT", "HIVE_LOG", "TEZ_AM"]`
- `TEZ_TASK : ["STDERR", "STDOUT", "SYSTEM_LOGS"]`

轮换日志

Amazon EMR Serverless 可以轮换 Spark 应用程序日志和事件日志。日志轮换有助于解决长时间运行的作业生成大型日志文件，而占用磁盘空间的问题。轮换日志有助于节省磁盘存储空间，减少因磁盘空间不足而导致的作业失败次数。

默认情况下，日志轮换处于启用状态，且仅适用于 Spark 作业。

Spark 事件日志

> **Note**
>
> Spark 事件日志轮换适用于所有 Amazon EMR 发行版标签。

EMR Serverless 不是生成单个事件日志文件，而是定期轮换事件日志并删除旧的事件日志文件。轮换日志不会影响上传到 S3 存储桶的日志。

Spark 应用程序日志

> **Note**
>
> Spark 应用程序日志轮换适用于所有 Amazon EMR 发行版标签。

EMR Serverless 还会轮换驱动程序和执行程序的 Spark 应用程序日志，例如 `stdout` 和 `stderr`。您可以使用 Spark History Server 和 Live UI 链接在 Studio 中选择日志链接来访问最新的日志文件。日志文件是最新日志的截断版本。要查看旧的轮换日志，请在存储日志时指定 Amazon S3 位置。有关更多信息，请参阅使用 Amazon S3 存储桶的 EMR Serverless 日志记录。

您可以在以下位置找到最新的日志文件。EMR Serverless 每 15 秒刷新一次文件。这些文件的大小从 0MB 到 128MB 不等。

```
<example-S3-logUri>/applications/<application-id>/jobs/<job-id>/SPARK_DRIVER/stderr.gz
```

以下位置包含旧的轮换文件。每个文件都是 128MB。

```
<example-S3-logUri>/applications/<application-id>/jobs/<job-id>/SPARK_DRIVER/archived/
stderr_<index>.gz
```

同样的行为也适用于 Spark 执行程序。此更改仅适用于 S3 日志记录。日志轮换不会对上传到 Amazon 的日志流进行任何更改 CloudWatch。

EMR Serverless 7.1.0 及更高版本支持流处理和批处理作业重试。如果您在作业中启用了重试，EMR Serverless 会在此类作业的日志路径中添加前缀，以便更好地跟踪和区分日志。此路径包含所有轮换日志。

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

加密日志

使用托管存储加密 EMR Serverless 日志

要使用您自己的 KMS 密钥加密托管存储中的日志，请在提交作业运行时使用 `managedPersistenceMonitoringConfiguration` 配置。

```
{
```

```
    "monitoringConfiguration": {
        "managedPersistenceMonitoringConfiguration" : {
            "encryptionKeyArn": "key-arn"
        }
    }
}
```

使用 Amazon S3 存储桶加密 EMR Serverless 日志

要使用您自己的 KMS 密钥加密 Amazon S3 存储桶中的日志，请在提交作业运行时使用 `s3MonitoringConfiguration` 配置。

```
{
    "monitoringConfiguration": {
        "s3MonitoringConfiguration": {
            "logUri": "s3://amzn-s3-demo-logging-bucket/logs/",
            "encryptionKeyArn": "key-arn"
        }
    }
}
```

使用 Amazon 加密 EMR 无服务器日志 CloudWatch

要使用您自己的 KMS 密钥加密 Amazon CloudWatch 中的日志，请在提交任务运行时使用该`cloudWatchLoggingConfiguration`配置。

```
{
    "monitoringConfiguration": {
        "cloudWatchLoggingConfiguration": {
            "enabled": true,
            "encryptionKeyArn": "key-arn"
         }
     }
}
```

日志加密所需的权限

本节内容

- 所需的用户权限

- Amazon S3 和托管存储的加密密钥权限
- Amazon 的加密密钥权限 CloudWatch

所需的用户权限

提交作业、查看日志或应用程序的用户 UIs 必须具有使用密钥的权限。您可以在 KMS 密钥政策或 IAM 策略中为用户、组或角色指定权限。如果提交作业的用户没有 KMS 密钥权限，EMR Serverless 会拒绝提交作业运行。

示例密钥政策

以下密钥政策提供了对 `kms:GenerateDataKey` 和 `kms:Decrypt` 的权限：

```
{
    "Effect": "Allow",
    "Principal":{
       "AWS": "arn:aws:iam::111122223333:user/user-name"
     },
     "Action": [
       "kms:GenerateDataKey",
       "kms:Decrypt"
      ],
     "Resource": "*"
 }
```

示例 IAM 策略

以下 IAM 策略提供了对 `kms:GenerateDataKey` 和 `kms:Decrypt` 的权限：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ],
```

```
      "Resource": [
        "arn:aws:kms:*:123456789012:key/12345678-1234-1234-1234-123456789012"
      ],
      "Sid": "AllowKMSGeneratedatakey"
    }
  ]
}
```

要启动 Spark 或 Tez UI，请授予用户、组或角色访问 `emr-serverless:GetDashboardForJobRun` API 的权限，如下所示：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-serverless:GetDashboardForJobRun"
      ],
      "Resource": [
        "*"
      ],
      "Sid": "AllowEMRSERVERLESSGetdashboardforjobrun"
    }
  ]
}
```

Amazon S3 和托管存储的加密密钥权限

在托管存储或 S3 存储桶中使用自己的加密密钥对日志进行加密时，请按如下方式配置 KMS 密钥权限。

在 KMS 密钥政策中，`emr-serverless.amazonaws.com` 主体必须具有以下权限：

```
{
    "Effect": "Allow",
    "Principal":{
       "Service": "emr-serverless.amazonaws.com"
```

```
      },
      "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
       ],
      "Resource": "*"
      "Condition": {
        "StringLike": {
          "aws:SourceArn": "arn:aws:emr-serverless:region:aws-account-id:/
applications/application-id"
        }
      }
 }
```

作为安全最佳实践，建议在 KMS 密钥策略中添加 `aws:SourceArn` 条件键。IAM 全局条件键 `aws:SourceArn` 可确保 EMR Serverless 仅将 KMS 密钥用于应用程序 ARN。

作业运行时角色必须在其 IAM 策略中具有以下权限：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ],
      "Resource": [
        "arn:aws:kms:*:123456789012:key/12345678-1234-1234-1234-123456789012"
      ],
      "Sid": "AllowKMSGeneratedatakey"
    }
  ]
}
```

Amazon 的加密密钥权限 CloudWatch

要将 KMS 密钥 ARN 关联到日志组，请对作业运行时角色使用以下 IAM 策略。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:AssociateKmsKey"
      ],
      "Resource": [
        "arn:aws:logs:*:123456789012:log-group:my-log-group-name:*"
      ],
      "Sid": "AllowLOGSAssociatekmskey"
    }
  ]
}
```

配置 KMS 密钥策略以向 Amazon 授予 KMS 权限 CloudWatch：

JSON

```
{
  "Version":"2012-10-17",
  "Id": "key-default-1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "ArnLike": {
          "kms:EncryptionContext:aws:logs:arn": "arn:aws:logs:*:123456789012:*"
        }
      },
      "Sid": "AllowKMSDecrypt"
```

```
    }
  ]
}
```

为 Amazon EMR Serverless 配置 Apache Log4j2 属性

本页面介绍了如何在 `StartJobRun` 中为 EMR Serverless 作业配置自定义 Apache Log4j 2.x 属性。如果要在应用程序级别配置 Log4j 分类，请参阅 EMR Serverless 的默认应用程序配置。

为 Amazon EMR Serverless 配置 Spark Log4j2 属性

在 Amazon EMR 6.8.0 及更高版本中，您可以自定义 Apache Log4j 2.x 属性，以指定细粒度的日志配置。这简化了 EMR Serverless 上 Spark 作业的故障排除。要配置这些属性，请使用 `spark-driver-log4j2` 和 `spark-executor-log4j2` 分类。

主题

Spark 的 Log4j2 分类

要自定义 Spark 日志配置，请在 `applicationConfiguration` 中使用以下分类。要配置 Log4j 2.x 属性，请使用以下 `properties`。

`spark-driver-log4j2`

该分类为驱动程序设置 `log4j2.properties` 文件中的值。

`spark-executor-log4j2`

该分类为执行程序设置 `log4j2.properties` 文件中的值。

Spark 的 Log4j2 配置示例

以下示例展示了如何向 `applicationConfiguration` 提交 Spark 作业，以便为 Spark 驱动程序和执行程序自定义 Log4j2 配置。

要在应用程序级别而不是在提交作业时配置 Log4j 分类，请参阅 EMR Serverless 的默认应用程序配置。

```
aws emr-serverless start-job-run \
    --application-id application-id \
    --execution-role-arn job-role-arn \
    --job-driver '{
        "sparkSubmit": {
            "entryPoint": "/usr/lib/spark/examples/jars/spark-examples.jar",
            "entryPointArguments": ["1"],
            "sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi --conf
 spark.executor.cores=4 --conf spark.executor.memory=20g --conf spark.driver.cores=4 --
conf spark.driver.memory=8g --conf spark.executor.instances=1"
        }
    }'
    --configuration-overrides '{
        "applicationConfiguration": [
             {
                "classification": "spark-driver-log4j2",
                "properties": {
                    "rootLogger.level":"error", // will only display Spark error logs
                    "logger.IdentifierForClass.name": "classpath for setting logger",
                    "logger.IdentifierForClass.level": "info"

                }
            },
            {
                "classification": "spark-executor-log4j2",
                "properties": {
                    "rootLogger.level":"error", // will only display Spark error logs
                    "logger.IdentifierForClass.name": "classpath for setting logger",
                    "logger.IdentifierForClass.level": "info"
                }
            }
       ]
    }'
```

示例 Spark 作业中的 Log4j2

以下代码示例演示了如何在初始化应用程序的自定义 Log4j2 配置时创建 Spark 应用程序。

Python

Example使用 Log4j2 通过 Python 执行 Spark 作业

```
import os
import sys

from pyspark import SparkConf, SparkContext
from pyspark.sql import SparkSession

app_name = "PySparkApp"
if __name__ == "__main__":
    spark = SparkSession\
        .builder\
        .appName(app_name)\
        .getOrCreate()

    spark.sparkContext._conf.getAll()
    sc = spark.sparkContext
    log4jLogger = sc._jvm.org.apache.log4j
    LOGGER = log4jLogger.LogManager.getLogger(app_name)

    LOGGER.info("pyspark script logger info")
    LOGGER.warn("pyspark script logger warn")
    LOGGER.error("pyspark script logger error")

    // your code here

    spark.stop()
```

要在执行 Spark 作业时为驱动程序自定义 Log4j2，请使用以下配置：

```
{
   "classification": "spark-driver-log4j2",
      "properties": {
          "rootLogger.level":"error", // only display Spark error logs
          "logger.PySparkApp.level": "info",
          "logger.PySparkApp.name": "PySparkApp"
      }
}
```

Scala

Example使用 Log4j2 通过 Scala 执行 Spark 作业

```
import org.apache.log4j.Logger
import org.apache.spark.sql.SparkSession

object ExampleClass {
  def main(args: Array[String]): Unit = {
    val spark = SparkSession
    .builder
    .appName(this.getClass.getName)
    .getOrCreate()

    val logger = Logger.getLogger(this.getClass);
    logger.info("script logging info logs")
    logger.warn("script logging warn logs")
    logger.error("script logging error logs")

// your code here
    spark.stop()
  }
}
```

要在执行 Spark 作业时为驱动程序自定义 Log4j2，请使用以下配置：

```
{
   "classification": "spark-driver-log4j2",
      "properties": {
          "rootLogger.level":"error", // only display Spark error logs
          "logger.ExampleClass.level": "info",
          "logger.ExampleClass.name": "ExampleClass"
      }
}
```

Spark 的 Log4j2 注意事项

以下 Log4j2.x 属性不可针对 Spark 进程进行配置：

- `rootLogger.appenderRef.stdout.ref`
- `appender.console.type`

- `appender.console.name`
- `appender.console.target`
- `appender.console.layout.type`
- `appender.console.layout.pattern`

有关配置的 Log4j2.x 属性的详细信息，请参阅上的文件。`log4j2.properties.template` GitHub

监控 EMR Serverless

本节介绍监控 Amazon EMR Serverless 应用程序和作业的方法。

主题

- 监控 EMR Serverless 应用程序和作业
- 使用 Amazon Managed Service for Prometheus 监控 Spark 指标
- EMR Serverless 用量指标

监控 EMR Serverless 应用程序和作业

借助 EMR Serverless 的 Amazon CloudWatch 指标，您可以接收 1 分钟的 CloudWatch 指标并访问 CloudWatch 控制面板，以访问 EMR 无服务器应用程序的 near-real-time操作和性能。

EMR Serverless 每分钟发送一次指标。 CloudWatch EMR Serverless 在应用程序级别以及作业、工作人员类型和级别发布这些指标。 capacity-allocation-type

要开始使用 EMR 无服务器存储库中提供的 EMR 无服务器 CloudWatch 仪表板模板并进行部署。 GitHub

> **Note**
>
> EMR Serverless 交互式工作负载仅启用了应用程序级别监控，并具有新的工作线程类型维度 `Spark_Kernel`。要监控和调试交互式工作负载，请在 EMR Studio Workspace 中访问日志和 Apache Spark UI。

监控指标

> **Important**
>
> 我们正在重组指标显示以添加 ApplicationName 和 JobName 作为维度。对于 7.10 及更高版本，旧指标将不再更新。对于 7.10 以下的 EMR 发行版，旧的指标仍然可用。

当前维度

下表描述了 AWS/EMR Serverless 命名空间中可用的 EMR Serverless 维度。

EMR Serverless 指标的维度

维度	说明	
ApplicationId	使用应用程序 ID 筛选 EMR Serverless 应用程序的所有指标。	
ApplicationName	使用名称筛选 EMR Serverless 应用程序的所有指标。如果未提供名称或包含非 ASCII 字符，则系统会将其发布为 [未指定]。	
JobId	筛选 EMR Serverless 作业运行 ID 的所有指标。	
JobName	使用名称筛选 EMR Serverless 作业运行的所有指标。如果未提供名称或包含非 ASCII 字符，则系统会将其发布为 [未指定]。	
WorkerType	筛选给定工作线程类型的所有指标。例如，您可以筛选 Spark 作业的 SPARK_DRI	

维度	说明	
	VER 和 SPARK_EXECUTORS 。	
CapacityAllocationType	筛选给定容量分配类型的所有指标。例如，您可以筛选预初始化容量 PreInitCapacity 和其他容量 OnDemandCapacity 。	

应用程序级别监控

您可以使用 Amazon 指标在 EMR 无服务器应用程序级别监控容量使用情况。 CloudWatch 您还可以设置单个显示器来监控 CloudWatch 仪表板中的应用程序容量使用情况。

EMR Serverless 应用程序指标

指标	说明	单位	维度
MaxCPUAllowed	应用程序允许的最大 CPU 数。	vCPU	ApplicationId ，ApplicationName
MaxMemoryAllowed	应用程序允许的最大内存（GB）。	千兆字节（GB）	ApplicationId ，ApplicationName
MaxStorageAllowed	应用程序允许的最大存储空间（GB）。	千兆字节（GB）	ApplicationId ，ApplicationName
CPUAllocated	CPUs 分配的 v 的总数。	vCPU	ApplicationId ，ApplicationName ，WorkerType ，CapacityAllocationType
IdleWorkerCount	空闲的工作线程总数。	计数	ApplicationId ，ApplicationName ，WorkerTyp

指标	说明	单位	维度
			e ，CapacityA llocationType
MemoryAllocated	分配的总内存（GB ）。	千兆字节（GB ）	ApplicationId ， ApplicationName ， WorkerTyp e ，CapacityA llocationType
PendingCr eationWor kerCount	待创建的工作线程总数。	计数	ApplicationId ， ApplicationName ， WorkerTyp e ，CapacityA llocationType
RunningWo rkerCount	应用程序使用的工作线程总数。	计数	ApplicationId ， ApplicationName ， WorkerTyp e ，CapacityA llocationType
StorageAl located	分配的总磁盘存储空间（GB ）。	千兆字节（GB ）	ApplicationId ， ApplicationName ， WorkerTyp e ，CapacityA llocationType
TotalWork erCount	可用的工作线程总数。	计数	ApplicationId ， ApplicationName ， WorkerTyp e ，CapacityA llocationType

作业级别监控

Amazon EMR Serverless 每隔一分钟向 Amazon CloudWatch 发送以下作业级别指标。您可以按作业运行状态访问聚合作业运行的指标值。每个指标的单位是计数。

EMR Serverless 作业级别指标

指标	说明	维度
SubmittedJobs	处于“已提交”状态的作业数量。	ApplicationId，ApplicationName
PendingJobs	处于“待定”状态的作业数量。	ApplicationId，ApplicationName
ScheduledJobs	处于“已计划”状态的作业数量。	ApplicationId，ApplicationName
RunningJobs	处于“运行中”状态的作业数量。	ApplicationId，ApplicationName
SuccessJobs	处于“成功”状态的作业数量。	ApplicationId，ApplicationName
FailedJobs	处于“失败”状态的作业数量。	ApplicationId，ApplicationName
CancellingJobs	处于“取消中”状态的作业数量。	ApplicationId，ApplicationName
CancelledJobs	处于“已取消”状态的作业数量。	ApplicationId，ApplicationName

您可以使用特定于引擎的应用程序监控特定于引擎的指标，以了解正在运行和已完成的 EMR Serverless 作业。 UIs当您访问运行中作业的 UI 时，系统会显示具有实时更新的实时应用程序 UI。当您访问已完成作业的 UI 时，系统会显示持久性应用程序 UI。

运行作业

对于运行中的 EMR Serverless 作业，请访问提供特定于引擎的指标的实时界面。您可以使用 Apache Spark UI 或 Hive Tez UI 来监控和调试作业。要访问这些内容 UIs，请使用 EMR Studio 控制台或使用请求安全 URL 端点。 AWS Command Line Interface

已完成作业

对于已完成的 EMR Serverless 作业，请使用 Spark History Server 或 Persistent Hive Tez UI 访问 Spark 或 Hive 作业运行的作业详细信息、阶段、任务和指标。要访问这些内容 UIs，请使用 EMR Studio 控制台，或者使用请求安全 URL 端点。 AWS Command Line Interface

作业工作线程级别监控

Amazon EMR Serverless 将`AWS/EMRServerless`命名空间和指标组中可用的以下作业工作人员级别的`Job Worker Metrics`指标发送给亚马逊。 CloudWatchEMR Serverless 在作业运行期间，在作业级别、工作人员类型和级别收集单个工作人员的数据点。 capacity-allocation-type您可以使用 `ApplicationId` 作为一个维度来监控属于同一应用程序的多个作业。

> **Note**
>
> 要查看 EMR Serverless 任务在 Amazon CloudWatch 控制台中查看指标时使用的 CPU 和内存总量，请使用统计数据作为总和，将周期设置为 1 分钟。

EMR Serverless 作业工作线程级别指标

指标	说明	单位	维度
`WorkerCpuAllocated`	作业运行中分配给工作线程的 vCPU 核心总数。	vCPU	`JobId`、`JobName`、`ApplicationId` 、`ApplicationName` 、`WorkerType` 和 `CapacityAllocationType`
`WorkerCpuUsed`	作业运行中工作线程使用的 vCPU 核心总数。	vCPU	`JobId`、`JobName`、`ApplicationId` 、`ApplicationName` 、`WorkerType` 和

指标	说明	单位	维度
			CapacityAllocationType
WorkerMemoryAllocated	作业运行中分配给工作线程的总内存（GB）。	千兆字节（GB）	JobId、JobName、ApplicationId 、ApplicationName 、WorkerType 和 CapacityAllocationType
WorkerMemoryUsed	作业运行中工作线程使用的总内存（GB）。	千兆字节（GB）	JobId、JobName、ApplicationId 、ApplicationName 、WorkerType 和 CapacityAllocationType
WorkerEphemeralStorageAllocated	作业运行中分配给工作线程的临时存储字节数。	千兆字节（GB）	JobId、JobName、ApplicationId 、ApplicationName 、WorkerType 和 CapacityAllocationType
WorkerEphemeralStorageUsed	作业运行中工作线程使用的临时存储字节数。	千兆字节（GB）	JobId、JobName、ApplicationId 、ApplicationName 、WorkerType 和 CapacityAllocationType

指标	说明	单位	维度
WorkerStorageReadBytes	作业运行中工作线程从存储中读取的字节数。	字节	JobId、JobName、ApplicationId 、ApplicationName 、WorkerType 和 CapacityAllocationType
WorkerStorageWriteBytes	作业运行中从工作线程写入存储的字节数。	字节	JobId、JobName、ApplicationId 、ApplicationName 、WorkerType 和 CapacityAllocationType

下面的步骤介绍了如何访问各种类型的指标。

Console

使用控制台访问应用程序 UI

1. 按照控制台入门中的说明，导航到 EMR Studio 上的 EMR Serverless 应用程序。
2. 要访问特定于引擎的应用程序 UIs 和正在运行的作业的日志，请执行以下操作：
   a. 选择状态为 RUNNING 的作业。
   b. 在应用程序详细信息页面上选择作业，或导航到作业的作业详细信息页面。
   c. 在显示 UI 下拉菜单下，选择 Spark UI 或 Hive Tez UI，导航到适合您作业类型的应用程序 UI。
   d. 要访问 Spark 引擎日志，请导航到 Spark UI 中的执行程序选项卡，然后选择驱动程序的日志链接。要访问 Hive 引擎日志，请在 Hive Tez UI 中选择相应 DAG 的日志链接。
3. 要访问已完成作业的特定引擎应用程序 UIs 和日志，请执行以下操作：
   a. 选择状态为 SUCCESS 的作业。

b. 在应用程序的应用程序详细信息页面上选择作业，或导航到作业的作业详细信息页面。

c. 在显示 UI 下拉菜单下，选择 Spark History Server 或 Persistent Hive Tez UI，导航到适合您作业类型的应用程序 UI。

d. 要访问 Spark 引擎日志，请导航到 Spark UI 中的执行程序选项卡，然后选择驱动程序的日志链接。要访问 Hive 引擎日志，请在 Hive Tez UI 中选择相应 DAG 的日志链接。

AWS CLI

要使用访问您的应用程序用户界面 AWS CLI

- 要生成一个 URL 用于访问运行中和已完成作业的应用程序 UI，请调用 `GetDashboardForJobRun` API。

```
aws emr-serverless get-dashboard-for-job-run /
--application-id <application-id> /
--job-run-id <job-id>
```

生成的 URL 在 1 小时内有效。

使用 Amazon Managed Service for Prometheus 监控 Spark 指标

在 Amazon EMR 7.1.0 及更高版本中，您可以将 EMR Serverless 与 Amazon Managed Service for Prometheus 集成，来收集 EMR Serverless 作业和应用程序的 Apache Spark 指标。当您使用 AWS 控制台、EMR Serverless API 或提交任务或创建应用程序时，即可使用此集成。 AWS CLI

先决条件

在将 Spark 指标传输到 Amazon Managed Service for Prometheus 之前，应满足以下先决条件。

- 创建 Amazon Managed Service for Prometheus Workspace。此工作空间用作摄取端点。记下端点 - 远程写入 URL中显示的 URL。创建 EMR Serverless 应用程序时，您需要指定 URL。
- 要授予作业对 Amazon Managed Service for Prometheus 的访问权限以进行监控，请将以下策略添加到作业执行角色。

```
{
    "Sid": "AccessToPrometheus",
    "Effect": "Allow",
    "Action": ["aps:RemoteWrite"],
```

```
    "Resource": "arn:aws:aps:<AWS_REGION>:<AWS_ACCOUNT_ID>:workspace/<WORKSPACE_ID>"
}
```

设置

使用 AWS 控制台创建与亚马逊 Prometheus 托管服务集成的应用程序

1. 要创建应用程序，请参阅开始使用 Amazon EMR Serverless。
2. 在创建应用程序时，选择使用自定义设置，然后在要配置的字段中指定信息来配置应用程序。
3. 在应用程序日志和指标下，选择将引擎指标传输到 Amazon Managed Service for Prometheus，然后指定远程写入 URL。
4. 指定所需的任何其他配置设置，然后选择创建并启动应用程序。

使用 AWS CLI 或 EMR 无服务器 API

在运行或命令时，您还可以使用 AWS CLI 或 EMR Serverless API 将您的 EMR 无服务器应用程序与适用于 Prometheus 的亚马逊托管服务集成。`create-application` `start-job-run`

create-application

```
aws emr-serverless create-application \
--release-label emr-7.1.0 \
--type "SPARK" \
--monitoring-configuration '{
    "prometheusMonitoringConfiguration": {
        "remoteWriteUrl": "https://aps-workspaces.<AWS_REGION>.amazonaws.com/
workspaces/<WORKSPACE_ID>/api/v1/remote_write"
    }
}'
```

start-job-run

```
aws emr-serverless start-job-run \
--application-id <APPPLICATION_ID> \
--execution-role-arn <JOB_EXECUTION_ROLE> \
--job-driver '{
    "sparkSubmit": {
        "entryPoint": "local:///usr/lib/spark/examples/src/main/python/pi.py",
        "entryPointArguments": ["10000"],
```

```
        "sparkSubmitParameters": "--conf spark.dynamicAllocation.maxExecutors=10"
    }
}' \
--configuration-overrides '{
     "monitoringConfiguration": {
        "prometheusMonitoringConfiguration": {
            "remoteWriteUrl": "https://aps-workspaces.<AWS_REGION>.amazonaws.com/
workspaces/<WORKSPACE_ID>/api/v1/remote_write"
        }
    }
}'
```

在命令中包含 `prometheusMonitoringConfiguration` 表示 EMR Serverless 必须使用代理运行 Spark 作业，该代理负责收集 Spark 指标并将其写入 Amazon Managed Service for Prometheus 的 `remoteWriteUrl` 端点。然后，您可以使用 Amazon Managed Service for Prometheus 中的 Spark 指标进行可视化、警报和分析。

高级配置属性

EMR Serverless 使用 Spark 中名为 `PrometheusServlet` 的组件来收集 Spark 指标，并将性能数据转换为与 Amazon Managed Service for Prometheus 兼容的数据。默认情况下，EMR Serverless 会在 Spark 中设置默认值，并在使用 `PrometheusMonitoringConfiguration` 提交作业时解析驱动程序和执行程序指标。

下表介绍了在提交 Spark 作业以将指标发送到 Amazon Managed Service for Prometheus 时配置的所有属性。

Spark 属性	默认 值	说明
`spark.metrics.conf.*.sink.prometheusServlet.class`	org.apache.spark.metrics.sink。 PrometheusServlet	Spark 用来将指标发送到 Amazon Managed Service for Prometheus 的类。要覆盖默认行为，请指定您自己的自定义类。
`spark.metrics.conf.*.source.jvm.class`	org.apache.spark.metrics.source。 JvmSource	Spark 用来从底层 Java 虚拟机收集和发送关键指标的类。要停止收集 JVM 指标，可将该属性设置为空字符串（如 ""），

Spark 属性	默认 值	说明
		将其禁用。要覆盖默认行为，请指定您自己的自定义类。
`spark.metrics.conf.driver.sink.prometheusServlet.path`	/metrics/prometheus	Amazon Managed Service for Prometheus 用来从驱动程序收集指标的独特 URL。要覆盖默认行为，请指定您自己的路径。要停止收集驱动程序指标，可将该属性设置为空字符串（如 ""），将其禁用。
`spark.metrics.conf.executor.sink.prometheusServlet.path`	/metrics/executor/prometheus	Amazon Managed Service for Prometheus 用来从执行程序收集指标的独特 URL。要覆盖默认行为，请指定您自己的路径。要停止收集执行程序指标，可将该属性设置为空字符串（如 ""），将其禁用。

有关 Spark 指标的更多信息，请参阅 Apache Spark 指标。

注意事项和限制

当使用 Amazon Managed Service for Prometheus 从 EMR Serverless 收集指标时，请考虑以下注意事项和限制。

- 只有当 Amazon Managed Service for Prometheus 在AWS 区域 中普遍可用的情况下，才支持将 Amazon Managed Service for Prometheus 与 EMR Serverless 结合使用。
- 运行代理以在 Amazon Managed Service for Prometheus 上收集 Spark 指标时需要工作线程提供更多资源。如果选择较小的工作线程规模（例如 1 个 vCPU 工作线程），作业运行时间可能会增加。
- 仅 Amazon EMR 7.1.0 及更高版本支持将 Amazon Managed Service for Prometheus 与 EMR Serverless 结合使用。
- Amazon Managed Service for Prometheus 必须部署在运行 EMR Serverless 的同一账户中才能收集指标。

EMR Serverless 用量指标

您可以使用 Amazon CloudWatch 使用量指标来了解您的账户使用的资源。使用这些指标在 CloudWatch 图表和仪表板上可视化您的服务使用情况。

EMR Serverless 用量指标与服务配额对应。您可以配置警报，以在用量接近服务配额时向您发出警报。有关更多信息，请参阅服务配额用户指南中的服务配额和亚马逊 CloudWatch警报。

有关 EMR Serverless 服务配额的更多信息，请参阅EMR Serverless 端点和配额。

EMR Serverless 的服务配额用量指标

EMR Serverless 在 `AWS/Usage` 命名空间中发布以下服务配额用量指标。

指标	说明
`ResourceCount`	账户中正在运行的指定资源的总数。资源由与指标关联的维度定义。

EMR Serverless 服务配额用量指标的维度

您可以使用以下维度来细化 EMR Serverless 发布的用量指标。

维度	值	说明
`Service`	EMR Serverless	AWS 服务 包含资源的名称。
`Type`	资源	EMR Serverless 报告的实体类型。
`Resource`	vCPU	EMR Serverless 跟踪的资源类型。
`Class`	无	EMR Serverless 跟踪的资源类。

使用无服务器自动执行 EMR Amazon EventBridge

您可以使用 Amazon EventBridge 自动执行系统事件， AWS 服务 并自动响应系统事件，例如应用程序可用性问题或资源更改。 EventBridge 提供近乎实时的系统事件流，这些事件描述了您的 AWS 资源变化。您可以编写简单规则来指示您关注的事件，并指示要在事件匹配规则时执行的自动化操作。使用 EventBridge，您可以自动：

- 调用一个 AWS Lambda 函数
- 将事件中继到 Amazon Kinesis Data Streams
- 激活 AWS Step Functions 状态机
- 通知 Amazon SNS 主题或 Amazon SQS 队列

例如，当您 EventBridge 与 EMR Serverless 一起使用时，您可以在 ETL 任务成功时激活一个 AWS Lambda 函数，或者在 ETL 任务失败时通知 Amazon SNS 主题。

EMR Serverless 会发出四种事件：

- 应用程序状态更改事件：每次应用程序状态更改时发出的事件。有关应用程序状态的更多信息，请参阅应用程序状态。
- 作业运行状态更改事件：每次作业运行状态更改时发出的事件。有关更多信息，请参阅任务运行状态。
- 作业运行重试事件：每次 Amazon EMR Serverless 7.1.0 及更高版本的作业运行重试时发出的事件。
- 作业资源利用率更新事件：作业运行每隔约 30 分钟更新资源利用率时发出的事件。

EMR 无服务器事件示例 EventBridge

EMR Serverless 报告的事件为 `source` 分配值 `aws.emr-serverless`，如以下示例所示。

应用程序状态更改事件

以下示例事件显示了处于 `CREATING` 状态的应用程序。

```
{
    "version": "0",
    "id": "9fd3cf79-1ff1-b633-4dd9-34508dc1e660",
    "detail-type": "EMR Serverless Application State Change",
```

```
    "source": "aws.emr-serverless",
    "account": "123456789012",
    "time": "2022-05-31T21:16:31Z",
    "region": "us-east-1",
    "resources": [],
    "detail": {
        "applicationId": "00f1cbsc6anuij25",
        "applicationName": "3965ad00-8fba-4932-a6c8-ded32786fd42",
        "arn": "arn:aws:emr-serverless:us-east-1:111122223333:/
applications/00f1cbsc6anuij25",
        "releaseLabel": "emr-6.6.0",
        "state": "CREATING",
        "type": "HIVE",
        "createdAt": "2022-05-31T21:16:31.547953Z",
        "updatedAt": "2022-05-31T21:16:31.547970Z",
        "autoStopConfig": {
            "enabled": true,
            "idleTimeout": 15
        },
        "autoStartConfig": {
            "enabled": true
        }
    }
}
```

作业运行状态更改事件

以下示例事件显示了状态从 SCHEDULED 变为 RUNNING 的作业运行。

```
{
    "version": "0",
    "id": "00df3ec6-5da1-36e6-ab71-20f0de68f8a0",
    "detail-type": "EMR Serverless Job Run State Change",
    "source": "aws.emr-serverless",
    "account": "123456789012",
    "time": "2022-05-31T21:07:42Z",
    "region": "us-east-1",
    "resources": [],
    "detail": {
        "jobRunId": "00f1cbn5g4bb0c01",
        "applicationId": "00f1982r1uukb925",
        "arn": "arn:aws:emr-serverless:us-east-1:123456789012:/
applications/00f1982r1uukb925/jobruns/00f1cbn5g4bb0c01",
        "releaseLabel": "emr-6.6.0",
```

```
        "state": "RUNNING",
        "previousState": "SCHEDULED",
        "createdBy": "arn:aws:sts::123456789012:assumed-role/
TestRole-402dcef3ad14993c15d28263f64381e4cda34775/6622b6233b6d42f59c25dd2637346242",
        "updatedAt": "2022-05-31T21:07:42.299487Z",
        "createdAt": "2022-05-31T21:07:25.325900Z"
    }
}
```

作业运行重试事件

以下是作业运行重试事件的示例。

```
{
    "version": "0",
    "id": "00df3ec6-5da1-36e6-ab71-20f0de68f8a0",
    "detail-type": "EMR Serverless Job Run Retry",
    "source": "aws.emr-serverless",
    "account": "123456789012",
    "time": "2022-05-31T21:07:42Z",
    "region": "us-east-1",
    "resources": [],
    "detail": {
        "jobRunId": "00f1cbn5g4bb0c01",
        "applicationId": "00f1982r1uukb925",
        "arn": "arn:aws:emr-serverless:us-east-1:123456789012:/
applications/00f1982r1uukb925/jobruns/00f1cbn5g4bb0c01",
        "releaseLabel": "emr-6.6.0",
        "createdBy": "arn:aws:sts::123456789012:assumed-role/
TestRole-402dcef3ad14993c15d28263f64381e4cda34775/6622b6233b6d42f59c25dd2637346242",
        "updatedAt": "2022-05-31T21:07:42.299487Z",
        "createdAt": "2022-05-31T21:07:25.325900Z",
        //Attempt Details
        "previousAttempt": 1,
        "previousAttemptState": "FAILED",
        "previousAttemptCreatedAt": "2022-05-31T21:07:25.325900Z",
        "previousAttemptEndedAt": "2022-05-31T21:07:30.325900Z",
        "newAttempt": 2,
        "newAttemptCreatedAt": "2022-05-31T21:07:30.325900Z"
    }
}
```

作业资源利用率更新

以下示例事件显示了运行后转至最终状态的作业的最终资源利用率更新。

```
{
    "version": "0",
    "id": "00df3ec6-5da1-36e6-ab71-20f0de68f8a0",
    "detail-type": "EMR Serverless Job Resource Utilization Update",
    "source": "aws.emr-serverless",
    "account": "123456789012",
    "time": "2022-05-31T21:07:42Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:emr-serverless:us-east-1:123456789012:/applications/00f1982r1uukb925/
jobruns/00f1cbn5g4bb0c01"
    ],
    "detail": {
        "applicationId": "00f1982r1uukb925",
        "jobRunId": "00f1cbn5g4bb0c01",
        "attempt": 1,
        "mode": "BATCH",
        "createdAt": "2022-05-31T21:07:25.325900Z",
        "startedAt": "2022-05-31T21:07:26.123Z",
        "calculatedFrom": "2022-05-31T21:07:42.299487Z",
        "calculatedTo": "2022-05-31T21:07:30.325900Z",
        "resourceUtilizationFinal": true,
        "resourceUtilizationForInterval": {
            "vCPUHour": 0.023,
            "memoryGBHour": 0.114,
            "storageGBHour": 0.228
        },
        "billedResourceUtilizationForInterval": {
            "vCPUHour": 0.067,
            "memoryGBHour": 0.333,
            "storageGBHour": 0
        },
        "totalResourceUtilization": {
            "vCPUHour": 0.023,
            "memoryGBHour": 0.114,
            "storageGBHour": 0.228
        },
        "totalBilledResourceUtilization": {
            "vCPUHour": 0.067,
            "memoryGBHour": 0.333,
            "storageGBHour": 0
        }
```

```
    }
}
```

仅当作业转至运行状态时，startAt 字段才会出现在事件中。

标注资源

使用标签为每个资源分配您自己的元数据来帮助您管理 EMR Serverless 资源。本节概述了标签功能，并介绍了如何创建标签。

主题

什么是标签？

标签是您分配给 AWS 资源的标签。每个标签都由键和值组成，这两个参数都由您定义。标签使您能够按用途、所有者和环境等属性对 AWS 资源进行分类。在具有相同类型的许多资源时，请根据分配给资源的标签快速识别具体的资源。例如，请为 Amazon EMR Serverless 应用程序定义一组标签，以跟踪每个应用程序的所有者和堆栈级别。我们建议为每个资源类型设计一组一致的标签键。

标签不会自动分配至资源。向资源添加标签后，请随时修改标签的值或从资源中移除标签。标签对 Amazon EMR Serverless 没有任何语义上的意义，应严格按字符串进行解析。如果您添加的标签的键与该资源上现有标签的键相同，则新值会覆盖旧值。

如果您使用 IAM，则可以控制 AWS 账户中哪些用户有权管理标签。有关基于标签的访问策略示例，请参阅基于标签的访问控制策略。

标记您的资源

您可以标记新的或现有的应用程序和作业运行。如果您使用的是 Amazon EMR Serverless API AWS CLI、或 AWS 软件开发工具包，则可以使用相关 API 操作中的`tags`参数将标签应用于新资源。您也可以通过使用 `TagResource` API 操作将标签应用于现有资源。

您可使用某些创建资源操作，以在创建资源时为其指定标签。在这种情况下，如果在创建资源期间无法应用标签，则无法创建资源。此机制可确保对于您希望在创建时标记的资源，要么使用指定的标签创建，要么完全不创建。如果在创建时标记资源，则不需要在创建资源后运行自定义标记脚本。

下表描述了可以标记的 Amazon EMR Serverless 资源。

可标记的 资源

资源	支持标签	支持标签传播	支持在创建时添加标签（Amazon EMR 无服务器 API AWS CLI和 SDK） AWS	创建 API（可以在创建过程中添加标签）
应用程序	是	不，与应用程序关联的标签不会传播到提交给应用程序的作业运行。	是	`CreateApplication`
任务运行	是	否	是	`StartJobRun`

标记限制

标签适用以下基本限制：

- 每个资源最多可以有 50 个用户创建的标签。
- 对于每个资源，每个标签键必须是唯一的，并且每个标签键只能有一个值。
- 最大键长度为 128 个 Unicode 字符（采用 UTF-8 格式）。
- 最大值长度为 256 个 Unicode 字符（采用 UTF-8 格式）。
- 允许使用的字符包括以 UTF-8 表示的字母、数字和空格以及以下字符：_ . : / = + - @。
- 标签的键不能是空字符串。标签值可以是空字符串，但是不能是 null。
- 标签键和值区分大小写。
- 请不要使用 `AWS:` 或任何大小写组合，例如键或值的前缀。这些字符串保留供 AWS 使用。

使用 AWS CLI 和 Amazon EMR Serverless API 处理标签

使用以下 AWS CLI 命令或 Amazon EMR Serverless API 操作为您的资源添加、更新、列出和删除标签。

标签的 CLI 命令和 API 操作

资源	支持标签	支持标签传播
添加或覆盖一个或多个标签	tag-resource	TagResource
列出资源的标签	list-tags-for-resource	ListTagsForResource
删除一个或多个标签	untag-resource	UntagResource

以下示例展示了如何使用 AWS CLI为资源添加标签或取消标签。

标记现有应用程序

以下命令可标记现有应用程序。

```
aws emr-serverless tag-resource --resource-arn resource_ARN --tags team=devs
```

取消现有应用程序标记

以下命令可从现有应用程序中删除标签。

```
aws emr-serverless untag-resource --resource-arn resource_ARN --tag-keys tag_key
```

列出资源的标签

以下命令列出与现有资源关联的标签。

```
aws emr-serverless list-tags-for-resource --resource-arn resource_ARN
```

EMR Serverless 教程

本节介绍了使用 EMR Serverless 应用程序时的常见用例。其中包括各种工具，例如用于处理大型数据集的 Hudi 和 Iceberg，以及使用 Python 和 Python 库提交 Spark 作业的工具。

主题

将 Java 17 与 Amazon EMR Serverless 结合使用

在 Amazon EMR 6.11.0 及更高版本中，请将 EMR Serverless Spark 作业配置为使用 Java 虚拟机 (JVM) 的 Java 17 运行时。使用以下方法之一配置 Spark 与 Java 17。

`JAVA_HOME`

要覆盖 EMR Serverless 6.11.0 及更高版本的 JVM 设置，请向其 `spark.emr-serverless.driverEnv` 和 `spark.executorEnv` 环境分类提供 `JAVA_HOME` 设置。

x86_64

设置所需的属性以指定 Java 17 作为 Spark 驱动程序和执行程序的 `JAVA_HOME` 配置：

```
--conf spark.emr-serverless.driverEnv.JAVA_HOME=/usr/lib/jvm/java-17-amazon-
corretto.x86_64/
```

```
--conf spark.executorEnv.JAVA_HOME=/usr/lib/jvm/java-17-amazon-corretto.x86_64/
```

arm_64

设置所需的属性以指定 Java 17 作为 Spark 驱动程序和执行程序的 JAVA_HOME 配置：

```
--conf spark.emr-serverless.driverEnv.JAVA_HOME=/usr/lib/jvm/java-17-amazon-
corretto.aarch64/
--conf spark.executorEnv.JAVA_HOME=/usr/lib/jvm/java-17-amazon-corretto.aarch64/
```

spark-defaults

或者，您也可以在 spark-defaults 分类中指定 Java 17，以覆盖 EMR Serverless 6.11.0 及更高版本的 JVM 设置。

x86_64

在 spark-defaults 分类中指定 Java 17：

```
{
"applicationConfiguration": [
         {
            "classification": "spark-defaults",
            "properties": {
               "spark.emr-serverless.driverEnv.JAVA_HOME" : "/usr/lib/jvm/java-17-
amazon-corretto.x86_64/",
               "spark.executorEnv.JAVA_HOME": "/usr/lib/jvm/java-17-amazon-
corretto.x86_64/"
            }
         }
      ]
}
```

arm_64

在 spark-defaults 分类中指定 Java 17：

```
{
"applicationConfiguration": [
         {
            "classification": "spark-defaults",
```

```
            "properties": {
               "spark.emr-serverless.driverEnv.JAVA_HOME" : "/usr/lib/jvm/java-17-
amazon-corretto.aarch64/",
               "spark.executorEnv.JAVA_HOME": "/usr/lib/jvm/java-17-amazon-
corretto.aarch64/"
            }
         }
      ]
}
```

将 Apache Hudi 与 EMR Serverless 结合使用

本节介绍了如何将 Apache Hudi 与 EMR Serverless 应用程序结合使用。Hudi 是一个数据管理框架，使数据处理更加简单。

将 Apache Hudi 与 EMR Serverless 应用程序结合使用

1. 在相应的 Spark 作业运行中设置所需的 Spark 属性。

```
spark.jars=/usr/lib/hudi/hudi-spark-bundle.jar,/usr/lib/hudi/hudi-utilities-
bundle.jar,/usr/lib/hudi/hudi-aws-bundle.jar
spark.serializer=org.apache.spark.serializer.KryoSerializer
```

2. 要将 Hudi 表同步到配置的目录，请将 Glue AWS 数据目录指定为您的元数据库，或者配置外部元数据库。EMR Serverless 支持 hms 作为 Hudi 工作负载 Hive 表的同步模式。EMR Serverless 默认激活此属性。要进一步了解如何设置元存储，请参阅 EMR Serverless 的元存储配置。

   > **Important**
   >
   > EMR Serverless 不支持 HIVEQL 或 JDBC 作为 Hive 表的同步模式选项来处理 Hudi 工作负载。要了解更多信息，请参阅同步模式。

   使用 Glue AWS 数据目录作为元数据存储时，请为 Hudi 作业指定以下配置属性。

```
--conf spark.jars=/usr/lib/hudi/hudi-spark-bundle.jar,
--conf spark.serializer=org.apache.spark.serializer.KryoSerializer,
--conf
 spark.hadoop.hive.metastore.client.factory.class=com.amazonaws.glue.catalog.metastore.AWSG
```

要了解有关 Amazon EMR 的 Apache Hudi 版本的更多信息，请参阅 Hudi 版本历史记录。

将 Apache Iceberg 与 EMR Serverless 结合使用

本节介绍了如何将 Apache Iceberg 与 EMR Serverless 应用程序结合使用。Apache Iceberg 是一种表格式，有助于处理数据湖中的大型数据集。

将 Apache Iceberg 与 EMR Serverless 应用程序结合使用

1. 在相应的 Spark 作业运行中设置所需的 Spark 属性。

```
spark.jars=/usr/share/aws/iceberg/lib/iceberg-spark3-runtime.jar
```

2. 将 AWS Glue 数据目录指定为元数据仓或配置外部元数据仓。要了解有关设置元数据存储的更多信息，请参阅 EMR Serverless 的元存储配置。

   配置要用于 Iceberg 的元存储属性。例如，如果要使用 AWS Glue 数据目录，请在应用程序配置中设置以下属性。

```
spark.sql.catalog.dev.warehouse=s3://amzn-s3-demo-bucket/EXAMPLE-PREFIX/
spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions
spark.sql.catalog.dev=org.apache.iceberg.spark.SparkCatalog
spark.sql.catalog.dev.catalog-impl=org.apache.iceberg.aws.glue.GlueCatalog
spark.hadoop.hive.metastore.client.factory.class=com.amazonaws.glue.catalog.metastore.AWSGl
```

   当您使用 AWS Glue 数据目录作为元数据存储时，请为 Iceberg 作业指定以下配置属性。

```
--conf spark.jars=/usr/share/aws/iceberg/lib/iceberg-spark3-runtime.jar,
--conf
 spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions,
--conf spark.sql.catalog.dev=org.apache.iceberg.spark.SparkCatalog,
--conf spark.sql.catalog.dev.catalog-impl=org.apache.iceberg.aws.glue.GlueCatalog,
--conf spark.sql.catalog.dev.warehouse=s3://amzn-s3-demo-bucket/EXAMPLE-PREFIX/
--conf
 spark.hadoop.hive.metastore.client.factory.class=com.amazonaws.glue.catalog.metastore.AWSG
```

要了解有关 Amazon EMR 的 Apache Iceberg 版本的更多信息，请参阅 Iceberg 版本历史记录。

将 Python 库与 EMR Serverless 结合使用

在 Amazon EMR 无服务器应用程序上运行 PySpark 作业时，请将各种 Python 库打包为依赖项。为此，请使用原生 Python 功能，构建虚拟环境，或者直接将 PySpark 作业配置为使用 Python 库。本页涵盖了每种方法。

使用 Python 原生功能

设置以下配置时，使用 PySpark 将 Python 文件 (`.py`)、压缩的 Python 包 (`.zip`) 和 Egg 文件 (`.egg`) 上传到 Spark 执行器。

```
--conf spark.submit.pyFiles=s3://amzn-s3-demo-bucket/EXAMPLE-PREFIX/<.py|.egg|.zip
 file>
```

有关如何使用 Python 虚拟环境执行 PySpark 作业的更多详细信息，请参阅使用 PySpark 原生功能。

使用 EMR Notebook 时，您可以通过执行以下代码来使 Python 依赖项在 Notebook 中可用：

```
    %%configure -f
{
    "conf": {
    "spark.submit.pyFiles":"s3:///amzn-s3-demo-bucket/EXAMPLE-PREFIX/<.py|.egg|.zip
 file>
                    }
}
```

构建 Python 虚拟环境

要为一项 PySpark 作业打包多个 Python 库，请创建隔离的 Python 虚拟环境。

1. 要构建 Python 虚拟环境，请使用以下命令。所示示例将 `scipy` 和 `matplotlib` 包安装到虚拟环境包中，并将存档复制到 Amazon S3 位置。

   > **Important**
   >
   > 您必须在类似的 Amazon Linux 2 环境中运行以下命令，并使用与 EMR Serverless 中所用版本相同的 Python，即适用于 Amazon EMR 6.6.0 的 Python 3.7.10。你可以在 EMR 无服务器示例存储库中找到 Dockerfile 示例。 GitHub

```
# initialize a python virtual environment
python3 -m venv pyspark_venvsource
source pyspark_venvsource/bin/activate

# optionally, ensure pip is up-to-date
pip3 install --upgrade pip

# install the python packages
pip3 install scipy
pip3 install matplotlib

# package the virtual environment into an archive
pip3 install venv-pack
venv-pack -f -o pyspark_venv.tar.gz

# copy the archive to an S3 location
aws s3 cp pyspark_venv.tar.gz s3://amzn-s3-demo-bucket/EXAMPLE-PREFIX/

# optionally, remove the virtual environment directory
rm -fr pyspark_venvsource
```

2. 提交 Spark 作业，并将属性设置为使用 Python 虚拟环境。

```
--conf spark.archives=s3://amzn-s3-demo-bucket/EXAMPLE-PREFIX/
pyspark_venv.tar.gz#environment
--conf spark.emr-serverless.driverEnv.PYSPARK_DRIVER_PYTHON=./environment/bin/
python
--conf spark.emr-serverless.driverEnv.PYSPARK_PYTHON=./environment/bin/python
--conf spark.executorEnv.PYSPARK_PYTHON=./environment/bin/python
```

请注意，如果不覆盖原始 Python 二进制文件，前面设置序列中的第二个配置将是 `--conf spark.executorEnv.PYSPARK_PYTHON=python`。

有关如何使用 Python 虚拟环境进行 PySpark 作业的更多信息，请参阅使用 Virtualenv。有关如何提交 Spark 作业的更多示例，请参阅运行 EMR Serverless 作业时使用 Spark 配置。

将 PySpark 作业配置为使用 Python 库

在 Amazon EMR 6.12.0 及更高版本中，您可以直接将 EMR 无服务器 PySpark作业配置为使用流行的数据科学 Python 库，例如 p andas，无需任何其他设置。NumPyPyArrow

以下示例演示如何为 PySpark作业打包每个 Python 库。

NumPy

NumPy 是一个用于科学计算的 Python 库，它为数学、排序、随机模拟和基本统计提供多维数组和运算。要使用 NumPy，请运行以下命令：

```
import numpy
```

pandas

pandas 是一个基于它的 Python 库。 NumPy熊猫图书馆为数据科学家提供了DataFrame数据结构和数据分析工具。要使用 pandas，请运行以下命令：

```
import pandas
```

PyArrow

PyArrow 是一个 Python 库，用于管理内存中的列式数据以提高工作性能。 PyArrow 基于 Apache Arrow 跨语言开发规范，这是一种以列式格式表示和交换数据的标准方法。要使用 PyArrow，请运行以下命令：

```
import pyarrow
```

在 EMR Serverless 中使用不同的 Python 版本

除了 将 Python 库与 EMR Serverless 结合使用 中的用例之外，您还可以使用 Python 虚拟环境来处理不同于 Amazon EMR 发行版中打包的 Amazon EMR Serverless 应用程序版本的 Python 版本。为此，请使用想要的 Python 版本构建 Python 虚拟环境。

从 Python 虚拟环境提交作业

1. 使用以下示例中的命令构建虚拟环境。此示例将 Python 3.9.9 安装到虚拟环境包中，并将存档复制到 Amazon S3 位置。

> **Important**
>
> 如果使用 Amazon EMR 7.0.0 及更高版本，请在类似于 EMR Serverless 应用程序所用环境的 Amazon Linux 2023 环境中运行命令。
> 如果使用 6.15.0 或更低版本，请在类似 Amazon Linux 2 环境中运行以下命令。

```
# install Python 3.9.9 and activate the venv
yum install -y gcc openssl-devel bzip2-devel libffi-devel tar gzip wget make
wget https://www.python.org/ftp/python/3.9.9/Python-3.9.9.tgz && \
tar xzf Python-3.9.9.tgz && cd Python-3.9.9 && \
./configure --enable-optimizations && \
make altinstall

# create python venv with Python 3.9.9
python3.9 -m venv pyspark_venv_python_3.9.9 --copies
source pyspark_venv_python_3.9.9/bin/activate

# copy system python3 libraries to venv
cp -r /usr/local/lib/python3.9/* ./pyspark_venv_python_3.9.9/lib/python3.9/

# package venv to archive.
# **Note** that you have to supply --python-prefix option
# to make sure python starts with the path where your
# copied libraries are present.
# Copying the python binary to the "environment" directory.
pip3 install venv-pack
venv-pack -f -o pyspark_venv_python_3.9.9.tar.gz --python-prefix /home/hadoop/
environment

# stage the archive in S3
aws s3 cp pyspark_venv_python_3.9.9.tar.gz s3://<path>

# optionally, remove the virtual environment directory
rm -fr pyspark_venv_python_3.9.9
```

2. 设置属性以使用 Python 虚拟环境，然后提交 Spark 作业。

```
# note that the archive suffix "environment" is the same as the directory where you
 copied the Python binary.
```

```
--conf spark.archives=s3://amzn-s3-demo-bucket/EXAMPLE-PREFIX/
pyspark_venv_python_3.9.9.tar.gz#environment
--conf spark.emr-serverless.driverEnv.PYSPARK_DRIVER_PYTHON=./environment/bin/
python
--conf spark.emr-serverless.driverEnv.PYSPARK_PYTHON=./environment/bin/python
--conf spark.executorEnv.PYSPARK_PYTHON=./environment/bin/python
```

有关如何使用 Python 虚拟环境进行 PySpark 作业的更多信息，请参阅使用 Virtualenv。有关如何提交 Spark 作业的更多示例，请参阅运行 EMR Serverless 作业时使用 Spark 配置。

将 Delta Lake OSS 与 EMR Serverless 结合使用

Amazon EMR 6.9.0 及更高版本

> **Note**
>
> Amazon EMR 7.0.0 及更高版本使用 Delta Lake 3.0，该版本将 `delta-core.jar` 文件更名为 `delta-spark.jar`。如果您使用 Amazon EMR 7.0.0 或更高版本，请务必在配置中指定 `delta-spark.jar`。

Amazon EMR 6.9.0 及更高版本包含 Delta Lake，因此无需再自行打包 Delta Lake 或为 EMR Serverless 作业提供 `--packages` 标志。

1. 提交 EMR Serverless 作业时，请确保具有以下配置属性，并在 `sparkSubmitParameters` 字段中包含以下参数。

```
--conf spark.jars=/usr/share/aws/delta/lib/delta-core.jar,/usr/share/aws/delta/lib/
delta-storage.jar
    --conf spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension
    --conf
 spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog
```

2. 创建本地 `delta_sample.py` 以测试创建和读取 Delta 表。

```
# delta_sample.py
    from pyspark.sql import SparkSession

    import uuid
```

```
    url = "s3://amzn-s3-demo-bucket/delta-lake/output/%s/" % str(uuid.uuid4())
    spark = SparkSession.builder.appName("DeltaSample").getOrCreate()

    ## creates a Delta table and outputs to target S3 bucket
    spark.range(5).write.format("delta").save(url)

    ## reads a Delta table and outputs to target S3 bucket
    spark.read.format("delta").load(url).show
```

3. 使用 AWS CLI，将`delta_sample.py`文件上传到您的 Amazon S3 存储桶。然后使用 `start-job-run` 命令向现有的 EMR Serverless 应用程序提交作业。

```
aws s3 cp delta_sample.py s3://amzn-s3-demo-bucket/code/

    aws emr-serverless start-job-run \
        --application-id application-id \
        --execution-role-arn job-role-arn \
        --name emr-delta \
        --job-driver '{
            "sparkSubmit": {
                "entryPoint": "s3://amzn-s3-demo-bucket/code/delta_sample.py",
                "sparkSubmitParameters": "--conf spark.jars=/usr/share/
aws/delta/lib/delta-core.jar,/usr/share/aws/delta/lib/delta-storage.jar --
conf spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension --conf
 spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog"
            }
        }'
```

要将 Python 库与 Delta Lake 结合使用，请通过将其打包为依赖项或将其用作自定义映像来添加 `delta-core` 库。

或者，您可以使用 `SparkContext.addPyFile` 从 `delta-core` JAR 文件添加 Python 库：

```
import glob
from pyspark.sql import SparkSession

spark = SparkSession.builder.getOrCreate()
spark.sparkContext.addPyFile(glob.glob("/usr/share/aws/delta/lib/delta-core_*.jar")[0])
```

Amazon EMR 6.8.0 及更低版本

如果您使用的是 Amazon EMR 6.8.0 或更低版本，请按照以下步骤将 Delta Lake OSS 与 EMR Serverless 应用程序结合使用。

1. 要在您的 Amazon EMR Serverless 应用程序上构建与 Spark 版本兼容的 Delta Lak e 开源版本，请导航到 Delta GitHub 并按照说明进行操作。
2. 将三角洲湖库上传到您的 Amazon S3 存储桶中 AWS 账户。
3. 在应用程序配置中提交 EMR Serverless 作业时，请包含存储桶中现有的 Delta Lake JAR 文件。

```
--conf spark.jars=s3://amzn-s3-demo-bucket/jars/delta-core_2.12-1.1.0.jar
```

4. 为确保您可以对 Delta 表进行读取和写入，请运行示例 PySpark测试。

```
from pyspark import SparkConf, SparkContext
    from pyspark.sql import HiveContext, SparkSession

    import uuid

    conf = SparkConf()
    sc = SparkContext(conf=conf)
    sqlContext = HiveContext(sc)

    url = "s3://amzn-s3-demo-bucket/delta-lake/output/1.0.1/%s/" %
 str(uuid.uuid4())

    ## creates a Delta table and outputs to target S3 bucket
    session.range(5).write.format("delta").save(url)

    ## reads a Delta table and outputs to target S3 bucket
    session.read.format("delta").load(url).show
```

从 Airflow 提交 EMR Serverless 作业

Apache Airflow 中的 Amazon Provider 提供 EMR Serverless 运算符。有关运算符的更多信息，请参阅 Apache Airflow 文档中的 Amazon EMR Serverless Operators。

您可以使用 `EmrServerlessCreateApplicationOperator` 创建 Spark 或 Hive 应用程序。还可以使用 `EmrServerlessStartJobOperator` 通过新的应用程序启动一项或多项作业。

要在 Airflow 2.2.2 版本的 Amazon Managed Workflows for Apache Airflow（MWAA）中使用运算符，请在 `requirements.txt` 文件中添加以下行，并更新 MWAA 环境以使用新文件。

```
apache-airflow-providers-amazon==6.0.0
boto3>=1.23.9
```

请注意，Amazon Provider 5.0.0 版中增加了 EMR Serverless 支持。6.0.0 是与 Airflow 2.2.2 兼容的最新版本。您可以在 MWAA 上使用 Airflow 2.4.3 的后续版本。

下面的简略示例展示了如何创建应用程序、运行多个 Spark 作业，然后停止应用程序。EMR 无服务器示例存储库中提供了完整的示例。 GitHub 有关 `sparkSubmit` 配置的更多详细信息，请参阅 运行 EMR Serverless 作业时使用 Spark 配置。

```
from datetime import datetime

from airflow import DAG
from airflow.providers.amazon.aws.operators.emr import (
    EmrServerlessCreateApplicationOperator,
    EmrServerlessStartJobOperator,
    EmrServerlessDeleteApplicationOperator,
)

# Replace these with your correct values
JOB_ROLE_ARN = "arn:aws:iam::account-id:role/emr_serverless_default_role"
S3_LOGS_BUCKET = "amzn-s3-demo-bucket"

DEFAULT_MONITORING_CONFIG = {
    "monitoringConfiguration": {
        "s3MonitoringConfiguration": {"logUri": f"s3://amzn-s3-demo-bucket/logs/"}
    },
}

with DAG(
    dag_id="example_endtoend_emr_serverless_job",
    schedule_interval=None,
    start_date=datetime(2021, 1, 1),
    tags=["example"],
    catchup=False,
) as dag:
    create_app = EmrServerlessCreateApplicationOperator(
        task_id="create_spark_app",
        job_type="SPARK",
```

```
        release_label="emr-6.7.0",
        config={"name": "airflow-test"},
    )

    application_id = create_app.output

    job1 = EmrServerlessStartJobOperator(
        task_id="start_job_1",
        application_id=application_id,
        execution_role_arn=JOB_ROLE_ARN,
        job_driver={
            "sparkSubmit": {
                "entryPoint": "local:///usr/lib/spark/examples/src/main/python/
pi_fail.py",
            }
        },
        configuration_overrides=DEFAULT_MONITORING_CONFIG,
    )

    job2 = EmrServerlessStartJobOperator(
        task_id="start_job_2",
        application_id=application_id,
        execution_role_arn=JOB_ROLE_ARN,
        job_driver={
            "sparkSubmit": {
                "entryPoint": "local:///usr/lib/spark/examples/src/main/python/pi.py",
                "entryPointArguments": ["1000"]
            }
        },
        configuration_overrides=DEFAULT_MONITORING_CONFIG,
    )

    delete_app = EmrServerlessDeleteApplicationOperator(
        task_id="delete_app",
        application_id=application_id,
        trigger_rule="all_done",
    )

    (create_app >> [job1, job2] >> delete_app)
```

将 Hive 用户定义的函数与 EMR Serverless 结合使用

Hive 用户定义函数 (UDFs) 允许您创建自定义函数来处理记录或记录组。在本教程中，您将使用示例 UDF 和预先存在的 Amazon EMR Serverless 应用程序来运行输出查询结果的作业。要了解如何设置应用程序，请参阅 开始使用 Amazon EMR Serverless。

将 UDF 与 EMR Serverless 结合使用

1. 导航至查看GitHub示例 UDF。克隆存储库并切换到要使用的 git 分支。更新存储库 `pom.xml` 文件中的 `maven-compiler-plugin` 以获取源。同时将目标 Java 版本配置更新为 `1.8`。运行`mvn package -DskipTests`以创建包含您的示例的 JAR 文件 UDFs。
2. 创建 JAR 文件后，使用以下命令将其上传到 S3 存储桶。

```
aws s3 cp brickhouse-0.8.2-JS.jar s3://amzn-s3-demo-bucket/jars/
```

3. 创建一个示例文件，使用其中一个 UDF 示例函数。将此查询另存为 `udf_example.q` 并上传到 S3 存储桶。

```
add jar s3://amzn-s3-demo-bucket/jars/brickhouse-0.8.2-JS.jar;
CREATE TEMPORARY FUNCTION from_json AS 'brickhouse.udf.json.FromJsonUDF';
select from_json('{"key1":[0,1,2], "key2":[3,4,5,6], "key3":[7,8,9]}', map("",
 array(cast(0 as int))));
select from_json('{"key1":[0,1,2], "key2":[3,4,5,6], "key3":[7,8,9]}', map("",
 array(cast(0 as int))))["key1"][2];
```

4. 提交以下 Hive 作业。

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "hive": {
        "query": "s3://amzn-s3-demo-bucket/queries/udf_example.q",
        "parameters": "--hiveconf hive.exec.scratchdir=s3://amzn-s3-demo-bucket/
emr-serverless-hive/scratch --hiveconf hive.metastore.warehouse.dir=s3://'$BUCKET'/
emr-serverless-hive/warehouse"
    }
}' --configuration-overrides '{
    "applicationConfiguration": [{
        "classification": "hive-site",
        "properties": {
```

```
            "hive.driver.cores": "2",
            "hive.driver.memory": "6G"
        }
    }],
    "monitoringConfiguration": {
        "s3MonitoringConfiguration": {
            "logUri": "s3://amzn-s3-demo-bucket/logs/"
        }
    }
}'
```

5. 使用 get-job-run 命令检查作业的状态。等待状态变为 SUCCESS。

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

6. 使用以下命令下载输出文件。

```
aws s3 cp --recursive s3://amzn-s3-demo-bucket/logs/applications/application-id/
jobs/job-id/HIVE_DRIVER/ .
```

stdout.gz 文件类似下列内容。

```
{"key1":[0,1,2],"key2":[3,4,5,6],"key3":[7,8,9]}
2
```

在 EMR Serverless 中使用自定义映像

主题

- 使用自定义 Python 版本
- 使用自定义 Java 版本
- 构建数据科学映像
- 使用 Apache Sedona 处理地理空间数据
- 使用自定义映像的许可信息

使用自定义 Python 版本

您可以构建自定义映像，以使用不同版本的 Python。例如，要在 Spark 作业中使用 Python 3.10 版本，请运行以下命令：

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

USER root

# install python 3
RUN yum install -y gcc openssl-devel bzip2-devel libffi-devel tar gzip wget make
RUN wget https://www.python.org/ftp/python/3.10.0/Python-3.10.0.tgz && \
tar xzf Python-3.10.0.tgz && cd Python-3.10.0 && \
./configure --enable-optimizations && \
make altinstall

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

在提交 Spark 作业之前，请按如下方式设置属性以使用 Python 虚拟环境。

```
--conf spark.emr-serverless.driverEnv.PYSPARK_DRIVER_PYTHON=/usr/local/bin/python3.10
--conf spark.emr-serverless.driverEnv.PYSPARK_PYTHON=/usr/local/bin/python3.10
--conf spark.executorEnv.PYSPARK_PYTHON=/usr/local/bin/python3.10
```

使用自定义 Java 版本

以下示例演示了如何构建自定义映像，以便在 Spark 作业中使用 Java 11。

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

USER root

# install JDK 11
RUN amazon-linux-extras install java-openjdk11

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

在提交 Spark 作业之前，请按如下方式设置 Spark 属性以使用 Java 11。

```
--conf spark.executorEnv.JAVA_HOME=/usr/lib/jvm/java-11-
openjdk-11.0.16.0.8-1.amzn2.0.1.x86_64
--conf spark.emr-serverless.driverEnv.JAVA_HOME=/usr/lib/jvm/java-11-
openjdk-11.0.16.0.8-
```

构建数据科学映像

以下示例说明如何包含常见的数据科学 Python 包，例如 Pandas 和 NumPy。

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

USER root

# python packages
RUN pip3 install boto3 pandas numpy
RUN pip3 install -U scikit-learn==0.23.2 scipy
RUN pip3 install sk-dist
RUN pip3 install xgboost

# EMR Serverless runs the image as hadoop
USER hadoop:hadoop
```

使用 Apache Sedona 处理地理空间数据

以下示例演示了如何构建包含 Apache Sedona 的映像以进行地理空间处理。

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

USER root

RUN yum install -y wget
RUN wget https://repo1.maven.org/maven2/org/apache/sedona/sedona-core-3.0_2.12/1.3.0-
incubating/sedona-core-3.0_2.12-1.3.0-incubating.jar -P /usr/lib/spark/jars/
RUN pip3 install apache-sedona

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

使用自定义映像的许可信息

您可以利用 EMR Serverless 构建自定义映像，以执行特定任务或使用特定版本的软件包。自定义映像的修改和分发可能受规则和许可条款的约束。许可文本出现在后面的小节中。

适用于自定义映像的许可

Amazon.com 及其附属公司版权所有；保留所有权利。本软件属于AWS 客户协议下的 AWS 内容，未经许可不得分发。除了AWS 知识产权许可中的权限外，许可 AWS 方还授予您以下额外权限：

允许创建、复制和使用 AWS 内容的衍生作品，前提是满足以下条件：

- 您不会修改 AWS 内容本身，任何衍生品严格来说都是您添加新内容的结果。
- 内部复制品必须保留上述版权声明。
- 根据本许可条款，不允许以源代码或二进制形式进行外部分发，无论是否进行修改。

有关使用自定义映像的更多信息，请参阅在 EMR Serverless 中使用自定义映像。

在 Amazon EMR Serverless 上使用适用于 Apache Spark 的 Amazon Redshift 集成

在 Amazon EMR 发行版 6.9.0 及更高版本中，每个版本的映像都包含 Apache Spark 和 Amazon Redshift 之间的连接器。通过此连接器，请在 Amazon EMR Serverless 上使用 Spark 来处理存储在 Amazon Redshift 中的数据。集成基于 `spark-redshift` 开源连接器。对于 Amazon EMR Serverless，适用于 Apache Spark 的 Amazon Redshift 集成已作为本地集成包含在内。

主题

- 使用适用于 Apache Spark 的 Amazon Redshift 集成启动 Spark 应用程序
- 使用适用于 Apache Spark 的 Amazon Redshift 集成进行身份验证
- 在 Amazon Redshift 中进行读取和写入
- 使用 Spark 连接器时的注意事项和限制

使用适用于 Apache Spark 的 Amazon Redshift 集成启动 Spark 应用程序

要使用与 EMR Serverless 6.9.0 的集成，请在 Spark 作业中传递所需的 Spark-Redshift 依赖项。使用 `--jars` 包含 Redshift 连接器相关库。要访问 `--jars` 选项支持的其他文件位置，请参阅 Apache Spark 文档的 Advanced Dependency Management 部分。

- `spark-redshift.jar`
- `spark-avro.jar`
- `RedshiftJDBC.jar`
- `minimal-json.jar`

Amazon EMR 6.10.0 及更高版本不需要 `minimal-json.jar` 依赖关系，并且默认情况下会自动将其他依赖项安装到每个集群。以下示例展示了如何使用适用于 Apache Spark 的 Amazon Redshift 集成启动 Spark 应用程序。

Amazon EMR 6.10.0 +

通过 EMR Serverless 6.10.0 及更高版本上适用于 Apache Spark 的 Amazon Redshift 集成，在 Amazon EMR Serverless 上启动 Spark 作业。

```
spark-submit my_script.py
```

Amazon EMR 6.9.0

要通过 EMR Serverless 6.9.0 上适用于 Apache Spark 的 Amazon Redshift 集成，在 Amazon EMR Serverless 上启动 Spark 作业，请使用以下示例中所示的 `--jars` 选项。请注意，`--jars` 选项列出的路径是 JAR 文件的默认路径。

```
--jars
    /usr/share/aws/redshift/jdbc/RedshiftJDBC.jar,
    /usr/share/aws/redshift/spark-redshift/lib/spark-redshift.jar,
    /usr/share/aws/redshift/spark-redshift/lib/spark-avro.jar,
    /usr/share/aws/redshift/spark-redshift/lib/minimal-json.jar
```

```
spark-submit \
  --jars /usr/share/aws/redshift/jdbc/RedshiftJDBC.jar,/usr/share/aws/redshift/
spark-redshift/lib/spark-redshift.jar,/usr/share/aws/redshift/spark-redshift/lib/
spark-avro.jar,/usr/share/aws/redshift/spark-redshift/lib/minimal-json.jar \
  my_script.py
```

使用适用于 Apache Spark 的 Amazon Redshift 集成进行身份验证

用于 AWS Secrets Manager 检索凭证并连接亚马逊 Redshift

您可以将凭证存储在 Secrets Manager 中，向 Amazon Redshift 安全地进行身份验证，并让 Spark 作业调用 `GetSecretValue` API 来获取凭证：

```
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
```

使用 JDBC 驱动程序对 Amazon Redshift 进行身份验证

在 JDBC URL 中设置用户名和密码

您可以在 JDBC URL 中指定 Amazon Redshift 数据库名称和密码，向 Amazon Redshift 集群的 Spark 作业进行身份验证。

> **Note**
>
> 如果您在 URL 中传递数据库凭证，则有权访问该 URL 的任何人也可以访问凭证。通常不建议使用此方法，因为这不是一个安全的选项。

如果您的应用程序不考虑安全性，请使用以下格式在 JDBC URL 中设置用户名和密码：

```
jdbc:redshift://redshifthost:5439/database?user=username&password=password
```

将基于 IAM 的身份验证与 Amazon EMR Serverless 作业执行角色结合使用

从 Amazon EMR Serverless 发行版 6.9.0 开始，Amazon Redshift JDBC 驱动程序 2.1 或更高版本将打包到环境中。您可以使用 JDBC 驱动程序 2.1 及更高版本指定 JDBC URL，而不包括原始用户名和密码。

相反，请指定 `jdbc:redshift:iam://` 方案。这将命令 JDBC 驱动程序使用您的 EMR Serverless 作业执行角色来自动获取凭证。有关更多信息，请参阅《Amazon Redshift 管理指南》中的配置 JDBC 或 ODBC 连接以使用 IAM 凭证。该 URL 的示例如下：

```
jdbc:redshift:iam://examplecluster.abc123xyz789.us-west-2.redshift.amazonaws.com:5439/
dev
```

当满足提供的条件时，您的作业执行角色需要以下权限：

权限	任务执行角色所需的条件
`redshift:GetClusterCredentials`	JDBC 驱动程序从 Amazon Redshift 获取凭证所需的权限
`redshift:DescribeCluster`	在 JDBC URL 中指定 Amazon Redshift 集群和 AWS 区域 而非端点所需的权限
`redshift-serverless:GetCredentials`	JDBC 驱动程序从 Amazon Redshift Serverless 获取凭证所需的权限
`redshift-serverless:GetWorkgroup`	使用 Amazon Redshift Serverless 并根据工作组名称和区域指定 URL 所需的权限

连接到不同 VPC 中的 Amazon Redshift

在 VPC 下设置预置的 Amazon Redshift 集群或 Amazon Redshift Serverless 工作组时，请为 Amazon EMR Serverless 应用程序配置 VPC 连接才能访问资源。有关如何在 EMR Serverless 应用程序上配置 VPC 连接的更多信息，请参阅 为 EMR Serverless 应用程序配置 VPC 访问权限以连接数据。

- 如果您预置的 Amazon Redshift 集群或 Amazon Redshift Serverless 工作组可公开访问，请在创建 EMR Serverless 应用程序时指定一个或多个附加了 NAT 网关的私有子网。
- 如果您预置的 Amazon Redshift 集群或 Amazon Redshift Serverless 工作组不可公开访问，则必须为 Amazon Redshift 集群创建 Amazon Redshift 托管 VPC 端点，如 为 EMR Serverless 应用程序配置 VPC 访问权限以连接数据 中所述。或者，您也可以按照《Amazon Redshift 管理指南》中连接到 Amazon Redshift Serverless 的说明创建 Amazon Redshift Serverless 工作组。必须将集群或子组关联到您在创建 EMR Serverless 应用程序时指定的私有子网。

> **Note**
>
> 如果您使用基于 IAM 的身份验证，并且 EMR Serverless 应用程序的私有子网未附加 NAT 网关，则还必须在这些子网上为 Amazon Redshift 或 Amazon Redshift Serverless 创建 VPC 端点。这样，JDBC 驱动程序就可以获取凭证。

在 Amazon Redshift 中进行读取和写入

以下代码示例用于使用 PySpark 数据源 API 和 sparkSQL 从 Amazon Redshift 数据库读取和写入示例数据。

Data source API

使用 PySpark 数据源 API 从 Amazon Redshift 数据库读取和写入示例数据。

```
import boto3
from pyspark.sql import SQLContext

sc = # existing SparkContext
sql_context = SQLContext(sc)

url = "jdbc:redshift:iam://redshifthost:5439/database"
aws_iam_role_arn = "arn:aws:iam::account-id:role/role-name"

df = sql_context.read \
    .format("io.github.spark_redshift_community.spark.redshift") \
    .option("url", url) \
    .option("dbtable", "table-name") \
    .option("tempdir", "s3://path/for/temp/data") \
    .option("aws_iam_role", "aws-iam-role-arn") \
```

```
    .load()

df.write \
    .format("io.github.spark_redshift_community.spark.redshift") \
    .option("url", url) \
    .option("dbtable", "table-name-copy") \
    .option("tempdir", "s3://path/for/temp/data") \
    .option("aws_iam_role", "aws-iam-role-arn") \
    .mode("error") \
    .save()
```

SparkSQL

PySpark 用于通过 sparkSQL 读取和写入亚马逊 Redshift 数据库的示例数据。

```
import boto3
import json
import sys
import os
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .enableHiveSupport() \
    .getOrCreate()

url = "jdbc:redshift:iam://redshifthost:5439/database"
aws_iam_role_arn = "arn:aws:iam::account-id:role/role-name"

bucket = "s3://path/for/temp/data"
tableName = "table-name" # Redshift table name

s = f"""CREATE TABLE IF NOT EXISTS {table-name} (country string, data string)
    USING io.github.spark_redshift_community.spark.redshift
    OPTIONS (dbtable '{table-name}', tempdir '{bucket}', url '{url}', aws_iam_role
 '{aws-iam-role-arn}' ); """

spark.sql(s)

columns = ["country" ,"data"]
data = [("test-country","test-data")]
df = spark.sparkContext.parallelize(data).toDF(columns)

# Insert data into table
```

```
df.write.insertInto(table-name, overwrite=False)
df = spark.sql(f"SELECT * FROM {table-name}")
df.show()
```

使用 Spark 连接器时的注意事项和限制

- 建议您为从 Spark on Amazon EMR 到 Amazon Redshift 的 JDBC 连接启用 SSL。
- 作为最佳实践，建议您在 AWS Secrets Manager 中管理 Amazon Redshift 集群的凭证。有关示例，请参阅使用 AWS Secrets Manager 检索连接至亚马逊 Redshift 的凭证。
- 建议使用参数 `aws_iam_role` 为 Amazon Redshift 身份验证参数传递 IAM 角色。
- 参数 `tempformat` 目前不支持 Parquet 格式。
- `tempdir` URI 指向 Amazon S3 位置。此临时目录不会自动清理，因此可能会增加额外成本。
- 请考虑以下针对 Amazon Redshift 的建议：
  - 建议阻止对 Amazon Redshift 集群的公有访问。
  - 建议启用 Amazon Redshift 审计日志记录。
  - 建议启用 Amazon Redshift 静态加密。
- 请考虑以下针对 Amazon S3 的建议：
  - 建议阻止对 Amazon S3 存储桶的公有访问。
  - 建议使用 Amazon S3 服务器端加密以加密使用的 Amazon S3 存储桶。
  - 建议使用 Amazon S3 生命周期策略定义 Amazon S3 存储桶的保留规则。
  - Amazon EMR 始终验证从开源导入到映像中的代码。出于安全原因，我们不支持从 Spark 到 Amazon S3 的以下身份验证方法：
    - 在`hadoop-env`配置分类中设置 AWS 访问密钥
    - 在 `tempdir` URI 中对 AWS 访问密钥进行编码

有关使用连接器及其支持参数的更多信息，请参阅以下资源：

- Amazon Redshift Management Guide（《Amazon Redshift 管理指南》）中的 Amazon Redshift integration for Apache Spark（适用于 Apache Spark 的 Amazon Redshift 集成）
- Github 上的 `spark-redshift` 社区存储库

使用 Amazon EMR Serverless 连接到 DynamoDB

在本教程中，您要将美国地名委员会的数据子集上传到 Amazon S3 存储桶，然后使用 Amazon EMR Serverless 上的 Hive 或 Spark 将数据复制到 Amazon DynamoDB 表进行查询。

步骤 1：将数据上传到 Amazon S3 存储桶

要创建 Amazon 存储桶，请按照《Amazon Simple Storage Service 控制台用户指南》中创建存储桶的说明操作。将对 *amzn-s3-demo-bucket* 的引用替换为新建存储桶的名称。现在，您的 EMR Serverless 应用程序已准备好运行作业。

1. 使用以下命令下载示例数据存档 `features.zip`。

```
wget https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
features.zip
```

2. 从存档中提取 `features.txt` 文件，访问文件的前几行：

```
unzip features.zip
head features.txt
```

结果应类似以下内容。

```
1535908|Big Run|Stream|WV|38.6370428|-80.8595469|794
875609|Constable Hook|Cape|NJ|40.657881|-74.0990309|7
1217998|Gooseberry Island|Island|RI|41.4534361|-71.3253284|10
26603|Boone Moore Spring|Spring|AZ|34.0895692|-111.410065|3681
1506738|Missouri Flat|Flat|WA|46.7634987|-117.0346113|2605
1181348|Minnow Run|Stream|PA|40.0820178|-79.3800349|1558
1288759|Hunting Creek|Stream|TN|36.343969|-83.8029682|1024
533060|Big Charles Bayou|Bay|LA|29.6046517|-91.9828654|0
829689|Greenwood Creek|Stream|NE|41.596086|-103.0499296|3671
541692|Button Willow Island|Island|LA|31.9579389|-93.0648847|98
```

这里每行中的字段表示唯一标识符、名称、自然特征类型、州、纬度（度）、经度（度）和高度（英尺）。

3. 将数据上传到 Amazon S3

```
aws s3 cp features.txt s3://amzn-s3-demo-bucket/features/
```

步骤 2：创建 Hive 表

使用 Apache Spark 或 Hive 创建一个新的 Hive 表，其中包含 Amazon S3 中上传的数据。

Spark

要使用 Spark 创建 Hive 表，请运行以下命令。

```
import org.apache.spark.sql.SparkSession

val sparkSession = SparkSession.builder().enableHiveSupport().getOrCreate()

sparkSession.sql("CREATE TABLE hive_features \
    (feature_id BIGINT, \
    feature_name STRING, \
    feature_class STRING, \
    state_alpha STRING, \
    prim_lat_dec DOUBLE, \
    prim_long_dec DOUBLE, \
    elev_in_ft BIGINT) \
    ROW FORMAT DELIMITED \
    FIELDS TERMINATED BY '|' \
    LINES TERMINATED BY '\n' \
    LOCATION 's3://amzn-s3-demo-bucket/features';")
```

现在，您有一个 Hive 表，其中包含 `features.txt` 文件中的数据。要验证数据是否在表中，请运行 Spark SQL 查询，如以下示例所示。

```
sparkSession.sql(
    "SELECT state_alpha, COUNT(*) FROM hive_features GROUP BY state_alpha;")
```

Hive

要使用 Hive 创建 Hive 表，请运行以下命令。

```
CREATE TABLE hive_features
    (feature_id             BIGINT,
    feature_name            STRING ,
    feature_class           STRING ,
    state_alpha             STRING,
    prim_lat_dec            DOUBLE ,
```

```
    prim_long_dec           DOUBLE ,
    elev_in_ft              BIGINT)
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY '|'
    LINES TERMINATED BY '\n'
    LOCATION 's3://amzn-s3-demo-bucket/features';
```

现在，您有一个 Hive 表，其中包含 `features.txt` 文件中的数据。要验证数据是否在表中，请运行 HiveQL 查询，如以下示例所示。

```
SELECT state_alpha, COUNT(*) FROM hive_features GROUP BY state_alpha;
```

步骤 3：将数据复制到 DynamoDB

使用 Spark 或 Hive 将数据复制到新的 DynamoDB 表。

Spark

要将数据从您在上一步中创建的 Hive 表复制到 DynamoDB，请按照将数据复制到 DynamoDB 中的步骤 1-3 操作。这将创建一个名为 `Features` 的新 DynamoDB 表。然后，您可以直接从文本文件中读取数据，并将其复制到 DynamoDB 表中，如下例所示。

```
import com.amazonaws.services.dynamodbv2.model.AttributeValue
import org.apache.hadoop.dynamodb.DynamoDBItemWritable
import org.apache.hadoop.dynamodb.read.DynamoDBInputFormat
import org.apache.hadoop.io.Text
import org.apache.hadoop.mapred.JobConf
import org.apache.spark.SparkContext

import scala.collection.JavaConverters._

object EmrServerlessDynamoDbTest {

    def main(args: Array[String]): Unit = {

        jobConf.set("dynamodb.input.tableName", "Features")
        jobConf.set("dynamodb.output.tableName", "Features")
        jobConf.set("dynamodb.region", "region")

        jobConf.set("mapred.output.format.class",
 "org.apache.hadoop.dynamodb.write.DynamoDBOutputFormat")
```

```
        jobConf.set("mapred.input.format.class",
 "org.apache.hadoop.dynamodb.read.DynamoDBInputFormat")

        val rdd = sc.textFile("s3://amzn-s3-demo-bucket/ddb-connector/")
            .map(row => {
                val line = row.split("\\|")
                val item = new DynamoDBItemWritable()

                val elevInFt = if (line.length > 6) {
                    new AttributeValue().withN(line(6))
                } else {
                    new AttributeValue().withNULL(true)
                }

                item.setItem(Map(
                    "feature_id" -> new AttributeValue().withN(line(0)),
                    "feature_name" -> new AttributeValue(line(1)),
                    "feature_class" -> new AttributeValue(line(2)),
                    "state_alpha" -> new AttributeValue(line(3)),
                    "prim_lat_dec" -> new AttributeValue().withN(line(4)),
                    "prim_long_dec" -> new AttributeValue().withN(line(5)),
                    "elev_in_ft" -> elevInFt)
                    .asJava)
                (new Text(""), item)
        })
        rdd.saveAsHadoopDataset(jobConf)
    }
}
```

Hive

要将数据从您在上一步中创建的 Hive 表复制到 DynamoDB，请按照将数据复制到 DynamoDB 中的说明操作。

步骤 4：从 DynamoDB 查询数据

使用 Spark 或 Hive 查询 DynamoDB 表。

Spark

要查询您在上一步中创建的 DynamoDB 表中的数据，请使用 Spark SQL 或 Spark API。

MapReduce

Example使用 Spark SQL 查询 DynamoDB 表

以下 Spark SQL 查询将按字母顺序返回所有特征类型的列表。

```
val dataFrame = sparkSession.sql("SELECT DISTINCT feature_class \
    FROM ddb_features \
    ORDER BY feature_class;")
```

以下 Spark SQL 查询将返回以字母 M 开头的所有数据湖列表。

```
val dataFrame = sparkSession.sql("SELECT feature_name, state_alpha \
    FROM ddb_features \
    WHERE feature_class = 'Lake' \
    AND feature_name LIKE 'M%' \
    ORDER BY feature_name;")
```

以下 Spark SQL 查询将返回所有州的列表，其中至少有三个特征高于一英里。

```
val dataFrame = sparkSession.dql("SELECT state_alpha, feature_class, COUNT(*) \
    FROM ddb_features \
    WHERE elev_in_ft > 5280 \
    GROUP by state_alpha, feature_class \
    HAVING COUNT(*) >= 3 \
    ORDER BY state_alpha, feature_class;")
```

Example— 使用 Spark API 查询你的 DynamoDB 表 MapReduce

以下 MapReduce 查询按字母顺序返回所有要素类型的列表。

```
val df = sc.hadoopRDD(jobConf, classOf[DynamoDBInputFormat], classOf[Text],
 classOf[DynamoDBItemWritable])
    .map(pair => (pair._1, pair._2.getItem))
    .map(pair => pair._2.get("feature_class").getS)
    .distinct()
    .sortBy(value => value)
    .toDF("feature_class")
```

以下 MapReduce 查询返回以字母 M 开头的所有湖泊的列表。

```
val df = sc.hadoopRDD(jobConf, classOf[DynamoDBInputFormat], classOf[Text],
 classOf[DynamoDBItemWritable])
    .map(pair => (pair._1, pair._2.getItem))
```

```
    .filter(pair => "Lake".equals(pair._2.get("feature_class").getS))
    .filter(pair => pair._2.get("feature_name").getS.startsWith("M"))
    .map(pair => (pair._2.get("feature_name").getS,
 pair._2.get("state_alpha").getS))
    .sortBy(_._1)
    .toDF("feature_name", "state_alpha")
```

以下 MapReduce 查询返回包含至少三个高于一英里的要素的所有州的列表。

```
val df = sc.hadoopRDD(jobConf, classOf[DynamoDBInputFormat], classOf[Text],
 classOf[DynamoDBItemWritable])
    .map(pair => pair._2.getItem)
    .filter(pair => pair.get("elev_in_ft").getN != null)
    .filter(pair => Integer.parseInt(pair.get("elev_in_ft").getN) > 5280)
    .groupBy(pair => (pair.get("state_alpha").getS, pair.get("feature_class").getS))
    .filter(pair => pair._2.size >= 3)
    .map(pair => (pair._1._1, pair._1._2, pair._2.size))
    .sortBy(pair => (pair._1, pair._2))
    .toDF("state_alpha", "feature_class", "count")
```

Hive

要从您在上一步中创建的 DynamoDB 表查询数据，请按照查询 DynamoDB 表中的数据中的说明操作。

设置跨账户访问

要为 Amazon EMR Serverless 设置跨账户访问权限，请完成以下步骤。在示例中，`AccountA` 是您创建 Amazon EMR Serverless 应用程序的账户，`AccountB` 是您的 Amazon DynamoDB 所在的账户。

1. 在 `AccountB` 中创建 DynamoDB 表。有关更多信息，请参阅步骤 1：创建表。
2. 在 `AccountB` 中创建一个可访问 DynamoDB 表的 `Cross-Account-Role-B` IAM 角色。
   a. 登录 AWS 管理控制台 并打开 IAM 控制台，网址为https://console.aws.amazon.com/iam/。
   b. 选择角色，然后创建一个名为 `Cross-Account-Role-B` 的新角色。有关如何创建 IAM 角色的更多信息，请参阅用户指南中的创建 IAM 角色。
   c. 创建一个 IAM 策略，授予访问跨账户 DynamoDB 表的权限。然后将 IAM policy 附加到 `Cross-Account-Role-B`。

      以下是授予对 DynamoDB 表 `CrossAccountTable` 的访问权限的策略。

d. 编辑 Cross-Account-Role-B 角色的信任关系。

要为角色配置信任关系，请在 IAM 控制台中为在步骤 2: 中创建的角色选择信任关系选项卡 Cross-Account-Role-B。

选择编辑信任关系，然后添加以下策略文档。本文档允许 AccountA 中的 Job-Execution-Role-A 代入此 Cross-Account-Role-B 角色。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole"
      ],
      "Resource": "arn:aws:iam::123456789012:role/Job-Execution-Role-A",
      "Sid": "AllowSTSAssumerole"
    }
  ]
}
```

e. 授予 AccountA 中的 Job-Execution-Role-A 代入 Cross-Account-Role-B 的 - STS Assume role 权限。

在的 IAM 控制台中 AWS 账户 AccountA，选择Job-Execution-Role-A。添加以下 Job-Execution-Role-A 策略语句以便对 Cross-Account-Role-B 角色执行 AssumeRole 操作。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole"
```

```
      ],
      "Resource": [
        "arn:aws:iam::123456789012:role/Cross-Account-Role-B"
      ],
      "Sid": "AllowSTSAssumerole"
    }
  ]
}
```

f. 在核心站点分类中将 `dynamodb.customAWSCredentialsProvider` 属性的值设置为 `com.amazonaws.emr.AssumeRoleAWSCredentialsProvider`。使用 `Cross-Account-Role-B` 的 ARN 值设置环境变量 `ASSUME_ROLE_CREDENTIALS_ROLE_ARN`。

3. 使用 `Job-Execution-Role-A` 运行 Spark 或 Hive 作业。

注意事项

当您将 DynamoDB 连接器与 Apache Spark 或 Apache Hive 结合使用时，请注意这些行为和限制。

将 DynamoDB 连接器与 Apache Spark 结合使用时的注意事项

- Spark SQL 不支持使用存储处理程序选项创建 Hive 表。有关更多信息，请参阅 Apache Spark 文档中的指定 Hive 表的存储格式。
- Spark SQL 不支持使用存储处理程序执行 `STORED BY` 操作。如果您想通过外部 Hive 表与 DynamoDB 表交互，请先使用 Hive 创建该表。
- 要将查询转换为 DynamoDB 查询，DynamoDB 连接器会使用谓词下推。谓词下推按映射到 DynamoDB 表分区键的列筛选数据。谓词下推仅在您将连接器与 Spark SQL 配合使用时才起作用，而不是 API。 MapReduce

将 DynamoDB 连接器与 Apache Hive 结合使用时的注意事项

调整 mapper 的最大数量

- 如果使用 `SELECT` 查询从映射到 DynamoDB 的外部 Hive 表读取数据，EMR Serverless 上的映射任务数将按照为 DynamoDB 表配置的总读取吞吐量除以每个映射任务的吞吐量计算。每个映射任务的默认吞吐量为 100。
- Hive 作业可以使用超出每个 EMR Serverless 应用程序配置最大容器数的映射任务数，具体取决于为 DynamoDB 配置的读取吞吐量。此外，长时间运行 Hive 查询会占用 DynamoDB 表的所有预置读取容量。这会对其他用户产生负面影响。

- 您可以使用 dynamodb.max.map.tasks 属性设置映射任务的上限。您还可以使用此属性，根据任务容器大小调整每个映射任务读取的数据量。
- 您可以在 Hive 查询级别或 start-job-run 命令的 hive-site 分类中设置 dynamodb.max.map.tasks 属性。此值必须等于或大于 1。当 Hive 处理查询时，生成的 Hive 作业使用的 dynamodb.max.map.tasks 值不会超过从 DynamoDB 表读取时的值。

调整每个任务的写入吞吐量

- EMR Serverless 上每个任务的写入吞吐量是根据为 DynamoDB 表配置的总写入吞吐量除以 mapreduce.job.maps 属性值来计算的。对于 Hive，此属性的默认值为 2。因此，Hive 作业最后阶段的前两个任务可能会使用所有的写入吞吐量。这将导致同一作业或其他作业中其他任务的写入节流。
- 为避免写入节流，请根据最后阶段的任务数或希望为每个任务分配的写入吞吐量来设置 mapreduce.job.maps 属性值。在 EMR Serverless 上 start-job-run 命令的 mapred-site 分类中设置此属性。

安全性

云安全 AWS 是重中之重。作为 AWS 客户，您可以受益于专为满足大多数安全敏感型组织的要求而构建的数据中心和网络架构。

安全是双方共同承担 AWS 的责任。责任共担模式将其描述为云的 安全性和云中 的安全性：

- 云安全 — AWS 负责保护在 AWS 云中运行 AWS 服务的基础架构。 AWS 还为您提供安全使用的服务。作为 AWS 合规性计划的一部分，第三方审核人员将定期测试和验证安全性的有效性。要了解适用于 Amazon EMR Serverless 的合规性计划，请参阅 AWS 按合规性计划提供的范围内服务。
- 云端安全-您的责任由您使用的 AWS 服务决定。您还需要对其它因素负责，包括您的数据的敏感性、您公司的要求以及适用的法律法规。

本文档有助于您了解在使用 Amazon EMR Serverless 时如何应用责任共担模式。这些主题说明了如何配置 Amazon EMR Serverless 以及如何使用其他 AWS 服务来实现您的安全和合规目标。

主题

Amazon EMR Serverless 的安全最佳实践

Amazon EMR Serverless 提供了多项安全功能，供您在开发和实施自己的安全策略时考虑。以下最佳实践是一般指导原则，并不代表完整安全解决方案。这些最佳实践可能不适合环境或不满足环境要求，请将其视为有用的考虑因素而不是惯例。

采用最低权限原则

EMR Serverless 为使用 IAM 角色（如执行角色）的应用程序提供了精细的访问策略。我们建议向执行角色仅授予任务所需的最低权限集，例如覆盖应用程序和对日志目标的访问权限。我们还建议定期以及在应用程序代码发生变化时审核任务的权限。

隔离不受信任的应用程序代码

EMR Serverless 在属于不同 EMR Serverless 应用程序的作业之间建立了完全隔离的网络。如果需要作业级别隔离，请考虑将作业隔离到不同的 EMR Serverless 应用程序中。

基于角色的访问控制 (RBAC) 权限

管理员应严格控制 EMR Serverless 应用程序的基于角色的访问控制（RBAC）权限。

数据保护

责任 AWS 共担模式适用于 Amazon EMR Serverless 中的数据保护。如本模型所述 AWS ，负责保护运行所有 AWS 云的全球基础架构。您负责维护对托管在此基础结构上的内容的控制。此内容包括您使用的 AWS 服务的安全配置和管理任务。有关数据隐私的更多信息，请参阅数据隐私常见问题。有关欧洲数据保护的信息，请参阅 AWS 安全博客上的 AWS 分担责任模型和GDPR 博客文章。

出于数据保护目的，我们建议您使用 AWS 身份和访问管理 (IAM) 和 IAM保护 AWS 账户凭证并设置个人账户。这仅向每个用户授予履行其工作职责所需的权限。我们还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证（MFA）。
- 用于 SSL/TLS 与 AWS 资源通信。建议使用 TLS 1.2 或更高版本。
- 使用设置 API 和用户活动日志 AWS CloudTrail。
- 使用 AWS 加密解决方案以及 AWS 服务中的所有默认安全控制。
- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的个人数据。

- 使用 Amazon EMR Serverless 加密选项对静态数据和传输中数据进行加密。
- 如果您在 AWS 通过命令行界面或 API 进行访问时需要经过 FIPS 140-2 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅美国联邦信息处理标准 (FIPS) 第 140-2 版。

我们强烈建议您切勿将敏感的可识别信息（例如您客户的账号）放入自由格式字段（例如名称字段）。这包括您使用控制台、API 或使用 Amazon EMR Serverless 或其他 AWS 服务时。 AWS CLI AWS SDKs您输入 Amazon EMR Serverless 或其他服务的任何数据都可能被采集到诊断日志中。当您向外部服务器提供网址时，请勿在网址中包含凭证信息来验证您对该服务器的请求。

静态加密

数据加密有助于防止未经授权的用户在集群和关联的数据存储系统中读取数据。这包括保存到持久性媒体的数据（称为静态数据）和在网络中传输时可能被拦截的数据（称为传输中的数据）。

数据加密需要密钥和凭证。您可以从多个选项中进行选择，包括由管理的密钥 AWS Key Management Service、由 Amazon S3 管理的密钥以及您提供的自定义提供商提供的密钥和证书。 AWS KMS 用作密钥提供商时，加密密钥的存储和使用将产生费用。有关更多信息，请参阅 AWS KMS 定价。

在指定加密选项前，确定要使用的密钥和凭证管理系统。然后，针对您指定为加密设置一部分的自定义提供程序，来创建密钥和凭证。

Amazon S3 中 EMRFS 数据的静态加密

每个 EMR Serverless 应用程序都使用特定的发行版，其中包括 EMRFS（EMR 文件系统）。Amazon S3 加密适用于在 Amazon S3 中读取和写入的 EMR 文件系统（EMRFS）对象。启用静态加密时，您可以指定 Amazon S3 服务器端加密（SSE）或客户端加密（CSE）作为默认加密模式。（可选）使用每存储桶加密覆盖为单个存储桶指定不同的加密方法。无论是否启用了 Amazon S3 加密，传输层安全性（TLS）都会对 EMR 集群节点和 Amazon S3 之间正在传输的 EMRFS 对象进行加密。如果使用带有客户管理密钥的 Amazon S3 CSE，则在 EMR Serverless 应用程序中运行作业的执行角色必须有权访问密钥。有关 Amazon S3 加密的详细信息，请参阅《Amazon Simple Storage Service 开发人员指南》中的使用加密保护数据。

> **Note**
>
> 使用时 AWS KMS，会收取加密密钥的存储和使用费用。有关更多信息，请参阅 AWS KMS 定价。

Amazon S3 服务器端加密

默认情况下，所有 Amazon S3 存储桶都配置了加密，所有上传到 S3 存储桶的新对象都会自动静态加密，Amazon S3 在向磁盘写入数据时会在对象级别对数据进行加密，并在访问数据时对其进行解密。有关 SSE 的更多信息，请参阅《Amazon Simple Storage Service 开发人员指南》中的使用服务器端加密保护数据。

在 Amazon EMR Serverless 中指定 SSE 时，可选择两个不同的密钥管理系统：

- SSE-S3 - Amazon S3 为您管理密钥。无需在 EMR Serverless 上进行额外设置。
- SSE-KMS-您可以使用 AWS KMS key 来设置适用于 EMR Serverless 的策略。无需在 EMR Serverless 上进行额外设置。

要对写入 Amazon S3 的数据使用 AWS KMS 加密，在使用 `StartJobRun` API 时有两种选择。您可以对写入 Amazon S3 的所有内容启用加密，也可以对写入特定存储桶的数据启用加密。有关 `StartJobRun` API 的更多信息，请参阅 EMR Serverless API 参考。

要对写入 Amazon S3 的所有数据启用 AWS KMS 加密，请在调用 `StartJobRun` API 时使用以下命令。

```
--conf spark.hadoop.fs.s3.enableServerSideEncryption=true
--conf spark.hadoop.fs.s3.serverSideEncryption.kms.keyId=<kms_id>
```

要对写入特定存储桶的数据启用 AWS KMS 加密，请在调用 `StartJobRun` API 时使用以下命令。

```
--conf spark.hadoop.fs.s3.bucket.<amzn-s3-demo-bucket1>.enableServerSideEncryption=true
--conf spark.hadoop.fs.s3.bucket.<amzn-s3-demo-
bucket1>.serverSideEncryption.kms.keyId=<kms-id>
```

使用客户提供密钥的 SSE（SSE-C）不能与 EMR Serverless 一起使用。

Amazon S3 客户端加密

对于 Amazon S3 客户端加密，在每个 Amazon EMR 发行版上提供的 EMRFS 客户端中进行 Amazon S3 加密和解密。在对象上载到 Amazon S3 之前对其进行加密，并在下载后对其进行解密。您指定的提供程序会提供客户端使用的加密密钥。客户端可以使用 AWS KMS 提供的密钥（CSE-KMS）或提供客户端根密钥（CSE-C）的自定义 Java 类。CSE-KMS 和 CSE-C 之间的加密细节略有不同，具体取决于指定的提供程序以及正在解密或加密对象的元数据。如果使用带有客户管理密钥的 Amazon S3 CSE，则在 EMR Serverless 应用程序中运行作业的执行角色必须有权访问密钥。可能会收取额外的

KMS 费用。有关这些差异的更多信息，请参阅《Amazon Simple Storage Service 开发人员指南》中的使用客户端加密保护数据。

本地磁盘加密

临时存储中的数据使用行业标准 AES-256 加密算法通过服务自有的密钥加密。

密钥管理

您可以将 KMS 配置为自动轮换 KMS 密钥。这会每年转动一次密钥，同时无限期地保存旧密钥，以便您的数据仍然可以被解密。有关更多信息，请参阅 Rotating customer-managed keys。

传输中加密

Amazon EMR Serverless 提供以下特定于应用程序的加密功能：

- Spark
  - 默认情况下，Spark 驱动程序和执行程序之间的通信是经过身份验证的内部通信。驱动程序和执行程序之间的 RPC 通信是加密的。
- Hive
  - AWS Glue 元数据仓和 EMR 无服务器应用程序之间的通信通过 TLS 进行。

您应该仅允许使用 Amazon S3 存储桶 IAM 策略上的 a ws: SecureTransport 条件通过 HTTPS (TLS) 进行加密连接。

Amazon EMR Serverless 中的 Identity and Access Management（IAM）

AWS Identity and Access Management (IAM) AWS 服务 可帮助管理员安全地控制对 AWS 资源的访问权限。IAM 管理员控制谁可以通过身份验证（登录）和授权（拥有权限）来使用 Amazon EMR Serverless 资源。您可以使用 IAM AWS 服务 ，无需支付额外费用。

主题

- 受众
- 使用身份进行身份验证
- 使用策略管理访问

- EMR Serverless 如何与 IAM 结合使用
- 使用 EMR Serverless 的服务相关角色
- Amazon EMR Serverless 的作业运行时角色
- EMR Serverless 的用户访问策略示例
- 基于标签的访问控制策略
- EMR Serverless 基于身份的策略示例
- Amazon EMR 托管策略的无服务器更新 AWS
- Amazon EMR Serverless 身份和访问问题排查

受众

您的使用方式 AWS Identity and Access Management (IAM) 因您的角色而异：

- 服务用户：如果您无法访问功能，请从管理员处请求权限（请参阅Amazon EMR Serverless 身份和访问问题排查）
- 服务管理员：确定用户访问权限并提交权限请求（请参阅Amazon EMR Serverless 中的 Identity and Access Management （IAM））
- IAM 管理员：编写用于管理访问权限的策略（请参阅EMR Serverless 基于身份的策略示例）

使用身份进行身份验证

身份验证是您 AWS 使用身份凭证登录的方式。您必须以 IAM 用户身份进行身份验证 AWS 账户根用户，或者通过担任 IAM 角色进行身份验证。

您可以使用来自身份源的证书 AWS IAM Identity Center （例如（IAM Identity Center）、单点登录身份验证或 Google/Facebook 证书，以联合身份登录。有关登录的更多信息，请参阅《AWS 登录 用户指南》中的如何登录您的 AWS 账户。

对于编程访问， AWS 提供 SDK 和 CLI 来对请求进行加密签名。有关更多信息，请参阅《IAM 用户指南》中的适用于 API 请求的AWS 签名版本 4。

AWS 账户 root 用户

创建时 AWS 账户，首先会有一个名为 AWS 账户 root 用户的登录身份，该身份可以完全访问所有资源 AWS 服务 和资源。我们强烈建议不要使用根用户进行日常任务。有关需要根用户凭证的任务，请参阅《IAM 用户指南》中的需要根用户凭证的任务。

联合身份

作为最佳实践，要求人类用户使用与身份提供商的联合身份验证才能 AWS 服务 使用临时证书进行访问。

联合身份是指来自您的企业目录、Web 身份提供商的用户 Directory Service ，或者 AWS 服务 使用来自身份源的凭据进行访问的用户。联合身份代入可提供临时凭证的角色。

要集中管理访问权限，建议使用。 AWS IAM Identity Center有关更多信息，请参阅《AWS IAM Identity Center 用户指南》中的什么是 IAM Identity Center？。

IAM 用户和群组

IAM 用户是对某个人员或应用程序具有特定权限的一个身份。建议使用临时凭证，而非具有长期凭证的 IAM 用户。有关更多信息，请参阅 IAM 用户指南中的要求人类用户使用身份提供商的联合身份验证才能 AWS 使用临时证书进行访问。

IAM 组指定一组 IAM 用户，便于更轻松地对大量用户进行权限管理。有关更多信息，请参阅《IAM 用户指南》中的 IAM 用户使用案例。

IAM 角色

IAM 角色是具有特定权限的身份，可提供临时凭证。您可以通过从用户切换到 IAM 角色（控制台）或调用 AWS CLI 或 AWS API 操作来代入角色。有关更多信息，请参阅《IAM 用户指南》中的担任角色的方法。

IAM 角色对于联合用户访问、临时 IAM 用户权限、跨账户访问、跨服务访问以及在 Amazon EC2 上运行的应用程序非常有用。有关更多信息，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

使用策略管理访问

您可以 AWS 通过创建策略并将其附加到 AWS 身份或资源来控制中的访问权限。策略定义了与身份或资源关联时的权限。 AWS 在委托人提出请求时评估这些政策。大多数策略都以 JSON 文档的 AWS 形式存储在中。有关 JSON 策略文档的更多信息，请参阅《IAM 用户指南》中的 JSON 策略概述。

管理员使用策略，通过定义哪个主体可以在什么条件下对哪些资源执行哪些操作来指定谁有权访问什么。

默认情况下，用户和角色没有权限。IAM 管理员创建 IAM 策略并将其添加到角色中，然后用户可以担任这些角色。IAM 策略定义权限，与执行操作所用的方法无关。

基于身份的策略

基于身份的策略是您附加到身份（用户、组或角色）的 JSON 权限策略文档。这些策略控制身份可以执行什么操作、对哪些资源执行以及在什么条件下执行。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的使用客户管理型策略定义自定义 IAM 权限。

基于身份的策略可以是内联策略（直接嵌入到单个身份中）或托管策略（附加到多个身份的独立策略）。要了解如何在托管策略和内联策略之间进行选择，请参阅《IAM 用户指南》中的在托管策略与内联策略之间进行选择。

基于资源的策略

基于资源的策略是附加到资源的 JSON 策略文档。示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。您必须在基于资源的策略中指定主体。

基于资源的策略是位于该服务中的内联策略。您不能在基于资源的策略中使用 IAM 中的 AWS 托管策略。

其他策略类型

AWS 支持其他策略类型，这些策略类型可以设置更常见的策略类型授予的最大权限：

- 权限边界 – 设置基于身份的策略可以授予 IAM 实体的最大权限。有关更多信息，请参阅《 IAM 用户指南》中的 IAM 实体的权限边界。
- 服务控制策略 (SCPs)-在中指定组织或组织单位的最大权限 AWS Organizations。有关更多信息，请参阅《AWS Organizations 用户指南》中的服务控制策略。
- 资源控制策略 (RCPs)-设置账户中资源的最大可用权限。有关更多信息，请参阅《AWS Organizations 用户指南》中的资源控制策略 (RCPs)。
- 会话策略 – 在为角色或联合用户创建临时会话时，作为参数传递的高级策略。有关更多信息，请参阅《IAM 用户指南》中的会话策略。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解在涉及多种策略类型时如何 AWS 确定是否允许请求，请参阅 IAM 用户指南中的策略评估逻辑。

EMR Serverless 如何与 IAM 结合使用

在使用 IAM 管理对 Amazon EMR Serverless 的访问之前，请了解哪些 IAM 功能可用于 Amazon EMR Serverless。

与 EMR Serverless 结合使用的 IAM 功能

IAM 功能	Amazon EMR Serverless 支持
基于身份的策略	是
基于资源的策略	否
策略操作	是
策略资源	是
策略条件密钥	否
ACLs	否
ABAC（策略中的标签）	是
临时凭证	是
主体权限	是
服务角色	否
服务关联角色	是

要全面了解 EMR Serverless 和其他 AWS 服务如何与大多数 IAM 功能配合使用，请参阅 IAM 用户指南中与 IAM 配合使用的AWS 服务。

EMR Serverless 基于身份的策略

支持基于身份的策略：是

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的使用客户管理型策略定义自定义 IAM 权限。

通过使用 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源以及允许或拒绝操作的条件。要了解可在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》中的 IAM JSON 策略元素引用。

EMR Serverless 基于身份的策略示例

要访问 Amazon EMR Serverless 基于身份的策略示例，请参阅 EMR Serverless 基于身份的策略示例。

EMR Serverless 中基于资源的策略

支持基于资源的策略：否

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

要启用跨账户访问，您可以将整个账户或其他账户中的 IAM 实体指定为基于资源的策略中的主体。有关更多信息，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

EMR Serverless 的策略操作

支持策略操作：是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

JSON 策略的 `Action` 元素描述可用于在策略中允许或拒绝访问的操作。在策略中包含操作以授予执行关联操作的权限。

要查看 EMR Serverless 操作的列表，请参阅《服务授权参考》中 Amazon EMR Serverless 的操作、资源和条件键。

EMR Serverless 中的策略操作在操作前使用以下前缀。

```
emr-serverless
```

要在单个语句中指定多项操作，请使用逗号将它们隔开。

```
"Action": [
      "emr-serverless:action1",
      "emr-serverless:action2"
         ]
```

要访问 Amazon EMR Serverless 基于身份的策略示例，请参阅 EMR Serverless 基于身份的策略示例。

EMR Serverless 的策略资源

支持策略资源：是

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

`Resource` JSON 策略元素指定要向其应用操作的一个或多个对象。作为最佳实践，请使用其 Amazon 资源名称（ARN）指定资源。对于不支持资源级权限的操作，请使用通配符 (*) 指示语句应用于所有资源。

```
"Resource": "*"
```

要参阅 Amazon EMR 无服务器资源类型及其 ARNs列表，请参阅《服务授权参考》中的 Amazon EMR Serverless 定义的资源。要了解哪些操作指定每个资源的 ARN，请参阅 Amazon EMR Serverless 的操作、资源和条件键。

要访问 Amazon EMR Serverless 基于身份的策略示例，请参阅 EMR Serverless 基于身份的策略示例。

EMR Serverless 的策略条件键

策略条件键支持

支持特定于服务的策略条件密钥	否

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

Condition 元素根据定义的条件指定语句何时执行。您可以创建使用条件运算符（例如，等于或小于）的条件表达式，以使策略中的条件与请求中的值相匹配。要查看所有 AWS 全局条件键，请参阅 IAM 用户指南中的AWS 全局条件上下文密钥。

要查看 Amazon EMR Serverless 条件键的列表，并了解可以使用条件键的操作和资源，请参阅《服务授权参考》中 Amazon EMR Serveress 的操作、资源和条件键。

所有 Amazon EC2 操作都支持 aws:RequestedRegion 和 ec2:Region 条件键。有关更多信息，请参阅示例：限制对特定区域的访问。

EMR Serverless 中的访问控制列表 (ACLs)

支持 ACLs：否

访问控制列表 (ACLs) 控制哪些委托人（账户成员、用户或角色）有权访问资源。 ACLs 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

EMR Serverless 基于属性的访问权限控制（ABAC）

基于属性的访问权限控制（ABAC）支持

支持 ABAC（策略中的标签）	是

基于属性的访问权限控制（ABAC）是一种授权策略，该策略基于称为标签的属性来定义权限。您可以将标签附加到 IAM 实体和 AWS 资源，然后设计 ABAC 策略以允许在委托人的标签与资源上的标签匹配时进行操作。

要基于标签控制访问，您需要使用 aws:ResourceTag/*key-name*aws:RequestTag/*key-name* 或 aws:TagKeys 条件键在策略的条件元素中提供标签信息。

如果某个服务对于每种资源类型都支持所有这三个条件键，则对于该服务，该值为是。如果某个服务仅对于部分资源类型支持所有这三个条件键，则该值为部分。

有关 ABAC 的更多信息，请参阅《IAM 用户指南》中的使用 ABAC 授权定义权限。要查看设置 ABAC 步骤的教程，请参阅《IAM 用户指南》中的使用基于属性的访问权限控制（ABAC）。

在 EMR Serverless 中使用临时凭证

支持临时凭证：是

临时证书提供对 AWS 资源的短期访问权限，并且是在您使用联合身份或切换角色时自动创建的。AWS 建议您动态生成临时证书，而不是使用长期访问密钥。有关更多信息，请参阅《IAM 用户指南》中的 IAM 中的临时安全凭证和使用 IAM 的。AWS 服务

EMR Serverless 的跨服务主体权限

支持转发访问会话（FAS）：是

转发访问会话 (FAS) 使用调用主体的权限 AWS 服务，再加上 AWS 服务 向下游服务发出请求的请求。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。

EMR Serverless 的服务角色

支持服务角色	否

EEMR Serverless 的服务相关角色

支持服务相关角色	是

有关创建或管理服务相关角色的详细信息，请参阅能够与 IAM 搭配使用的AWS 服务。在表中查找服务相关角色列中包含 Yes 的表。选择是链接以访问该服务的服务相关角色文档。

使用 EMR Serverless 的服务相关角色

Amazon EMR 无服务器使用 AWS Identity and Access Management (IAM) 服务相关角色。服务相关角色是一种独特的 IAM 角色类型，直接关联到 EMR Serverless。服务相关角色由 EMR Serverless 预定义，包括该服务代表您调用 AWS 其他服务所需的所有权限。

服务相关角色使设置 EMR Serverless 变得更加容易，因为您无需手动添加必要的权限。EMR Serverless 定义了其服务相关角色的权限，除非另有定义，否则只有 EMR Serverless 可以代入其角色。定义的权限包括信任策略和权限策略，以及不能附加到任何其他 IAM 实体的权限策略。

只有在首先删除相关资源后，您才能删除服务关联角色。这可以保护您的 EMR Serverless 资源，因为您不会无意中删除对资源的访问权限。

有关支持服务相关角色的其他服务的信息，请参阅与 IAM 配合使用的AWS 服务，并检查服务相关角色列中显示为是的服务。选择是和链接，访问该服务的服务相关角色文档。

EMR Serverless 的服务相关角色权限

EMR Serverless 使用名为的服务相关角色使其AWSServiceRoleForAmazonEMRServerless能够代表您进行呼叫 AWS APIs 。

AWSServiceRoleForAmazonEMRServerless 服务相关角色信任以下服务来代入该角色：

- `ops.emr-serverless.amazonaws.com`

名为 `AmazonEMRServerlessServiceRolePolicy` 的角色权限策略允许 EMR Serverless 对指定资源完成以下操作。

> **Note**
>
> 托管策略内容发生变化，此处显示的策略可能已过时。在 Amazon EMRServerless ServiceRolePolicy 中查看 up-to-date政策最多的内容 AWS 管理控制台。

- 操作：`ec2:CreateNetworkInterface`
- 操作：`ec2:DeleteNetworkInterface`
- 操作：`ec2:DescribeNetworkInterfaces`
- 操作：`ec2:DescribeSecurityGroups`
- 操作：`ec2:DescribeSubnets`
- 操作：`ec2:DescribeVpcs`
- 操作：`ec2:DescribeDhcpOptions`
- 操作：`ec2:DescribeRouteTables`
- 操作：`cloudwatch:PutMetricData`

以下是完整的 `AmazonEMRServerlessServiceRolePolicy` 策略。

JSON

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

```
      "Sid": "EC2PolicyStatement",
      "Effect": "Allow",
      "Action": [
        "ec2:CreateNetworkInterface",
        "ec2:DeleteNetworkInterface",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcs",
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeRouteTables"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "CloudWatchPolicyStatement",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringEquals": {
          "cloudwatch:namespace": [
            "AWS/EMRServerless",
            "AWS/Usage"
          ]
        }
      }
    }
  ]
}
```

以下信任策略附加到此角色，允许 EMR Serverless 主体代入此角色。

JSON

```
{
```

```
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole"
      ],
      "Resource": "arn:aws:iam::123456789012:role/aws-service-role/emr-
serverless.amazonaws.com/AWSServiceRoleForEMRServerless",
      "Sid": "AllowSTSAssumerole"
    }
  ]
}
```

您必须配置权限，允许 IAM 实体（如用户、组或角色）创建、编辑或删除服务关联角色。有关更多信息，请参阅《IAM 用户指南》中的服务相关角色权限。

为 EMR Serverless 创建服务相关角色

您无需手动创建服务关联角色。当您在（ AWS 管理控制台 使用 EMR Studio）、或 API AWS 中创建新的 EMR 无服务器应用程序时，EMR Serverless 会为您创建服务相关角色。 AWS CLI您必须配置权限，允许 IAM 实体（如用户、组或角色）创建、编辑或删除服务关联角色。

使用 IAM 创建 AWSServiceRoleForAmazonEMRServerless 服务相关角色

将以下语句添加到需要创建服务相关角色的 IAM 实体的权限策略。

```
{
    "Effect": "Allow",
    "Action": [
        "iam:CreateServiceLinkedRole"
    ],
    "Resource": "arn:aws:iam::*:role/aws-service-role/ops.emr-serverless.amazonaws.com/
AWSServiceRoleForAmazonEMRServerless*",
    "Condition": {"StringLike": {"iam:AWSServiceName": "ops.emr-
serverless.amazonaws.com"}}
}
```

如果您删除此服务相关角色，然后需要再次创建，请使用相同流程在账户中重新创建此角色。当您创建新的 EMR Serverless 应用程序时，EMR Serverless 会再次为您创建服务相关角色。

您还可以使用 IAM 控制台通过 EMR Serverless 用例创建服务相关角色。在 AWS CLI 或 AWS API 中，使用服务名称创建服务相关角色。ops.emr-serverless.amazonaws.com有关更多信息，请参阅《IAM 用户指南》中的创建服务相关角色。如果您删除了此服务相关角色，请使用同样的过程再次创建角色。

编辑 EMR Serverless 的服务相关角色

EMR Serverless 不允许您编辑 AWSServiceRoleForAmazonEMRServerless 服务相关角色，因为可能会有多个实体引用该角色。您无法编辑 EMR Serverless 服务相关角色使用的 AWS拥有的 IAM 策略，因为它包含 EMR Serverless 所需的所有必要权限。不过，您可以使用 IAM 编辑角色的说明。

使用 IAM 编辑 AWSServiceRoleForAmazonEMRServerless 服务相关角色的描述

将以下语句添加到需要编辑服务相关角色的描述的 IAM 实体的权限策略。

```
{
    "Effect": "Allow",
    "Action": [
        "iam: UpdateRoleDescription"
    ],
    "Resource": "arn:aws:iam::*:role/aws-service-role/ops.emr-serverless.amazonaws.com/
AWSServiceRoleForAmazonEMRServerless*",
    "Condition": {"StringLike": {"iam:AWSServiceName": "ops.emr-
serverless.amazonaws.com"}}
}
```

有关更多信息，请参阅《IAM 用户指南》中的编辑服务相关角色。

删除 EMR Serverless 的服务相关角色

如果不再需要使用某个需要服务相关角色的特征或服务，我们建议您删除该角色。这样可避免出现未被主动监控或维护的未使用实体。但是，请先删除所有区域中的所有 EMR Serverless 应用程序，然后删除服务相关角色。

> **Note**
>
> 如果在尝试删除与角色关联的资源时，EMR Serverless 服务正在使用该角色，那么删除可能会失败。如果发生这种情况，请等待几分钟后重试。

使用 IAM 删除 AWSServiceRoleForAmazonEMRServerless 服务相关角色

将以下语句添加到需要删除服务相关角色的 IAM 实体的权限策略。

```
{
    "Effect": "Allow",
    "Action": [
        "iam:DeleteServiceLinkedRole",
        "iam:GetServiceLinkedRoleDeletionStatus"
    ],
    "Resource": "arn:aws:iam::*:role/aws-service-role/ops.emr-serverless.amazonaws.com/
AWSServiceRoleForAmazonEMRServerless*",
    "Condition": {"StringLike": {"iam:AWSServiceName": "ops.emr-
serverless.amazonaws.com"}}
}
```

使用 IAM 手动删除服务关联角色

使用 IAM 控制台 AWS CLI、或 AWS API 删除 AWSServiceRoleForAmazonEMRServerless 服务相关角色。有关更多信息，请参阅《IAM 用户指南》中的删除服务相关角色。

EMR Serverless 服务相关角色的支持区域

EMR Serverless 支持在提供服务的所有区域使用服务相关角色。有关更多信息，请参阅 AWS 区域和端点。

Amazon EMR Serverless 的作业运行时角色

您可以指定 EMR Serverless 作业运行在代表您调用其他服务时可以代入的 IAM 角色权限。这包括访问任何数据源、目标以及其他 AWS 资源（例如 Amazon Redshift 集群和 DynamoDB 表）的 Amazon S3。要了解有关如何创建角色的更多信息，请参阅 创建作业运行时角色。

运行时策略示例

您可以将运行时策略（如下所示）附加到作业运行时角色。以下作业运行时策略允许：

- 对包含 EMR 示例的 Amazon S3 存储桶的读取访问权限。
- 对 S3 存储桶的完全访问权限。
- 创建并读取 AWS Glue 数据目录的权限。

要添加对 DynamoDB 等其他 AWS 资源的访问权限，您需要在创建运行时角色时在策略中包含这些资源的权限。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "ReadAccessForEMRSamples",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::*.elasticmapreduce",
        "arn:aws:s3:::*.elasticmapreduce/*"
      ]
    },
    {
      "Sid": "FullAccessToS3Bucket",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:ListBucket",
        "s3:DeleteObject"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket",
        "arn:aws:s3:::amzn-s3-demo-bucket/*"
      ]
    },
    {
      "Sid": "GlueCreateAndReadDataCatalog",
      "Effect": "Allow",
      "Action": [
        "glue:GetDatabase",
        "glue:CreateDatabase",
        "glue:GetDataBases",
        "glue:CreateTable",
        "glue:GetTable",
        "glue:UpdateTable",
        "glue:DeleteTable",
        "glue:GetTables",
```

```
        "glue:GetPartition",
        "glue:GetPartitions",
        "glue:CreatePartition",
        "glue:BatchCreatePartition",
        "glue:GetUserDefinedFunctions"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

传递角色权限

您可以将 IAM 权限策略附加到用户角色，允许用户仅传递批准的角色。这样，管理员就可以控制哪些用户可以将特定作业运行时角色传递给 EMR Serverless 作业。要了解有关设置权限的更多信息，请参阅向用户授予将角色传递给 AWS 服务的权限。

以下是一个示例策略，该策略允许将作业运行时角色传递给 EMR Serverless 服务主体。

```
{
     "Effect": "Allow",
     "Action": "iam:PassRole",
     "Resource": "arn:aws:iam::1234567890:role/JobRuntimeRoleForEMRServerless",
        "Condition": {
                "StringLike": {
                    "iam:PassedToService": "emr-serverless.amazonaws.com"
                }
            }
}
```

与运行时角色关联的托管权限策略

当您通过 EMR Studio 控制台向 EMR Serverless 提交作业运行时，有一个步骤需要您选择要与您的应用程序关联的运行时角色。控制台中的每个选择都有关联的底层托管策略，了解这些策略非常重要。这三个选择如下：

1. 所有存储桶 — 当您选择此选项时，它会指定 AmazonS3 FullAccess AWS 托管策略，该策略提供对所有存储桶的完全访问权限。

2. 特定存储桶：这指定您选择的每个存储桶的 Amazon 资源名称（ARN）标识符。不包括底层托管策略。
3. 无：不包括托管策略权限。

我们建议添加特定存储桶。如果您选择所有存储桶，请记住，它会为所有存储桶设置完全访问权限。

EMR Serverless 的用户访问策略示例

您可以根据您希望每个用户在与 EMR Serverless 应用程序交互时执行的操作，为用户设置精细的策略。以下策略示例可能有助于为用户设置适当的权限。本节仅重点介绍 EMR Serverless 策略。有关 EMR Studio 用户策略的示例，请参阅配置 EMR Studio 用户权限。有关如何将策略附加到 IAM 用户（主体）的信息，请参阅《IAM 用户指南》中的管理 IAM 策略。

高级用户策略

要向 EMR Serverless 授予所有必需的操作，请创建 `AmazonEMRServerlessFullAccess` 策略并将其附加到所需的 IAM 用户、角色或组。

以下是一个示例策略，该策略允许高级用户创建和修改 EMR Serverless 应用程序，并执行提交和调试作业等其他操作。示例显示了 EMR Serverless 需要对其他服务执行的所有操作。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "EMRServerlessActions",
      "Effect": "Allow",
      "Action": [
        "emr-serverless:CreateApplication",
        "emr-serverless:UpdateApplication",
        "emr-serverless:DeleteApplication",
        "emr-serverless:ListApplications",
        "emr-serverless:GetApplication",
        "emr-serverless:StartApplication",
        "emr-serverless:StopApplication",
        "emr-serverless:StartJobRun",
        "emr-serverless:CancelJobRun",
        "emr-serverless:ListJobRuns",
        "emr-serverless:GetJobRun"
```

```
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

当您启用到 VPC 的网络连接时，EMR Serverless 应用程序会创建 Amazon EC2 弹性网络接口（ENI），与 VPC 资源通信。以下策略确保任何新的 EC2 ENIs 都只能在 EMR 无服务器应用程序的环境中创建。

> **Note**
>
> 我们强烈建议您设置此政策，以确保 ENIs除非在启动 EMR Serverless 应用程序的情况下，否则用户无法创建 EC2。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowEC2ENICreationWithEMRTags",
      "Effect": "Allow",
      "Action": [
        "ec2:CreateNetworkInterface"
      ],
      "Resource": [
        "arn:aws:ec2:*:*:network-interface/*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:CalledViaLast": "ops.emr-serverless.amazonaws.com"
        }
      }
    }
  ]
}
```

如果要限制 EMR Serverless 对某些子网的访问，可使用标签条件来标记每个子网。此 IAM 策略确保 EMR 无服务器应用程序只能在允许的子网 ENIs 内创建 EC2。

```
{
    "Sid": "AllowEC2ENICreationInSubnetAndSecurityGroupWithEMRTags",
    "Effect": "Allow",
    "Action": [
        "ec2:CreateNetworkInterface"
    ],
    "Resource": [
        "arn:aws:ec2:*:*:subnet/*",
        "arn:aws:ec2:*:*:security-group/*"
    ],
    "Condition": {
        "StringEquals": {
            "aws:ResourceTag/KEY": "VALUE"
        }
    }
}
```

> **Important**
>
> 如果您是创建第一个应用程序的管理员或高级用户，则必须配置权限策略，以允许您创建 EMR Serverless 服务相关角色。要了解更多信息，请参阅使用 EMR Serverless 的服务相关角色。

以下 IAM 策略允许您为账户创建 EMR Serverless 服务相关角色。

```
{
   "Sid":"AllowEMRServerlessServiceLinkedRoleCreation",
   "Effect":"Allow",
   "Action":"iam:CreateServiceLinkedRole",
   "Resource":"arn:aws:iam::account-id:role/aws-service-role/ops.emr-
serverless.amazonaws.com/AWSServiceRoleForAmazonEMRServerless"
}
```

数据工程师策略

以下是一个示例策略，该策略允许用户对 EMR Serverless 应用程序具有只读权限，并能提交和调试作业。请记住，由于此策略不会明确拒绝操作，因此仍可使用其它策略声明来授予对指定操作的访问权限。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "EMRServerlessActions",
      "Effect": "Allow",
      "Action": [
        "emr-serverless:ListApplications",
        "emr-serverless:GetApplication",
        "emr-serverless:StartApplication",
        "emr-serverless:StartJobRun",
        "emr-serverless:CancelJobRun",
        "emr-serverless:ListJobRuns",
        "emr-serverless:GetJobRun"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

使用标签进行访问控制

您可以使用标签条件进行精细访问控制。例如，您可以限制一个团队的用户，使他们只能向标记有团队名称的 EMR Serverless 应用程序提交作业。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "EMRServerlessActions",
      "Effect": "Allow",
      "Action": [
        "emr-serverless:ListApplications",
        "emr-serverless:GetApplication",
```

```
        "emr-serverless:StartApplication",
        "emr-serverless:StartJobRun",
        "emr-serverless:CancelJobRun",
        "emr-serverless:ListJobRuns",
        "emr-serverless:GetJobRun"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

基于标签的访问控制策略

您可以在基于身份的策略中使用条件，根据标签控制对应用程序和作业运行的访问。

以下示例演示了将条件运算符与 EMR Serverless 条件键结合使用的不同场景和方法。这些 IAM policy 声明仅用于演示目的，并且不应用于生产环境。可通过多种方式来组合策略声明，以根据要求授予和拒绝权限。有关规划和测试 IAM 策略的更多信息，请参阅 IAM 用户指南。

> **Important**
>
> 一个重要注意事项是，应该明确拒绝添加标签操作的权限。这可以防止用户标记资源，从而为其授予您不打算授予的权限。如果资源的标记操作未被拒绝，用户可以修改标记并规避基于标签的策略意图。有关拒绝添加标签操作的策略示例，请参阅拒绝添加和删除标签的访问权限。

以下示例演示了基于身份的权限策略，这些策略用于控制 EMR Serverless 应用程序允许的操作。

仅允许带特定标签值资源上的操作

在以下策略示例中，`StringEquals` 条件运算符尝试将 `dev` 与标签部分的值匹配。如果标签部分尚未添加到应用程序或不包含值 `dev`，则策略不会应用，并且此策略将不允许这些操作。如果没有其他策略语句允许这些操作，则用户只能使用标签中包含此值的应用程序。

JSON

```
{
```

```
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-serverless:GetApplication"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/department": "dev"
        }
      },
      "Sid": "AllowEMRSERVERLESSGetapplication"
    }
  ]
}
```

您也可以使用条件运算符指定多个标签值。例如，要允许对 department 标签包含值 dev 或 test 的应用程序执行操作，请将前面示例中的条件块替换为以下内容。

```
"Condition": {
        "StringEquals": {
          "emr-serverless:ResourceTag/department": ["dev", "test"]
        }
      }
```

创建资源时需要标签

在下面的示例中，创建应用程序时需要应用标签。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
```

```
        "emr-serverless:CreateApplication"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": "us-east-1"
        }
      },
      "Sid": "AllowEMRSERVERLESSCreateapplication"
    }
  ]
}
```

以下策略语句仅允许用户在应用程序具有 department 标签（可包含任何值）的情况下创建虚拟集群。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-serverless:CreateApplication"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": ["us-east-1", "us-west-2"]
        }
      },
      "Sid": "AllowEMRSERVERLESSCreateapplication"
    }
  ]
}
```

拒绝添加和删除标签的访问权限

此策略可防止用户在 EMR Serverless 应用程序上添加或删除 `department` 标签值不是 `dev` 的标签。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "emr-serverless:TagResource",
        "emr-serverless:UntagResource"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringNotEquals": {
          "aws:PrincipalTag/department": "dev"
        }
      },
      "Sid": "AllowEMRSERVERLESSTagresource"
    }
  ]
}
```

EMR Serverless 基于身份的策略示例

默认情况下，用户和角色没有权限创建或修改 Amazon EMR Serverless 资源。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。

要了解如何使用这些示例 JSON 策略文档创建基于 IAM 身份的策略，请参阅《IAM 用户指南》中的创建 IAM 策略（控制台）。

有关 Amazon EMR Serverless 定义的操作和资源类型（包括每种资源类型的格式）的详细信息，请参阅《服务授权参考》中的 Amazon EMR Serverless 的操作、资源和条件密钥。 ARNs

主题

- 策略最佳实践
- 允许用户访问其自己的权限

策略最佳实践

> **Note**
>
> EMR Serverless 不支持托管策略，因此下面列出的第一种做法并不适用。

基于身份的策略决定了某人是否可以在您的账户中创建、访问或删除 Amazon EMR Serverless 资源。这些操作可能会使 AWS 账户产生成本。创建或编辑基于身份的策略时，请遵循以下指南和建议：

- 开始使用 AWS 托管策略并转向最低权限权限 — 要开始向用户和工作负载授予权限，请使用为许多常见用例授予权限的AWS 托管策略。它们在你的版本中可用 AWS 账户。我们建议您通过定义针对您的用例的 AWS 客户托管策略来进一步减少权限。有关更多信息，请参阅《IAM 用户指南》中的AWS 托管策略或工作职能的AWS 托管策略。
- 应用最低权限：在使用 IAM 策略设置权限时，请仅授予执行任务所需的权限。为此，您可以定义在特定条件下可以对特定资源执行的操作，也称为最低权限许可。有关使用 IAM 应用权限的更多信息，请参阅《IAM 用户指南》中的 IAM 中的策略和权限。
- 使用 IAM 策略中的条件进一步限制访问权限：您可以向策略添加条件来限制对操作和资源的访问。例如，您可以编写策略条件来指定必须使用 SSL 发送所有请求。如果服务操作是通过特定的方式使用的，则也可以使用条件来授予对服务操作的访问权限 AWS 服务，例如 CloudFormation。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件。
- 使用 IAM Access Analyzer 验证您的 IAM 策略，以确保权限的安全性和功能性：IAM Access Analyzer 会验证新策略和现有策略，以确保策略符合 IAM 策略语言（JSON）和 IAM 最佳实践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，以帮助您制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的使用 IAM Access Analyzer 验证策略。
- 需要多重身份验证 (MFA)-如果 AWS 账户您的场景需要 IAM 用户或根用户，请启用 MFA 以提高安全性。若要在调用 API 操作时需要 MFA，请将 MFA 条件添加到您的策略中。有关更多信息，请参阅《IAM 用户指南》中的使用 MFA 保护 API 访问。

有关 IAM 中的最佳实操的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实践。

允许用户访问其自己的权限

该示例说明了您如何创建策略，以允许 IAM 用户查看附加到其用户身份的内联和托管式策略。此策略包括在控制台上或使用 AWS CLI 或 AWS API 以编程方式完成此操作的权限。

```
{
    "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": "*"
        }
    ]
}
```

Amazon EMR 托管策略的无服务器更新 AWS

访问有关自该服务开始跟踪这些更改以来对 Amazon EMR Serverless AWS 托管策略更新的详细信息。要获取有关此页面更改的自动提醒，请订阅 Amazon EMR Serverless 文档历史记录页面上的 RSS 源。

更改	描述	日期
亚马逊 EMRServerless ServiceRolePolicy -更新现有政策	亚马逊 EMR Serverless 在亚马逊政策中添加 `EC2PolicyStatement` 了新的`SidCloudWatchPolicyStatement` 和。EMRServerless ServiceRolePolicy	2024 年 1 月 25 日
亚马逊 EMRServerless ServiceRolePolicy -更新现有政策	Amazon EMR Serverless 添加了新的权限，允许 Amazon EMR Serverless 在 `"AWS/Usage"` 命名空间中发布有关 vCPU 使用情况的汇总账户指标。	2023 年 4 月 20 日
Amazon EMR Serverless 开始跟踪更改	Amazon EMR Serverless 开始跟踪其 AWS 托管策略的更改。	2023 年 4 月 20 日

Amazon EMR Serverless 身份和访问问题排查

使用以下信息帮助您诊断和修复在使用 Amazon EMR Serverless 和 IAM 时可能遇到的常见问题。

主题

- 我无权在 Amazon EMR Serverless 中执行操作
- 我无权执行 iam：PassRole
- 我想允许 AWS 账户之外的用户访问我的 Amazon EMR 无服务器资源
- 我无法从 EMR Studio 打开实时 UI/Spark 历史服务器来调试我的作业，或者当我尝试使用获取日志时出现 API 错误 get-dashboard-for-job-run

我无权在 Amazon EMR Serverless 中执行操作

如果 AWS 管理控制台 告诉您您无权执行某项操作，请联系您的管理员寻求帮助。管理员是指提供用户名和密码的人员。

当 mateojackson 用户尝试使用控制台访问有关虚构 *my-example-widget* 资源的详细信息，但不拥有虚构 emr-serverless:*GetWidget* 权限时，会发生以下示例错误。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform: emr-
serverless:GetWidget on resource: my-example-widget
```

在这种情况下，Mateo 请求他的管理员更新其策略，以允许他使用 emr-serverless:*GetWidget* 操作访问 *my-example-widget* 资源。

我无权执行 iam：PassRole

如果收到一条错误消息，表示您无权执行 iam:PassRole 操作，则必须更新策略以允许将角色传递给 Amazon EMR Serverless。

有些 AWS 服务 允许您将现有角色传递给该服务，而不是创建新的服务角色或服务相关角色。为此，您必须具有将角色传递到服务的权限。

当名为 marymajor 的 IAM 用户尝试使用控制台在 Amazon EMR Serverless 中执行操作时，会发生以下示例错误。但是，服务必须具有服务角色所授予的权限才可执行此操作。Mary 不具有将角色传递到服务的权限。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在这种情况下，必须更新 Mary 的策略以允许她执行 iam:PassRole 操作。

如果您需要帮助，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

我想允许 AWS 账户之外的用户访问我的 Amazon EMR 无服务器资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。您可以指定谁值得信赖，可以代入角色。对于支持基于资源的策略或访问控制列表 (ACLs) 的服务，您可以使用这些策略向人们授予访问您的资源的权限。

要了解更多信息，请参阅以下内容：

- 要了解 Amazon EMR Serverless 是否支持这些功能，请参阅 Amazon EMR Serverless 中的 Identity and Access Management （ IAM ）。
- 要了解如何提供对您拥有的资源的访问权限 AWS 账户 ，请参阅 IAM 用户指南中的向您拥有 AWS 账户 的另一个 IAM 用户提供访问权限。
- 要了解如何向第三方提供对您的资源的访问权限 AWS 账户，请参阅 IAM 用户指南中的向第三方提供访问权限。 AWS 账户
- 要了解如何通过身份联合验证提供访问权限，请参阅《IAM 用户指南》中的为经过外部身份验证的用户（身份联合验证）提供访问权限。
- 要了解使用角色和基于资源的策略进行跨账户访问之间的差别，请参阅《IAM 用户指南》中的 IAM 中的跨账户资源访问。

我无法从 EMR Studio 打开实时 UI/Spark 历史服务器来调试我的作业，或者当我尝试使用获取日志时出现 API 错误 `get-dashboard-for-job-run`

如果您使用 EMR Serverless 托管存储进行日志记录，而您的 EMR Serverless 应用程序位于具有 Amazon S3 VPC 端点的私有子网中，并且您附加了端点策略来控制访问，请将您的 VPC 策略中使用托管存储的 EMR Serverless 日志记录提到的权限添加到 S3 网关端点，以便 EMR Serverless 存储和提供应用程序日志。

可信身份传播

在 Amazon EMR 7.8.0 及更高版本中，您可以通过 Apache Livy Endpoint 将用户身份从 AWS IAM 身份中心传播到使用 EMR Serverless 的交互式工作负载。Apache Livy 交互式工作负载会将提供的身份进一步传播到 Amazon S3、Lake Formation 和 Amazon Redshift 等下游服务，从而在这些下游服务中通过用户身份实现安全的数据访问。以下各节提供了概念概述、先决条件以及通过 Apache Livy Endpoint 将身份启动和传播到 EMR Serverless 交互式工作负载所需的步骤。

概述

对于任何规模和类型的组织，推荐使用 I@@ AM Identit y Center AWS 进行员工身份验证和授权。使用 Identity Center，在现有身份源中创建和管理用户身份 AWS，或者连接现有身份源，包括微软 Active Directory、Okta、Ping Identity JumpCloud、Google Workspace 和微软 Entra ID（前身为 Azure AD）。

可信身份传播是 AWS IAM Identity Center 的一项功能，互联 AWS 服务的管理员可以使用该功能来授予和审计对服务数据的访问权限。对这些数据的访问权限基于用户属性，例如组关联。设置可信身份

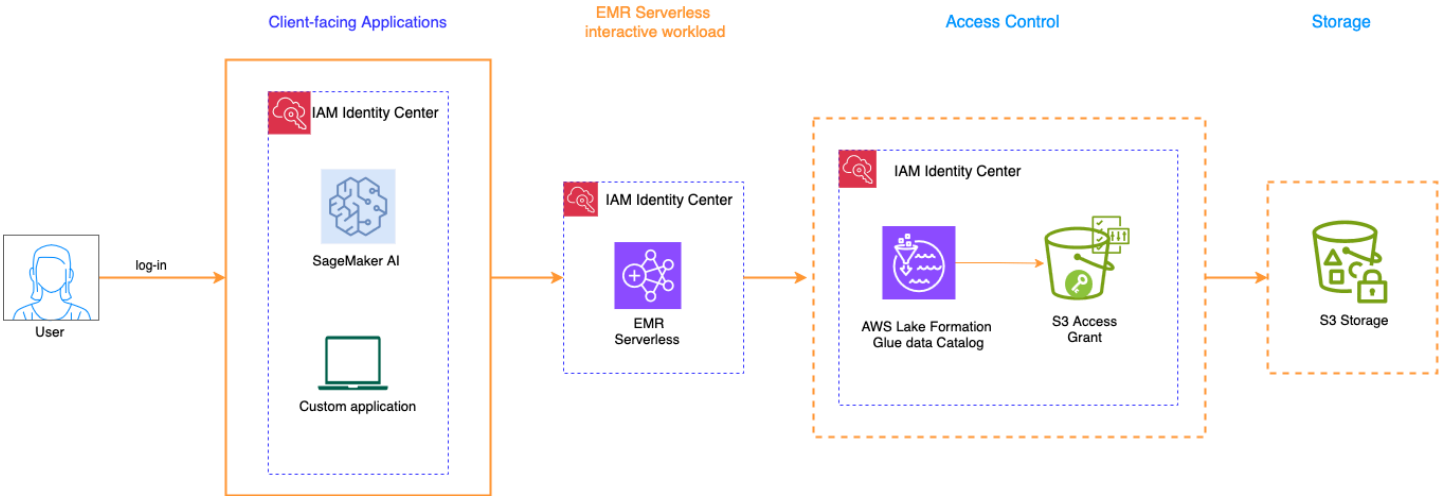
传播需要互联 AWS 服务的管理员和 IAM Identity Center 管理员之间的协作。有关更多信息，请参阅《IAM Identity Center 用户指南》中的先决条件和注意事项。

功能和优势

将 EMR Serverless Apache Livy Endpoint 与 IAM Identity Center 可信身份传播集成具有以下优势：

- 能够在 La AWS ke Form AWS ation 托管的 Glue 数据目录表上使用身份中心身份强制执行表级授权。
- 能够在 Amazon Redshift 集群上使用 IAM Identity Center 身份强制执行授权。
- 可实现端到端的全程用户操作跟踪，以满足审计的需要。
- 能够在 S3 Access Grants 托管的 S3 前缀上使用 Identity Center 身份强制执行 Amazon S3 前缀级别的授权。

工作原理



使用案例示例

数据准备和特征工程

来自多个研究团队的数据科学家可以使用统一的数据平台协作完成复杂的项目。他们使用企业凭证登录 SageMaker 人工智能，立即访问跨多个 AWS 账户的庞大共享数据湖。开始新机器学习模型的特征工程时，通过 EMR Serverless 启动的 Spark 会话会根据其传播的身份强制执行 Lake Formation 列级和行级安全策略。科学家可以使用自己熟悉的工具高效地准备数据和开展特征工程，同时合规团队也可以确信每次数据交互都会得到自动跟踪和审计。这种安全的协作环境不仅可以加快研究管道，同时还可确保遵守受监管行业严格的数据保护标准。

可信身份传播入门

本节帮助您使用 Apache Livy Endpoint 配置 EMR 无服务器应用程序，使其与 AWS IAM 身份中心集成并启用可信身份传播。

先决条件

- 您要在其中创建可信身份传播的 AWS 区域中的身份中心实例，该实例已启用 EMR Serverless Apache Livy Endpoint。一个 AWS 账户的身份中心实例只能存在于单个区域中。请参阅启用 IAM 身份中心并将您的身份来源中的用户和群组配置到 IAM Identit y Center。
- 为交互式工作负载将与之交互的下游服务（例如 Lake Formation 或 S3 访问权限管控或 Amazon Redshift 集群）启用可信身份传播，以便访问数据。

创建启用了可信身份传播的 EMR Serverless 应用程序的权限

除了访问 EMR Serverless 所需的基本权限外，您还必须为用于创建启用可信身份传播的 EMR Serverless 应用程序的 IAM 身份或角色配置其他权限。对于可信身份传播，EMR Serverless 是您账户中管理身份中心应用程序的单 creates/bootstraps 一服务，该服务利用该服务向下游进行身份验证和身份传播。

```
"sso:DescribeInstance",
"sso:CreateApplication",
"sso:DeleteApplication",
"sso:PutApplicationAuthenticationMethod",
"sso:PutApplicationAssignmentConfiguration",
"sso:PutApplicationGrant",
"sso:PutApplicationAccessScope"
```

- `sso:DescribeInstance`— 授予描述和验证您在参数中指定的 IAM Identity Center InstanceArn 的权限。 identity-center-configuration
- `sso:CreateApplication`— 授予创建用于操作的 EMR Serverless 托管 IAM 身份中心应用程序的权限。 trusted-identity-propatgion
- `sso:DeleteApplication`：授予清理 EMR Serverless 托管 IAM Identity Center 应用程序的权限
- `sso:PutApplicationAuthenticationMethod`：授予在 EMR Serverless 托管 IAM Identity Center 应用程序上放置 authenticationMethod 的权限，以便 emr-serverless 服务主体与 IAM Identity Center 应用程序交互。

- `sso:PutApplicationAssignmentConfiguration`— 授予在 IAM 身份中心应用程序上设置 ser-assignment-not-required “U” 设置的权限。
- `sso:PutApplicationGrant`：授予在 IAM Identity Center 应用程序上应用 token-exchange、introspectToken、refreshToken 和 revokeToken 授权的权限。
- `sso:PutApplicationAccessScope`：授予将启用可信身份传播的下游作用域应用于 IAM Identity Center 应用程序的权限。我们应用 “redshift: connect”、“lakeformation: query” 和 “s3: read_write” 范围来启用这些服务。 trusted-identity-propagation

创建启用可信身份传播的 EMR Serverless 应用程序

您必须使用 `identityCenterInstanceArn` 指定 `–identity-center-configuration` 字段，才能在应用程序中启用可信身份传播。使用以下示例命令创建启用可信身份传播的 EMR Serverless 应用程序。

> **Note**
>
> 您还必须指定 `--interactive-configuration '{"livyEndpointEnabled":true}'`，因为仅对 Apache Livy Endpoint 启用了可信身份传播。

```
aws emr-serverless create-application \
  --release-label emr-7.8.0 \
  --type "SPARK" \
  --identity-center-configuration '{"identityCenterInstanceArn" :
 "arn:aws:sso:::instance/ssoins-123456789"}' \
  --interactive-configuration '{"livyEndpointEnabled":true}'
```

- `identity-center-configuration`：（可选）启用 Identity Center 可信身份传播（如果已指定）。
- `identityCenterInstanceArn`：（必需）Identity Center 实例 ARN。

如果您没有所需的 Identity Center 权限（如前所述），请先创建没有可信身份传播的 EMR Serverless 应用程序（例如，不要指定 `–identity-center-configuration` 参数），然后让您的 Identity Center 管理员通过调用 update-application API 来启用可信身份传播，请参见以下示例：

```
aws emr-serverless update-application \
```

```
  --application-id applicationId \
  --identity-center-configuration '{"identityCenterInstanceArn" :
"arn:aws:sso:::instance/ssoins-123456789"}'
```

EMR Serverless 会在您的账户中创建一个服务托管式 Identity Center 应用程序，服务会利用该应用程序进行身份验证并将身份传播到下游服务。EMR Serverless 创建的托管身份中心应用程序将在您账户中所有启用的 trusted-identity-propagation EMR Serverless 应用程序之间共享。

> **Note**
>
> 请勿手动修改托管式 Identity Center 应用程序的设置。任何更改都可能影响您账户中所有 trusted-identity-propagation启用的 EMR Serverless 应用程序。

传播身份所需的作业执行角色权限

由于 EMR-Serverless 利用身份增强型 job-execution-role凭证将身份传播到下游服务 AWS ，因此任务执行角色的信任策略必须具有额外的权限，才能使用身份增强任务执行角色凭证，`sts:SetContext`以允许下游 trusted-identity-propagation服务（例如 S3 访问授予、Lake Formation 或 Amazon Redshift）。要了解有关如何创建角色的更多信息，请参阅创建作业运行时角色。

JSON

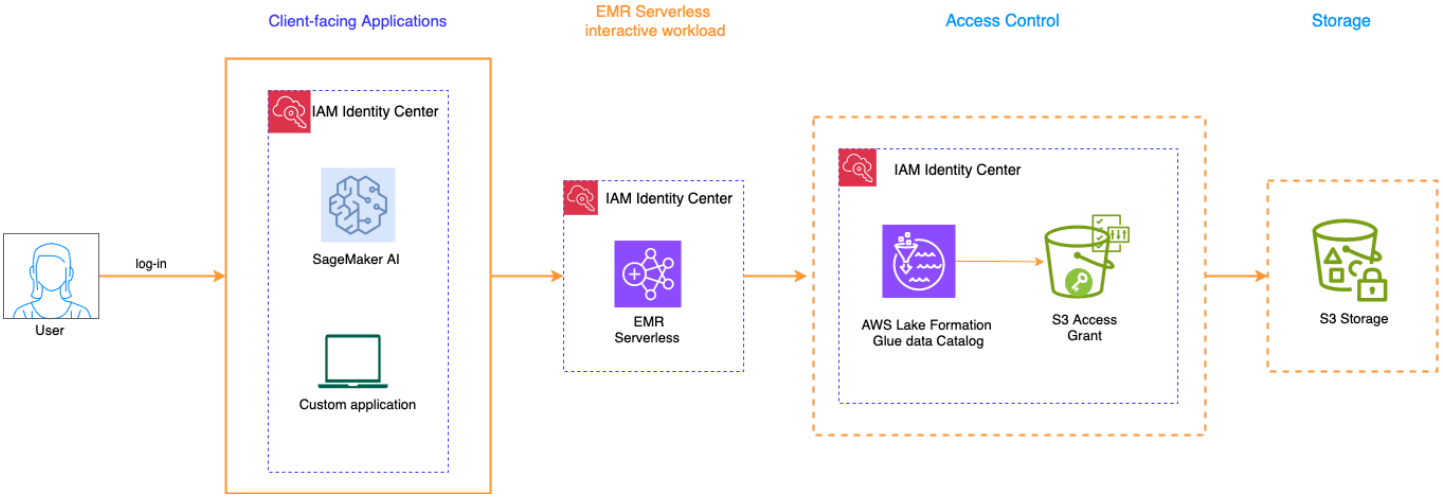
```
{
  "Version":"2012-10-17",
  "Statement": [
     {
     "Effect": "Allow",
     "Principal": {
     "Service": "emr-serverless.amazonaws.com"
     },
     "Action": [ "sts:AssumeRole", "sts:SetContext"]
     }
  ]
}
```

此外，还 JobExecutionRole 需要下游 AWS 服务的权限，job-run 会调用这些下游服务来使用用户身份获取数据。请参阅以下链接来配置 S3 访问权限管控、Lake Formation。

- 将 Lake Formation 与 EMR Serverless 结合使用
- 将 Amazon S3 访问权限授予与 EMR 无服务器配合使用

交互式工作负载的可信身份传播

通过 Apache Livy 端点将身份传播到交互式工作负载的步骤取决于您的用户是与托管开发环境（如 AWS Amazon SageMaker AI 托管开发环境）进行交互，还是作为面向客户端的应用程序与您自己的自托管笔记本环境进行交互。



AWS 托管开发环境

以下面向客户端的 AWS 托管应用程序支持使用 EMR-Serverless Apache Livy 端点进行可信身份传播：

- 亚马逊 SageMaker AI

客户自行管理的自托管 Notebook 环境

要为自定义开发的应用程序的用户启用可信身份传播，请参阅AWS 安全博客中的使用可信身份传播以编程方式访问 AWS 服务。

用户后台会话

用户后台会话使长时间运行的分析和机器学习流程即使在用户已从笔记本界面注销后仍能继续进行。此功能是通过 EMR Serverless 与 IAM Identity Center 的可信身份传播功能集成来实现的。本节介绍用户后台会话的配置选项和行为。

> **Note**
>
> 用户后台会话适用于通过 Amazon SageMaker Unified Studio 等笔记本界面启动的 Spark 工作负载。启用或禁用此功能仅影响新的 Livy 会话；现有活动的 Livy 会话不受影响。

配置用户后台会话

必须在两个级别上启用用户后台会话才能正常运行：

1. IAM 身份中心实例级别 — 通常由 IdC 管理员配置
2. EMR 无服务器应用程序级别 — 由 EMR 无服务器应用程序管理员配置

为 EMR 无服务器应用程序启用用户后台会话

要为 EMR Serverless 应用程序启用用户后台会话，您必须在创建或更新应用程序`identityCenterConfiguration`时将`userBackgroundSessionsEnabled``true`参数设置为。

先决条件

- 您用于 EMR Serverless 应用程序的 IAM 角色必须具有权限。 create/update `sso:PutApplicationSessionConfiguration`此权限允许 EMR Serverless 在 EMR Serverless 托管 IdC 应用程序级别启用用户后台会话。
- 您的 EMR Serverless 应用程序必须使用版本标签 7.8 或更高版本，并且必须启用可信身份传播。

要启用用户后台会话，请使用 AWS CLI

```
aws emr-serverless create-application \
    --name "my-analytics-app" \
    --type "SPARK" \
    --release-label "emr-7.8.0" \
    --identity-center-configuration '{"identityCenterInstanceArn":
 "arn:aws:sso:::instance/ssoins-1234567890abcdef", "userBackgroundSessionsEnabled":
 true}'
```

要更新现有应用程序，请执行以下操作：

```
aws emr-serverless update-application \
```

```
    --application-id applicationId \
    --identity-center-configuration '{"identityCenterInstanceArn":
"arn:aws:sso:::instance/ssoins-1234567890abcdef", "userBackgroundSessionsEnabled":
true}'
```

配置矩阵

有效的用户后台会话配置取决于 EMR Serverless 应用程序设置和 IAM Identity Center 实例级别设置：

用户后台会话配置矩阵

userBackgroundSession 已启用 IAM 身份中心	已启用 EMR 无服务器 userBackgroundSessions	行为
是	TRUE	用户后台会话已启用
是	FALSE	会话过期，用户注销
否	TRUE	应用程序 creation/update 因异常而失败
否	FALSE	会话过期，用户注销

默认用户后台会话持续时间

默认情况下，在 IAM Identity Center 中，所有用户后台会话的持续时间限制为 7 天。管理员可以在 IAM Identity Center 控制台中修改此持续时间。此设置在 IAM Identity Center 实例级别应用，会影响该实例内所有受支持的 IAM Identity Center 应用程序。

- 持续时间可以设置为从 15 分钟到 90 天之间的任何值。
- 此设置在 IAM Identity Center 控制台的 “设置” → “身份验证” → “配置” 下配置（“非交互式作业” 部分）

> **Note**
>
> EMR Serverless Livy 会话的单独最大持续时间限制为 24 小时。当达到 Livy 会话限制或用户后台会话持续时间时（以先到者为准），会话将终止。

禁用用户后台会话的影响

在 IAM 身份中心禁用用户后台会话时：

现有的 Livy 会话

如果它们是在启用用户后台会话的情况下启动的，则可以不间断地继续运行。这些会话将继续使用其现有的后台会话令牌，直到这些会话自然终止或被明确停止。

全新 Livy 会话

将使用标准的可信身份传播流程，并在用户注销或交互式会话到期（例如关闭 Amazon SageMaker Unified Studio JupyterLab 笔记本时）时终止。

更改用户后台会话持续时间

在 IAM Identity Center 中修改用户后台会话的持续时间设置时：

现有的 Livy 会话

继续以与启动时相同的后台会话持续时间运行。

全新 Livy 会话

将使用新的会话时长进行后台会话。

注意事项

会话终止条件

使用用户后台会话时，Livy 会话将继续运行，直到出现以下情况之一：

- 用户后台会话过期（基于 IdC 配置，最长 90 天）
- 管理员已手动撤销用户后台会话
- Livy 会话达到其空闲超时（默认值：上次执行语句后 1 小时）
- Livy 会话已达到其最大持续时间（24 小时）
- 用户明确停止或重新启动笔记本内核

数据持久性

使用用户后台会话时：

- 用户注销后无法重新连接到笔记本界面查看结果
- 将您的 Spark 语句配置为在执行完成之前将结果写入永久存储（例如 Amazon S3）

成本影响

- 即使用户结束了 Amazon SageMaker Unified Studio 会 JupyterLab 话，任务仍将继续运行直至完成，并且将在完成的整个运行期间产生费用。
- 监控您的活动后台会话，以避免因忘记或放弃会话而产生不必要的费用。

功能可用性

EMR Serverless 的用户后台会话可用于：

- 仅限 Spark 引擎（不支持 Hive 引擎）
- 仅限 Livy 交互式会话（不支持批处理作业和流式处理作业）
- EMR Serverless 版本标签 7.8 及更高版本

EMR 无服务器集成的注意事项 Trusted-Identity-Propagation

将 IAM 身份中心 Trusted-Identity-Propagation与 EMR Serverelss 应用程序配合使用时，请考虑以下几点：

- Amazon EMR 7.8.0 及更高版本支持通过 Identity Center 进行可信身份传播，而且仅支持 Apache Spark。
- 可信身份传播只能用于通过 Apache Livy 端点的 EMR Serverless 交互式工作负载。通过 EMR Studio 的交互式工作负载不支持可信身份传播
- Batch 作业和流式处理工作负载不支持可信身份传播
- 使用 AWS Lake Formation、使用可信身份传播的精细访问控制可用于通过 Apache Livy 端点使用 EMR Serverless 的交互式工作负载。
- 以下 AWS 区域支持使用 Amazon EMR 进行可信身份传播：
  - af-south-1：非洲（开普敦）
  - ap-east-1：亚太地区（香港）
  - ap-northeast-1：亚太地区（东京）
  - ap-northeast-2：亚太地区（首尔）

- ap-northeast-3：亚太地区（大阪）
- ap-south-1：亚太地区（孟买）
- ap-southeast-1：亚太地区（新加坡）
- ap-southeast-2：亚太地区（悉尼）
- ap-southeast-3：亚太地区（雅加达）
- ca-central-1：加拿大（中部）
- ca-west-1：加拿大（卡尔加里）
- eu-central-1：欧洲地区（法兰克福）
- eu-north-1：欧洲（斯德哥尔摩）
- eu-south-1：欧洲地区（米兰）
- eu-south-2：欧洲（西班牙）
- eu-west-1：欧洲地区（爱尔兰）
- eu-west-2：欧洲（伦敦）
- eu-west-3：欧洲（巴黎）
- me-central-1：中东（阿联酋）
- me-south-1：中东（巴林）
- sa-east-1：南美洲（圣保罗）
- us-east-1：美国东部（弗吉尼亚北部）
- us-east-2：美国东部（俄亥俄）
- us-west-1：美国西部（加利福尼亚北部）
- us-west-2：美国西部（俄勒冈）

将 Lake Formation 与 EMR Serverless 结合使用

您可以将 EMR Serverless 应用程序配置为使用具有完整表访问权限或精细访问控制的 Lake Formation。有关每种访问模式下支持的功能的详细信息，请查看下表。

功能可用性

功能	可从
Hive、Iceberg 表的读取操作（选择、描述）	EMR 7.2+

功能	可从
多方言视图	EMR 7.6+
Delta Lake 和 Hudi 表的读取操作（选择、描述）	EMR 7.6+
Hive、Iceberg 的完整桌子访问权限	EMR 7.9+
三角洲湖全桌畅游	EMR 7.11+
为 Hive、Iceberg 和 Delta Lake 表写入操作（DDL、DML）	EMR 7.12+
Hudi 的完整桌子访问权限	EMR 7.12+

EMR Serverless 的 Lake Formation 完整表访问权限

在 Amazon EMR 7.8.0 及更高版本中，您可以利用带有 Glue Data Catalog 的 Lake Formation AWS ，其中任务运行时角色拥有完整的表权限，不受细粒度访问控制的限制。此功能允许您从 EMR Serverless Spark 批处理和交互式作业中读取和写入受 Lake Formation 保护的表。请参阅以下部分，了解有关 Lake Formation 以及如何将其与 EMR Serverless 结合使用的更多信息。

对 Lake Formation 使用全表访问

您可以从 EMR Serverless Spark 作业或交互式会话中访问受 La AWS ke Formation 保护的 Glue 数据目录表，在这些会话中，作业的运行时角色具有完整表访问权限。您无需在 EMR 无服务器应用程序上启用 AWS Lake Formation。将 Spark 作业配置为全表访问权限 (FTA) 时， AWS Lake Formation 凭据将用于 La AWS ke Formati read/write on 注册表的 S3 数据，而作业的运行时角色凭据将用于未在 AWS Lake Formation 中注册的 read/write 表。

> **Important**
>
> 请勿启用 AWS Lake Formation 进行精细访问控制。在同一个 EMR 集群或应用程序上，作业不能同时运行全表访问 (FTA) 和精细访问控制 (FGAC)。

步骤 1：在 Lake Formation 中启用全表访问

要使用全表访问 (FTA) 模式，您必须允许第三方查询引擎访问数据，而无需在 La AWS ke Formation 中进行 IAM 会话标签验证。要启用此模式，请按照 Application integration for full table access 中的步骤操作。

> **Note**
>
> 访问跨账户表时，必须在生产者账户和消费者账户中启用全表访问。同样，在访问跨区域表时，必须在生产者和使用者区域中都启用此设置。

第 2 步：设置作业运行时角色的 IAM 权限

要获得基础数据的读取或写入权限，除 Lake Formation 权限外，作业运行时角色还需要具有 `lakeformation:GetDataAccess` IAM 权限。获得此权限后，Lake Formation 将授权访问数据的临时凭证请求。

下面是一个策略示例，展示了如何提供 IAM 权限以访问 Amazon S3 中的脚本、将日志上传到 S3、AWS Glue API 权限以及访问 Lake Formation 的权限。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "ScriptAccess",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::*.amzn-s3-demo-bucket/scripts"
      ]
    },
    {
      "Sid": "LoggingAccess",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
```

```
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket/logs/*"
      ]
    },
    {
      "Sid": "GlueCatalogAccess",
      "Effect": "Allow",
      "Action": [
        "glue:Get*",
        "glue:Create*",
        "glue:Update*"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "LakeFormationAccess",
      "Effect": "Allow",
      "Action": [
        "lakeformation:GetDataAccess"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

步骤 2.1：配置 Lake Formation 权限

- 从 S3 读取数据的 Spark 作业需要 Lake Formation SELECT 权限。
- 在 S3 中输入 write/delete 数据的 Spark 任务需要 Lake Formation ALL（超级）权限。
- 与 Glue Data Catalog 交互的 Spark 作业需要视情况具有 DESCRIBE、ALTER、DROP 权限。

有关更多信息，请参阅授予对 Data Catalog 资源的权限。

步骤 3：使用 Lake Formation 初始化 Spark 会话以实现全表访问

先决条件

AWS 必须将 Glue 数据目录配置为元数据仓才能访问 Lake Formation 表。

设置以下设置以将 Glue 目录配置为元存储：

```
--conf spark.sql.catalogImplementation=hive
--conf
 spark.hive.metastore.client.factory.class=com.amazonaws.glue.catalog.metastore.AWSGlueDataCata
```

有关为 EMR Serverless 启用 Data Catalog 的更多信息，请参阅 EMR Serverless 的元存储配置。

要访问在 AWS Lake Formation 中注册的表，需要在 Spark 初始化期间设置以下配置，将 Spark 配置为使用 AWS Lake Formation 凭据。

Hive

```
--conf
 spark.hadoop.fs.s3.credentialsResolverClass=com.amazonaws.glue.accesscontrol.AWSLakeFormati
--conf spark.hadoop.fs.s3.useDirectoryHeaderAsFolderObject=true
--conf spark.hadoop.fs.s3.folderObject.autoAction.disabled=true
--conf spark.sql.catalog.skipLocationValidationOnCreateTable.enabled=true
--conf spark.sql.catalog.createDirectoryAfterTable.enabled=true
--conf spark.sql.catalog.dropDirectoryBeforeTable.enabled=true
```

Iceberg

```
--conf spark.sql.catalog.spark_catalog=org.apache.iceberg.spark.SparkSessionCatalog
--conf spark.sql.catalog.spark_catalog.warehouse=S3_DATA_LOCATION
--conf spark.sql.catalog.spark_catalog.client.region=REGION
--conf spark.sql.catalog.spark_catalog.type=glue
--conf spark.sql.catalog.spark_catalog.glue.account-id=ACCOUNT_ID
--conf spark.sql.catalog.spark_catalog.glue.lakeformation-enabled=true
--conf spark.sql.catalog.dropDirectoryBeforeTable.enabled=true
```

Delta Lake

```
--conf
 spark.hadoop.fs.s3.credentialsResolverClass=com.amazonaws.glue.accesscontrol.AWSLakeFormati
--conf spark.hadoop.fs.s3.useDirectoryHeaderAsFolderObject=true
--conf spark.hadoop.fs.s3.folderObject.autoAction.disabled=true
```

```
--conf spark.sql.catalog.skipLocationValidationOnCreateTable.enabled=true
--conf spark.sql.catalog.createDirectoryAfterTable.enabled=true
--conf spark.sql.catalog.dropDirectoryBeforeTable.enabled=true
```

Hudi

```
--conf
 spark.hadoop.fs.s3.credentialsResolverClass=com.amazonaws.glue.accesscontrol.AWSLakeFormati
--conf spark.hadoop.fs.s3.useDirectoryHeaderAsFolderObject=true
--conf spark.hadoop.fs.s3.folderObject.autoAction.disabled=true
--conf spark.sql.catalog.skipLocationValidationOnCreateTable.enabled=true
--conf spark.sql.catalog.createDirectoryAfterTable.enabled=true
--conf spark.sql.catalog.dropDirectoryBeforeTable.enabled=true
--conf spark.jars=/usr/lib/hudi/hudi-spark-bundle.jar
--conf spark.sql.extensions=org.apache.spark.sql.hudi.HoodieSparkSessionExtension
--conf
 spark.sql.catalog.spark_catalog=org.apache.spark.sql.hudi.catalog.HoodieCatalog
--conf spark.serializer=org.apache.spark.serializer.KryoSerializer
```

- `spark.hadoop.fs.s3.credentialsResolverClass=com.amazonaws.glue.accesscontrol.A`
  将 EMR 文件系统 (EMRFS) 或 EMR S3A 配置为对 Lake Formation 注册表使用 L AWS ake Formation S3 凭据。如果表未注册，请使用作业的运行时角色凭证。
- `spark.hadoop.fs.s3.useDirectoryHeaderAsFolderObject=true` 和 `spark.hadoop.fs.s3.folderObject.autoAction.disabled=true`：在创建 S3 文件夹时，将 EMRFS 配置为使用内容类型标头应用程序/x 目录，而不是 $folder$ 后缀。这在读取 Lake Formation 表时为必需，因为 Lake Formation 凭证不允许读取带有 $folder$ 后缀的表文件夹。
- `spark.sql.catalog.skipLocationValidationOnCreateTable.enabled=true`：将 Spark 配置为在创建表之前跳过空表位置验证。这对于注册到 Lake Formation 的表是必需的，因为只有在创建 Glue 数据目标表之后，用于验证空位置的 Lake Formation 凭证才会可用。如果未启用此配置，则作业的运行时角色凭证将会验证空表位置。
- `spark.sql.catalog.createDirectoryAfterTable.enabled=true`：将 Spark 配置为在 Hive 元数据存储中创建表后再创建 Amazon S3 文件夹。这对于注册到 Lake Formation 的表为必需，因为只有在创建 Glue Data Catalog 表之后，用于创建 S3 文件夹的 Lake Formation 凭证才会可用。
- `spark.sql.catalog.dropDirectoryBeforeTable.enabled=true`：将 Spark 配置为在 Hive 元数据存储中删除表之前删除 S3 文件夹。这对于注册到 Lake Formation 的表是必需的，因为在从 Glue Data Catalog 中删除表之后，用于删除 S3 文件夹的 Lake Formation 凭证将不可用。

- `spark.sql.catalog.<catalog>.glue.lakeformation-enabled=true`: 将 Iceberg 目录配置为使用 AWS Lake Formation S3 凭据对 Lake Formation 注册的表。如果表未注册，则会使用默认环境凭证。

在 SageMaker 统一工作室中配置全桌访问模式

要通过 JupyterLab 笔记本中的交互式 Spark 会话访问 Lake Formation 注册的表，请使用兼容权限模式。使用 %%configure 魔术命令设置 Spark 配置。根据表类型选择配置：

For Hive tables

```
%%configure -f
{
    "conf": {
        "spark.hadoop.fs.s3.credentialsResolverClass":
 "com.amazonaws.glue.accesscontrol.AWSLakeFormationCredentialResolver",
        "spark.hadoop.fs.s3.useDirectoryHeaderAsFolderObject": true,
        "spark.hadoop.fs.s3.folderObject.autoAction.disabled": true,
        "spark.sql.catalog.skipLocationValidationOnCreateTable.enabled": true,
        "spark.sql.catalog.createDirectoryAfterTable.enabled": true,
        "spark.sql.catalog.dropDirectoryBeforeTable.enabled": true
    }
}
```

For Iceberg tables

```
%%configure -f
{
    "conf": {
        "spark.sql.catalog.spark_catalog":
 "org.apache.iceberg.spark.SparkSessionCatalog",
        "spark.sql.catalog.spark_catalog.warehouse": "S3_DATA_LOCATION",
        "spark.sql.catalog.spark_catalog.client.region": "REGION",
        "spark.sql.catalog.spark_catalog.type": "glue",
        "spark.sql.catalog.spark_catalog.glue.account-id": "ACCOUNT_ID",
        "spark.sql.catalog.spark_catalog.glue.lakeformation-enabled": "true",
        "spark.sql.catalog.dropDirectoryBeforeTable.enabled": "true"
    }
}
```

For Delta Lake tables

```
%%configure -f
{
    "conf": {
        "spark.hadoop.fs.s3.credentialsResolverClass":
 "com.amazonaws.glue.accesscontrol.AWSLakeFormationCredentialResolver",
        "spark.hadoop.fs.s3.useDirectoryHeaderAsFolderObject": true,
        "spark.hadoop.fs.s3.folderObject.autoAction.disabled": true,
        "spark.sql.catalog.skipLocationValidationOnCreateTable.enabled": true,
        "spark.sql.catalog.createDirectoryAfterTable.enabled": true,
        "spark.sql.catalog.dropDirectoryBeforeTable.enabled": true
    }
}
```

For Hudi tables

```
%%configure -f
{
    "conf": {
        "spark.hadoop.fs.s3.credentialsResolverClass":
 "com.amazonaws.glue.accesscontrol.AWSLakeFormationCredentialResolver",
        "spark.hadoop.fs.s3.useDirectoryHeaderAsFolderObject": true,
        "spark.hadoop.fs.s3.folderObject.autoAction.disabled": true,
        "spark.sql.catalog.skipLocationValidationOnCreateTable.enabled": true,
        "spark.sql.catalog.createDirectoryAfterTable.enabled": true,
        "spark.sql.catalog.dropDirectoryBeforeTable.enabled": true,
        "spark.jars": "/usr/lib/hudi/hudi-spark-bundle.jar",
        "spark.sql.extensions":
 "org.apache.spark.sql.hudi.HoodieSparkSessionExtension",
        "spark.sql.catalog.spark_catalog":
 "org.apache.spark.sql.hudi.catalog.HoodieCatalog",
        "spark.serializer": "org.apache.spark.serializer.KryoSerializer"
    }
}
```

替换占位符：

- `S3_DATA_LOCATION`：您的 S3 存储桶路径
- `REGION`: AWS 区域（例如 us-east-1）
- `ACCOUNT_ID`: 您的 AWS 账户 ID

> **Note**
>
> 必须在笔记本中执行任何 Spark 操作之前设置这些配置。

支持的操作

这些操作将使用 AWS Lake Formation 凭据来访问表格数据。

- CREATE TABLE
- ALTER TABLE
- INSERT INTO
- INSERT OVERWRITE
- UPDATE
- MERGE INTO
- DELETE FROM
- ANALYZE TABLE
- REPAIR TABLE
- DROP TABLE
- Spark Datasource 查询
- Spark Datasource 写入

> **Note**
>
> 上面未列出的操作将继续使用 IAM 权限来访问表数据。

注意事项

- 如果使用未启用全表访问的作业创建 Hive 表，并且未插入任何记录，则后续从启用全表访问的作业进行的读取或写入操作都将失败。这是因为未启用全表访问的 EMR Spark 会将 `$folder$` 后缀添加到表文件夹名称。要解决此问题，您可以使用以下任一方法：
  - 从未启用 FTA 的作业在表中至少插入一行。

  - 配置未启用 FTA 的作业，从而确保不在 S3 文件夹名称中使用 `$folder$` 后缀。可以通过设置 Spark 配置 `spark.hadoop.fs.s3.useDirectoryHeaderAsFolderObject=true` 来实现此目的。
  - `s3://path/to/table/table_name`使用 S3 控制台或 S3 CLI 在表格位置创建 AWS S3 文件夹。 AWS
- 从 Amazon EMR 版本 7.8.0 开始，EMR 文件系统 (EMRFS) 支持全表访问，从 Amazon EMR 版本 7.10.0 开始，S3A 文件系统支持全表访问。
- Hive、Iceberg、Delta 和 Hudi 表支持完整表访问权限。
- Hudi FTA Write Support 注意事项：
  - Hudi FTA 写入要求在作业执行期间使用 HoodieCredentialedHadoopStorage 凭证自动售出。运行 Hudi 作业时设置以下配置：`hoodie.storage.class=org.apache.spark.sql.hudi.storage.HoodieCredentialedH`
  - 从 Amazon EMR 7.12 版本开始，Hudi 的全表访问权限 (FTA) 写入支持已推出。
  - Hudi FTA 写入支持目前仅适用于默认 Hudi 配置。自定义或非默认 Hudi 设置可能无法完全支持，并可能导致意外行为。
  - 在 FTA 写入模式下，目前不支持 Hudi Merge-On-Read (MOR) 表的群集。
- 使用 Lake Formation 精细访问控制 (FGAC) 规则或 Glue Data Catalog 视图引用表的作业将会失败。要使用 FGAC 规则或 Glue Data Catalog 视图查询表，您必须使用 FGAC 模式。您可以按照 AWS 文档中概述的步骤启用 FGAC 模式：使用 EMR Serverless 和 Lake Formation 进行精 AWS 细的访问控制。
- 全表访问模式不支持 Spark Streaming。
- 将 Spark DataFrame 写入 Lake Formation 表时，Hive 和 Iceberg 表仅支持 APPEND 模式：`df.write.mode("append").saveAsTable(table_name)`
- 创建外部表需要 IAM 权限。
- 由于 Lake Formation 会在 Spark 作业中临时缓存凭证，因此当前正在运行的 Spark 批处理作业或交互式会话可能无法反映权限更改。
- 您必须使用用户定义的角色而不是服务相关角色：Lake Formation 对角色的要求。

Hudi FTA Write Support——支持的操作

下表显示了 Hudi Copy-On-Write (COW) 和 Merge-On-Read (MOR) 表在 “全表访问” 模式下支持的写入操作：

Hudi FTA 支持的写入操作

表类型	操作	SQL 写入命令	Status
牛	INSERT	INSERT INTO TABLE	支持
牛	INSERT	插入表格-分区（静态、动态）	支持
牛	INSERT	INSERT OVERWRITE	支持
牛	INSERT	插入覆盖-分区（静态、动态）	支持
UPDATE	UPDATE	UPDATE TABLE	支持
牛	UPDATE	更新表-更改分区	不支持
DELETE	DELETE	DELETE FROM TABLE	支持
ALTER	ALTER	更改表-重命名为	不支持
牛	ALTER	更改表格-设置 TBLPROPERTIES	支持
牛	ALTER	更改表格-未设置 TBLPROPERTIES	支持
牛	ALTER	更改表格-更改列	支持
牛	ALTER	更改表格-添加列	支持
牛	ALTER	更改表-添加分区	支持
牛	ALTER	更改表-删除分区	支持

表类型	操作	SQL 写入命令	Status
牛	ALTER	更改表-恢复分区	支持
牛	ALTER	修复表同步分区	支持
DROP	DROP	DROP TABLE	支持
牛	DROP	删除表-清除	支持
CREATE	CREATE	创建表-托管	支持
牛	CREATE	创建表-分区依据	支持
牛	CREATE	如果不存在则创建表	支持
牛	CREATE	CREATE TABLE LIKE	支持
牛	CREATE	CREATE TABLE AS SELECT	支持
CREATE	CREATE	使用位置创建表-外部表	不支持
数据框（插入）	数据框（插入）	saveAsTable. 覆盖	支持
牛	数据框（插入）	saveAsTable.Append	不支持
牛	数据框（插入）	saveAsTable。忽略	支持
牛	数据框（插入）	saveAsTable.ErrorIfExists	支持
牛	数据框（插入）	saveAsTable -外部表（路径）	不支持

表类型	操作	SQL 写入命令	Status
牛	数据框（插入）	保存（路径）-DF v1	不支持
更多	INSERT	INSERT INTO TABLE	支持
更多	INSERT	插入表格-分区（静态、动态）	支持
更多	INSERT	INSERT OVERWRITE	支持
更多	INSERT	插入覆盖-分区（静态、动态）	支持
UPDATE	UPDATE	UPDATE TABLE	支持
更多	UPDATE	更新表-更改分区	不支持
DELETE	DELETE	DELETE FROM TABLE	支持
ALTER	ALTER	更改表-重命名为	不支持
更多	ALTER	更改表格-设置 TBLPROPERTIES	支持
更多	ALTER	更改表格-未设置 TBLPROPERTIES	支持
更多	ALTER	更改表格-更改列	支持
更多	ALTER	更改表格-添加列	支持
更多	ALTER	更改表-添加分区	支持

表类型	操作	SQL 写入命令	Status
更多	ALTER	更改表-删除分区	支持
更多	ALTER	更改表-恢复分区	支持
更多	ALTER	修复表同步分区	支持
DROP	DROP	DROP TABLE	支持
更多	DROP	删除表-清除	支持
CREATE	CREATE	创建表-托管	支持
更多	CREATE	创建表-分区依据	支持
更多	CREATE	如果不存在则创建表	支持
更多	CREATE	CREATE TABLE LIKE	支持
更多	CREATE	CREATE TABLE AS SELECT	支持
CREATE	CREATE	使用位置创建表-外部表	不支持
DATAFRAME（UPSERT）	DATAFRAME（UPSERT）	saveAsTable. 覆盖	支持
更多	DATAFRAME（UPSERT）	saveAsTable.Append	不支持
更多	DATAFRAME（UPSERT）	saveAsTable。忽略	支持
更多	DATAFRAME（UPSERT）	saveAsTable.ErrorIfExists	支持

表类型	操作	SQL 写入命令	Status
更多	DATAFRAME（UPSERT）	saveAsTable -外部表（路径）	不支持
更多	DATAFRAME（UPSERT）	保存（路径）-DF v1	不支持
数据框（删除）	数据框（删除）	saveAsTable.Append	不支持
更多	数据框（删除）	saveAsTable -外部表（路径）	不支持
更多	数据框（删除）	保存（路径）-DF v1	不支持
数据框（批量插入）	数据框（批量插入）	saveAsTable. 覆盖	支持
更多	数据框（批量插入）	saveAsTable.Append	不支持
更多	数据框（批量插入）	saveAsTable。忽略	支持
更多	数据框（批量插入）	saveAsTable.ErrorIfExists	支持
更多	数据框（批量插入）	saveAsTable -外部表（路径）	不支持
更多	数据框（批量插入）	保存（路径）-DF v1	不支持

将 EMR Serverless 与配合使用以实现精细 AWS Lake Formation 的访问控制

概述

在 Amazon EMR 7.2.0 及更高版本中， AWS Lake Formation 可以利用对由 S3 支持的数据目录表应用精细的访问控制。此功能允许您为 Amazon EMR Serverless Spark 作业中的 read 查询配置表、行、列和单元格级别的访问控制。要为 Apache Spark 批处理作业和交互式会话配置精细访问控制，请使用 EMR Studio。请参阅以下部分，了解有关 Lake Formation 以及如何将其与 EMR Serverless 结合使用的更多信息。

使用 Amazon EMR 无服务器会 AWS Lake Formation 产生额外费用。有关更多信息，请参阅 Amazon EMR 定价。

EMR Serverless 是如何使用的 AWS Lake Formation

将 EMR Serverless 与 Lake Formation 结合使用，您可以对每个 Spark 作业强制执行一层权限，以便在 EMR Serverless 执行作业时应用 Lake Formation 权限控制。EMR Serverless 使用 Spark 资源配置文件创建两个配置文件来有效执行作业。用户配置文件执行用户提供的代码，而系统配置文件强制执行 Lake Formation 策略。有关更多信息，请参阅什么是 AWS Lake Formation 以及注意事项和限制。

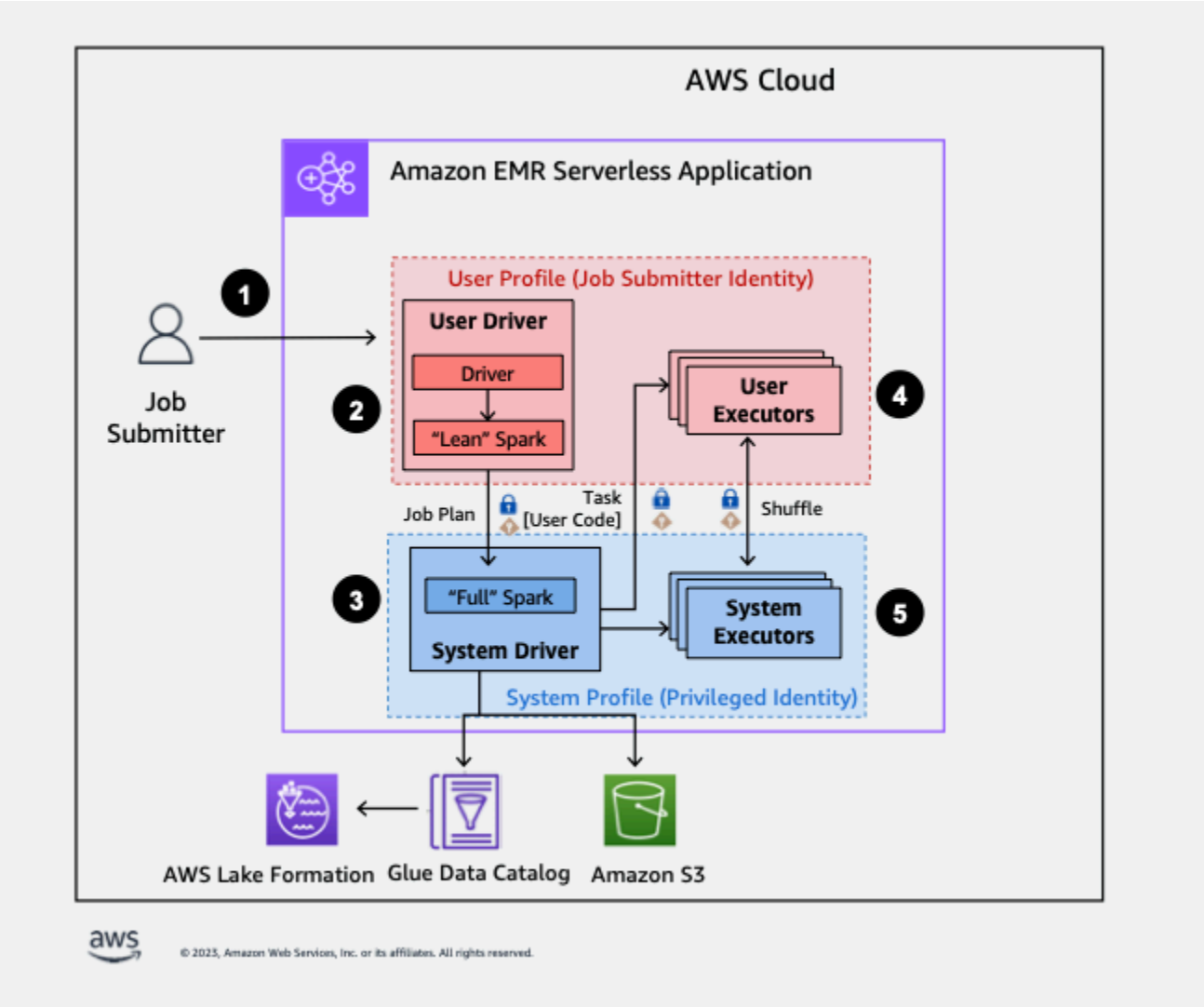
在使用 Lake Formation 的预初始化容量时，我们建议至少使用两个 Spark 驱动程序。每个启用 Lake Formation 的作业都使用两个 Spark 驱动程序，一个用于用户配置文件，一个用于系统配置文件。为了获得最佳性能，与不使用 Lake Formation 相比，支持 Lake Formation 的作业应使用双倍数量的驱动程序。

在 EMR Serverless 上运行 Spark 作业时，还请考虑动态分配对资源管理和集群性能的影响。每个资源配置文件的最大执行程序数的 `spark.dynamicAllocation.maxExecutors` 配置适用于用户和系统执行程序。如果将该数字配置为等于允许的最大执行程序数，则您的作业运行可能会因为一种类型的执行程序使用所有可用资源而卡住，这会在运行作业时阻止其他执行程序。

为避免资源耗尽，EMR Serverless 将每个资源配置文件的默认最大执行程序数量设置为 `spark.dynamicAllocation.maxExecutors` 值的 90%。如果指定 `spark.dynamicAllocation.maxExecutorsRatio` 的值在 0 和 1 之间，可以覆盖此配置。此外，请配置以下属性来优化资源分配和整体性能：

- `spark.dynamicAllocation.cachedExecutorIdleTimeout`
- `spark.dynamicAllocation.shuffleTracking.timeout`
- `spark.cleaner.periodicGC.interval`

下面简要概述了 EMR Serverless 如何访问受 Lake Formation 安全策略保护的数据。



1. 用户将 Spark 作业提交到 AWS Lake Formation启用了 EMR 的无服务器应用程序。
2. EMR Serverless 会将作业发送到用户驱动程序，并在用户配置文件中运行作业。用户驱动程序运行精简版的 Spark，该版本无法启动任务、请求执行程序、访问 S3 或 Glue Catalog。其构建了作业计划。
3. EMR Serverless 设置了第二个驱动程序（称为系统驱动程序），在系统配置文件中运行（使用特权身份）。EMR Serverless 在两个驱动程序之间建立了加密的 TLS 通道来进行通信。用户驱动程序使用该通道将作业计划发送到系统驱动程序。系统驱动程序不会运行用户提交的代码。而是运行完整的 Spark，并与 S3 和数据目录通信，以访问数据。并向执行程序发送请求，将作业计划编译成一系列执行阶段。
4. 然后，EMR Serverless 使用用户驱动程序或系统驱动程序在执行程序上运行这些阶段。任何阶段的用户代码都只能在用户配置文件执行程序上运行。

5. 从受安全筛选器保护的数据目录表中读取数据的阶段 AWS Lake Formation 或应用安全筛选器的阶段将委托给系统执行者。

在 Amazon EMR 中启用 Lake Formation

要启用 Lake Formation，请在创建 EMR Serverless 应用程序时，在运行时配置参数的 `spark-defaults` 分类下将 `spark.emr-serverless.lakeformation.enabled` 设置为 `true`。

```
aws emr-serverless create-application \
    --release-label emr-7.12.0 \
    --runtime-configuration '{
     "classification": "spark-defaults",
     "properties": {
      "spark.emr-serverless.lakeformation.enabled": "true"
      }
    }' \
    --type "SPARK"
```

您还可以在 EMR Studio 中创建新应用程序时启用 Lake Formation。在其他配置下，选择使用 Lake Formation 进行精细访问控制。

当您将 Lake Formation 与 EMR Serverless 结合使用时，会默认启用工作线程间加密，因此您无需再显式启用工作线程间加密。

为 Spark 作业启用 Lake Formation

要为单个 Spark 作业启用 Lake Formation，请在使用 `spark-submit` 时将 `spark.emr-serverless.lakeformation.enabled` 设置为 true。

```
--conf spark.emr-serverless.lakeformation.enabled=true
```

作业运行时角色 IAM 权限

Lake For AWS mation 权限控制对 Glue 数据目录资源、Amazon S3 位置以及这些位置的基础数据的访问权限。IAM 权限控制对 Lake Formation 和 AWS Glue APIs 以及资源的访问。虽然您可能拥有 Lake Formation 权限来访问数据目录（SELECT）中的表，但如果没有对 `glue:Get*` API 操作的 IAM 权限，操作就会失败。

下面是一个策略示例，展示了如何提供 IAM 权限以访问 S3 中的脚本、将日志上传到 S3、 AWS Glue API 权限以及访问 Lake Formation 的权限。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "ScriptAccess",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::*.amzn-s3-demo-bucket/scripts",
        "arn:aws:s3:::*.amzn-s3-demo-bucket/*"
      ]
    },
    {
      "Sid": "LoggingAccess",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket/logs/*"
      ]
    },
    {
      "Sid": "GlueCatalogAccess",
      "Effect": "Allow",
      "Action": [
        "glue:Get*",
        "glue:Create*",
        "glue:Update*"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "LakeFormationAccess",
      "Effect": "Allow",
      "Action": [
```

```
        "lakeformation:GetDataAccess"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

设置 Lake Formation 的作业运行时角色权限

首先，在 Lake Formation 中注册 Hive 表的位置。然后在所需的表上创建作业运行时角色的权限。有关 Lake Formation 的更多详情，请参阅什么是 AWS Lake Formation？ 在《AWS Lake Formation 开发人员指南》中。

设置 Lake Formation 权限后，请在 Amazon EMR Serverless 上提交 Spark 作业。有关 Spark 作业的更多信息，请参阅 Spark 示例。

提交作业运行

设置 Lake Formation 授权后，便可以在 Amazon EMR Serverless 上提交 Spark 作业。以下部分展示了如何配置和提交作业运行属性的示例。

权限要求

未在中注册的表 AWS Lake Formation

对于未向注册的表 AWS Lake Formation，作业运行时角色同时访问 Glue 数据 AWS 目录和 Amazon S3 中的基础表数据。这要求任务运行时角色拥有相应的 IAM 权限，可同时执行 Glue AWS 和 Amazon S3 操作。

在中注册的表 AWS Lake Formation

对于注册到的表 AWS Lake Formation，作业运行时角色访问 AWS Glue 数据目录元数据，而 Lake Formation 提供的临时证书则访问 Amazon S3 中的基础表数据。执行操作所需的 Lake For AWS mation 权限取决于 Spark 任务启动的 Glue 数据目录和 Amazon S3 API 调用，可以总结如下：

- D@@ ES CRIBE 权限允许运行时角色读取数据目录中的表或数据库元数据
- ALTER 权限允许运行时角色修改数据目录中的表或数据库元数据

- DROP 权限允许运行时角色从数据目录中删除表或数据库元数据

- SE@@ LECT 权限允许运行时角色从 Amazon S3 读取表数据
- INS@@ ERT 权限允许运行时角色将表数据写入 Amazon S3
- 删除权限允许运行时角色从 Amazon S3 中删除表数据

> **Note**
>
> 当 Spark 任务调用 G AWS lue 来检索表元数据并调用 Amazon S3 来检索表数据时，Lake Formation 会延迟评估权限。在 Spark 发出需要缺少权限的 AWS Glue 或 Amazon S3 调用之前，使用权限不足的运行时角色的任务不会失败。

> **Note**
>
> 在以下支持的表格矩阵中：
>
> - 标记为 “支持” 的操作仅使用 Lake Formation 凭据来访问在 Lake Formation 中注册的表的表数据。如果 Lake Formation 权限不足，则操作将不会回退到运行时角色证书。对于未在 Lake Formation 中注册的表，作业运行时角色凭据可以访问表数据。
> - 在 A mazon S3 位置上标记为 “支持” 且具有 IAM 权限的操作不会使用 Lake Formation 凭证访问亚马逊 S3 中的基础表数据。要运行这些操作，无论表是否已在 Lake Formation 中注册，任务运行时角色都必须具有访问表数据所必需的 Amazon S3 IAM 权限。

Hive

操作	AWS Lake Formation 权限	Support 状态
SELECT	SELECT	支持
CREATE TABLE	创建表	支持
CREATE TABLE LIKE	创建表	支持 Amazon S3 位置的 IAM 权限
CREATE TABLE AS SELECT	创建表	支持 Amazon S3 位置的 IAM 权限

操作	AWS Lake Formation 权限	Support 状态
DESCRIBE TABLE	DESCRIBE	支持
SHOW TBLPROPERTIES	DESCRIBE	支持
SHOW COLUMNS	DESCRIBE	支持
SHOW PARTITIONS	DESCRIBE	支持
SHOW CREATE TABLE	DESCRIBE	支持
更改表格 `tablename`	选择并更改	支持
更改餐桌`tablename` 布置位置	-	不支持
修改表`tablename` 添加分区	选择、插入和更改	支持
REPAIR TABLE	选择并更改	支持
加载数据		不支持
INSERT	插入和更改	支持
INSERT OVERWRITE	选择、插入、删除和更改	支持
DROP TABLE	选择、删除、删除和更改	支持
TRUNCATE TABLE	选择、插入、删除和更改	支持
DataFrame Writer V1	与相应的 SQL 操作相同	向现有表追加数据时支持。有关更多信息，请参阅注意事项和限制
DataFrame Writer V2	与相应的 SQL 操作相同	向现有表追加数据时支持。有关更多信息，请参阅注意事项和限制

Iceberg

操作	AWS Lake Formation 权限	Support 状态
SELECT	SELECT	支持
CREATE TABLE	创建表	支持
CREATE TABLE LIKE	创建表	支持 Amazon S3 位置的 IAM 权限
CREATE TABLE AS SELECT	创建表	支持 Amazon S3 位置的 IAM 权限
将表格替换为选定内容	选择、插入和更改	支持
DESCRIBE TABLE	DESCRIBE	支持 Amazon S3 位置的 IAM 权限
SHOW TBLPROPERTIES	DESCRIBE	支持 Amazon S3 位置的 IAM 权限
SHOW CREATE TABLE	DESCRIBE	支持 Amazon S3 位置的 IAM 权限
ALTER TABLE	选择、插入和更改	支持
ALTER TABLE SET LOCATION	选择、插入和更改	支持 Amazon S3 位置的 IAM 权限
更改表写入顺序依据	选择、插入和更改	支持 Amazon S3 位置的 IAM 权限
更改表写入分发者	选择、插入和更改	支持 Amazon S3 位置的 IAM 权限
更改表重命名表	CREATE_TABLE，然后删除	支持
INSERT INTO	选择、插入和更改	支持
INSERT OVERWRITE	选择、插入和更改	支持
DELETE	选择、插入和更改	支持

操作	AWS Lake Formation 权限	Support 状态
UPDATE	选择、插入和更改	支持
MERGE INTO	选择、插入和更改	支持
DROP TABLE	选择、删除和删除	支持
DataFrame Writer V1	-	不支持
DataFrame Writer V2	与相应的 SQL 操作相同	向现有表追加数据时支持。有关更多信息，请参阅注意事项和限制。
元数据表	SELECT	支持。某些表格是隐藏的。有关更多信息，请参阅注意事项和限制。
存储过程	-	支持满足以下条件的表： • 未在中注册的表 AWS Lake Formation • 不使用`register_table` 和的表 `migrate` 有关更多信息，请参阅注意事项和限制。

Iceberg 的 Spark 配置：以下示例展示了如何使用 Iceberg 配置 Spark。要运行 Iceberg 作业，请提供以下 `spark-submit` 属性。

```
--conf spark.sql.catalog.spark_catalog=org.apache.iceberg.spark.SparkSessionCatalog
--conf spark.sql.catalog.spark_catalog.warehouse=<S3_DATA_LOCATION>
--conf spark.sql.catalog.spark_catalog.glue.account-id=<ACCOUNT_ID>
--conf spark.sql.catalog.spark_catalog.client.region=<REGION>
--conf spark.sql.catalog.spark_catalog.glue.endpoint=https://
glue.<REGION>.amazonaws.com
```

Hudi

操作	AWS Lake Formation 权限	Support 状态
SELECT	SELECT	支持
CREATE TABLE	创建表	支持 Amazon S3 位置的 IAM 权限
CREATE TABLE LIKE	创建表	支持 Amazon S3 位置的 IAM 权限
CREATE TABLE AS SELECT	-	不支持
DESCRIBE TABLE	DESCRIBE	支持 Amazon S3 位置的 IAM 权限
SHOW TBLPROPERTIES	DESCRIBE	支持 Amazon S3 位置的 IAM 权限
SHOW COLUMNS	DESCRIBE	支持 Amazon S3 位置的 IAM 权限
SHOW CREATE TABLE	DESCRIBE	支持 Amazon S3 位置的 IAM 权限
ALTER TABLE	SELECT	支持 Amazon S3 位置的 IAM 权限
INSERT INTO	选择并更改	支持 Amazon S3 位置的 IAM 权限
INSERT OVERWRITE	选择并更改	支持 Amazon S3 位置的 IAM 权限
DELETE	-	不支持
UPDATE	-	不支持

操作	AWS Lake Formation 权限	Support 状态
MERGE INTO	-	不支持
DROP TABLE	选择并删除	支持 Amazon S3 位置的 IAM 权限
DataFrame Writer V1	-	不支持
DataFrame Writer V2	与相应的 SQL 操作相同	支持 Amazon S3 位置的 IAM 权限
元数据表	-	不支持
表维护和实用程序功能	-	不支持

以下示例使用 Hudi 配置 Spark，指定文件位置和使用所需的其他属性。

Hudi 的 Spark 配置：在笔记本中使用时，此代码片段指定 Hudi Spark 捆绑 JAR 文件的路径，从而在 Spark 中启用 Hudi 功能。它还将 Spark 配置为使用 AWS Glue 数据目录作为元数据库。

```
%%configure -f
{
    "conf": {
        "spark.jars": "/usr/lib/hudi/hudi-spark-bundle.jar",
        "spark.hadoop.hive.metastore.client.factory.class":
 "com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveClientFactory",
        "spark.serializer": "org.apache.spark.serializer.JavaSerializer",
        "spark.sql.catalog.spark_catalog":
 "org.apache.spark.sql.hudi.catalog.HoodieCatalog",
        "spark.sql.extensions":
 "org.apache.spark.sql.hudi.HoodieSparkSessionExtension"
    }
}
```

带有 AWS Glue 的 Hudi 的 Spark 配置：在笔记本中使用此片段时，Hudi 可以作为支持的数据湖格式，并确保 Hudi 库和依赖项可用。

```
%%configure
{
```

```
    "--conf": "spark.serializer=org.apache.spark.serializer.JavaSerializer --conf
spark.sql.catalog.spark_catalog=org.apache.spark.sql.hudi.catalog.HoodieCatalog --
conf
spark.sql.extensions=org.apache.spark.sql.hudi.HoodieSparkSessionExtension",
    "--datalake-formats": "hudi",
    "--enable-glue-datacatalog": True,
    "--enable-lakeformation-fine-grained-access": "true"
}
```

Delta Lake

操作	AWS Lake Formation 权限	Support 状态
SELECT	SELECT	支持
CREATE TABLE	创建表	支持
CREATE TABLE LIKE	-	不支持
CREATE TABLE AS SELECT	创建表	支持
将表格替换为选定内容	选择、插入和更改	支持
DESCRIBE TABLE	DESCRIBE	支持 Amazon S3 位置的 IAM 权限
SHOW TBLPROPERTIES	DESCRIBE	支持 Amazon S3 位置的 IAM 权限
SHOW COLUMNS	DESCRIBE	支持 Amazon S3 位置的 IAM 权限
SHOW CREATE TABLE	DESCRIBE	支持 Amazon S3 位置的 IAM 权限
ALTER TABLE	选择并插入	支持
ALTER TABLE SET LOCATION	选择并插入	支持 Amazon S3 位置的 IAM 权限

操作	AWS Lake Formation 权限	Support 状态
按以下方式更改表tablename 集群	选择并插入	支持 Amazon S3 位置的 IAM 权限
更改表tablename 添加约束	选择并插入	支持 Amazon S3 位置的 IAM 权限
更改表格tablename 删除约束	选择并插入	支持 Amazon S3 位置的 IAM 权限
INSERT INTO	选择并插入	支持
INSERT OVERWRITE	选择并插入	支持
DELETE	选择并插入	支持
UPDATE	选择并插入	支持
MERGE INTO	选择并插入	支持
DROP TABLE	选择、删除和删除	支持
DataFrame Writer V1	-	不支持
DataFrame Writer V2	与相应的 SQL 操作相同	支持
表维护和实用程序功能	-	不支持

EMR Serverless 与 Delta Lake：要在 EMR Serverless 上使用 Delta Lake 和 Lake Formation，请运行以下命令：

```
spark-sql \
  --conf
 spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension,com.amazonaws.emr.recordserver
 \
  --conf
 spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog \
```

调试作业

> **Note**
>
> 借助此功能，可以访问系统配置文件工作线程的 stdout 和 stderr 日志，其中可能包含未经筛选的敏感信息。以下权限仅应用于访问非生产数据。对于为用于生产作业而创建的应用程序，我们强烈建议您仅向管理员或具有更高数据访问权限的用户添加这些权限。

在 EMR-7.3.0 及更高版本中，EMR Serverless 为启用 Lake Formation 的批处理作业启用了自调试功能。为此，请使用 GetDashboardForJobRunAPI 中的新accessSystemProfile参数 Logs。如果“accessSystemProfile日志” 设置为 true，则可以访问系统配置文件工作人员的 stdout 和 stderr 日志，这些日志可用于调试启用 Lake Formation 的 EMR Serverless 批处理作业。

```
aws emr-serverless get-dashboard-for-job-run \
  --application-id application-id
  --job-run-id job-run-id
  --access-system-profile-logs
```

所需的权限

想要使用调试启用 Lake Formation 的批处理作业的委托人GetDashboardForJobRun必须具有以下额外权限：

```
{
    "Sid": "AccessSystemProfileLogs",
    "Effect": "Allow",
    "Action": [
        "emr-serverless:GetDashboardForJobRun",
        "emr-serverless:AccessSystemProfileLogs",
        "glue:GetDatabases",
        "glue:SearchTables"
    ],
    "Resource": [
        "arn:aws:emr-serverless:region:account-id:/applications/applicationId/
jobruns/jobid",
        "arn:aws:glue:region:account-id:catalog",
        "arn:aws:glue:region:account-id:database/*",
        "arn:aws:glue:region:account-id:table/*/*"
    ]
```

```
}
```

注意事项

对于使用与作业相同的账户访问 Lake Formation 中的数据库或表的作业，可以查看用于调试的系统配置文件日志。在以下情况下，它们不可见：

- 如果使用 Lake Formation 权限管理的数据目录具有跨账户数据库和表
- 如果使用 Lake Formation 权限管理的数据目录具有资源链接

使用 Glue Data Catalog 视图

您可以在 AWS Glue 数据目录中创建和管理视图，以便与 EMR Serverless 配合使用。这些视图通常被称为 AWS Glue 数据目录视图。这些视图之所以有用，是因为它们支持多个 SQL 查询引擎，因此您可以跨不同的 AWS 服务（例如 EMR Serverless 和 Amazon Redshift）访问相同的视图。 Amazon Athena

通过在数据目录中创建视图，在中使用资源授予和基于标签的访问控制 AWS Lake Formation 来授予对该视图的访问权限。使用这种访问控制方法，您无需为创建视图时引用的表配置其他访问权限。这种授予权限的方法称为定义者语义，这些视图称为定义者视图。有关 Lake Formation 中访问控制的更多信息，请参阅《Lake Formation 开发者指南》中的授予和撤消数据目录资源的权限。 AWS

数据目录视图对于以下用例非常实用：

- 精细访问控制：您可以创建一个视图，根据用户所需的权限来限制数据访问。例如，您可以使用 Data Catalog 中的视图来防止不在 HR 部门工作的员工查看个人身份信息 (PII)。
- 完整视图定义：通过对 Data Catalog 中的视图应用筛选条件，可确保 Data Catalog 中视图内的数据记录始终完整。
- 增强安全性：用于创建视图的查询定义必须完整。这种优势意味着 Data Catalog 中的视图不容易受到恶意行为者的 SQL 命令的影响。
- 简单共享数据-无需移动数据即可与其他 AWS 账户共享数据。有关更多信息，请参阅 Lake Formation 中的跨账户数据共享。

创建 Data Catalog 视图

创建 Data Catalog 视图的方法有很多种。其中包括使用 AWS CLI 或 Spark SQL。以下是一些示例。

Using SQL

下面展示了创建 Data Catalog 视图的语法。注意 MULTI DIALECT 视图类型。这将 Data Catalog 视图与其他视图区分开来。SECURITY 谓词指定为 DEFINER。这表示带有 DEFINER 语义的 Data Catalog 视图。

```
CREATE [ OR REPLACE ] PROTECTED MULTI DIALECT VIEW [IF NOT EXISTS] view_name
[(column_name [COMMENT column_comment], ...) ]
[ COMMENT view_comment ]
[TBLPROPERTIES (property_name = property_value, ... )]
SECURITY DEFINER
AS query;
```

以下是 CREATE 语句示例，其语法如下：

```
CREATE PROTECTED MULTI DIALECT VIEW catalog_view
SECURITY DEFINER
AS
SELECT order_date, sum(totalprice) AS price
FROM source_table
GROUP BY order_date
```

您还可以使用 SQL 在试运行模式下创建视图以测试视图创建，而无需实际创建资源。使用此选项会导致“试运行”，该试运行可以验证输入，如果验证成功，则返回将代表视图的 Glue AWS 表对象的 JSON。在这种情况下，不会创建实际视图。

```
CREATE [ OR REPLACE ] PROTECTED MULTI DIALECT VIEW view_name
SECURITY DEFINER
[ SHOW VIEW JSON ]
AS view-sql
```

Using the AWS CLI

> **Note**
>
> 使用 CLI 命令时，不会解析用于创建视图的 SQL。这可能会导致创建视图，但查询不成功。创建视图之前，请务必测试 SQL 语法。

您可以使用以下 CLI 命令创建视图：

```
aws glue create-table --cli-input-json '{
  "DatabaseName": "database",
  "TableInput": {
    "Name": "view",
    "StorageDescriptor": {
      "Columns": [
        {
          "Name": "col1",
          "Type": "data-type"
        },
        ...
        {
          "Name": "col_n",
          "Type": "data-type"
        }
      ],
      "SerdeInfo": {}
    },
    "ViewDefinition": {
      "SubObjects": [
        "arn:aws:glue:aws-region:aws-account-id:table/database/referenced-table1",
        ...
        "arn:aws:glue:aws-region:aws-account-id:table/database/referenced-tableN",
       ],
      "IsProtected": true,
      "Representations": [
        {
          "Dialect": "SPARK",
          "DialectVersion": "1.0",
          "ViewOriginalText": "Spark-SQL",
          "ViewExpandedText": "Spark-SQL"
        }
      ]
    }
  }
}'
```

支持的视图操作

以下命令片段展示了使用 Data Catalog 视图的各种方法：

- CREATE VIEW

创建数据目录视图。以下示例演示如何根据现有表格创建视图：

```
CREATE PROTECTED MULTI DIALECT VIEW catalog_view
SECURITY DEFINER AS SELECT * FROM my_catalog.my_database.source_table
```

- ALTER VIEW

  可用语法：

  - `ALTER VIEW view_name [FORCE] ADD DIALECT AS query`
  - `ALTER VIEW view_name [FORCE] UPDATE DIALECT AS query`
  - `ALTER VIEW view_name DROP DIALECT`

  您可以使用 `FORCE ADD DIALECT` 选项，根据新的引擎方言强制更新架构和子对象。请注意，如果不同时使用 `FORCE` 更新其他引擎方言，这样做可能会导致查询错误。下面展示了一个示例：

  ```
  ALTER VIEW catalog_view FORCE ADD DIALECT
  AS
  SELECT order_date, sum(totalprice) AS price
  FROM source_table
  GROUP BY orderdate;
  ```

  下面展示了如何更改视图来更新方言：

  ```
  ALTER VIEW catalog_view UPDATE DIALECT AS
  SELECT count(*) FROM my_catalog.my_database.source_table;
  ```

- DESCRIBE VIEW

  描述视图的可用语法：

  - `SHOW COLUMNS {FROM|IN} view_name [{FROM|IN} database_name]`— 如果用户拥有描述视图所需的 AWS Glue 和 Lake Formation 权限，则他们可以列出这些列。下面展示了几个用于显示列的示例命令：

    ```
    SHOW COLUMNS FROM my_database.source_table;
    SHOW COLUMNS IN my_database.source_table;
    ```

  - `DESCRIBE view_name`— 如果用户具有描述视图所需的 AWS Glue 和 Lake Formation 权限，则他们可以列出视图中的列及其元数据。

- DROP VIEW

  可用语法：

  - `DROP VIEW [ IF EXISTS ] view_name`

    以下示例显示了 DROP 语句，该语句用于在删除视图之前测试视图是否存在：

    ```
    DROP VIEW IF EXISTS catalog_view;
    ```

- 显示创建视图

  - `SHOW CREATE VIEW view_name`：显示创建指定视图的 SQL 语句。以下示例演示如何创建数据目录视图：

    ```
    SHOW CREATE TABLE my_database.catalog_view;
    CREATE PROTECTED MULTI DIALECT VIEW my_catalog.my_database.catalog_view (
      net_profit,
      customer_id,
      item_id,
      sold_date)
    TBLPROPERTIES (
      'transient_lastDdlTime' = '1736267222')
    SECURITY DEFINER AS SELECT * FROM
    my_database.store_sales_partitioned_lf WHERE customer_id IN (SELECT customer_id
     from source_table limit 10)
    ```

- SHOW VIEWS

  列出目录中的所有视图，例如常规视图、多方言视图 (MDV) 和没有 Spark 方言的 MDV。可用语法如下：

  - `SHOW VIEWS [{ FROM | IN } database_name] [LIKE regex_pattern]`:

    下面展示了用于显示视图的示例命令：

    ```
    SHOW VIEWS IN marketing_analytics LIKE 'catalog_view*';
    ```

有关创建和配置数据目录视图的更多信息，请参阅《 AWS Lake Formation 开发人员指南》中的 B AWS uilding Glue 数据目录视图。

查询 Data Catalog 视图

创建数据目录视图后，您可以使用启用了 AWS Lake Formation 精细访问控制的 Amazon EMR Serverless Spark 任务对其进行查询。作业运行时角色必须拥有对 Data Catalog 视图的 Lake Formation SELECT 权限。您无需授予对视图中引用的基础表的访问权限。

完成所有设置后，就可以查询视图。例如，在 EMR Studio 中创建 EMR Serverless 应用程序后，请运行以下查询来访问视图。

```
SELECT * from my_database.catalog_view LIMIT 10;
```

一个有用的功能是 invoker_principal。它返回 EMRS 作业运行时角色的唯一标识符。这可用于根据调用主体控制视图输出。您可以使用它在视图中添加一个条件，以便根据调用角色优化查询结果。作业运行时角色必须拥有执行 LakeFormation:GetDataLakePrincipal IAM 操作的权限才能使用此函数。

```
select invoker_principal();
```

例如，您可以将此函数添加到 WHERE 子句以优化查询结果。

注意事项和限制

创建 Data Catalog 视图时，以下内容适用：

- 只能使用 Amazon EMR 7.6 及更高版本创建 Data Catalog 视图。
- Data Catalog 视图定义者必须拥有对视图访问的基础基表的 SELECT 访问权限。如果特定基表对定义者角色施加了任何 Lake Formation 筛选条件，则创建 Data Catalog 视图将失败。
- 在 Lake Formation 中，基表不得具有 IAMAllowedPrincipals 数据湖权限。如果存在，则会出现错误 “多方言视图” 只能引用没有 IAMAllowed委托人权限的表。
- 该表的 Amazon S3 位置必须注册为 Lake Formation 数据湖位置。如果该表未注册，则会出现错误多方言视图只能引用 Lake Formation 托管表。有关如何在 Lake Formation 中注册亚马逊 S3 营业地点的信息，请参阅 AWS Lake Formation 开发者指南中的注册亚马逊 S3 营业地点。
- 只能创建 PROTECTED 数据目录视图。不支持 UNPROTECTED 视图。
- 您不能在数据目录视图定义中引用其他 AWS 账户中的表。也不能引用不同区域中同一账户中的表。
- 要跨账户或区域共享数据，必须使用 Lake Formation 资源链接跨账户和跨区域共享整个视图。
- 不支持用户定义的函数 (UDFs)。

- 您可以使用基于 Iceberg 表的视图。还支持开放式表格格式 Apache Hudi 和 Delta Lake。
- 不能在数据目录视图中引用其他视图。
- AWS Glue 数据目录视图架构始终使用小写形式存储。例如，如果使用 DDL 语句创建包含名为 `Castle` 的列的 Glue Data Catalog 视图，则在 Glue Data Catalog 中创建的列将被小写为 `castle`。如果您随后将 DML 查询中的列名指定为 `Castle` 或 `CASTLE`，EMR Spark 会将名称转换为小写，以便您运行查询。但是列标题会使用您在查询中指定的大小写形式显示。

  如果您希望在 DML 查询中指定的列名与 Glue Data Catalog 中的列名不匹配时查询失败，请设置 `spark.sql.caseSensitive=true`。

开放表格式支持

EMR Serverless 支持 Apache Hive、Apache Iceberg、Delta Lake（7.6.0+）和 Apache Hudi（7.6.0+）上的 SELECT 查询。从 EMR 7.12 开始，Apache Hive、Apache Iceberg 和 Delta Lake 表支持使用 Lake Formation 提供的凭据修改表数据的 DML 和 DDL 操作。

注意事项和限制

General

在 EMR Serverless 中使用 Lake Formation 时，请查看以下限制。

> **Note**
>
> 在 EMR Serverless 上为 Spark 作业启用 Lake Formation 时，作业会启动系统驱动程序和用户驱动程序。如果在启动时指定了预初始化容量，则会从预初始化容量中预置驱动程序，系统驱动程序的数量与指定的用户驱动程序的数量相等。如果选择按需容量，EMR Serverless 除了会启动用户驱动程序外，还会启动系统驱动程序。要估算在 Lake Formation 作业中使用 EMR Serverless 相关的成本，请使用 AWS 定价计算器。

- 启用 Lake Formation 的 Amazon EMR Serverless 适用于所有受支持的 EMR Serverless 区域。
- 启用 Lake Formation 的应用程序不支持使用自定义 EMR Serverless 映像。
- 您不能为 Lake Formation 作业关闭 `DynamicResourceAllocation`。
- 您只能将 Lake Formation 与 Spark 作业结合使用。

- 启用 Lake Formation 的 EMR Serverless 在整个作业中仅支持单个 Spark 会话。
- 启用 Lake Formation 的 EMR Serverless 仅支持通过资源链接共享的跨账户表查询。
- 不支持以下项：
  - 弹性分布式数据集（RDD）
  - Spark 流
  - 嵌套列的访问控制
- EMR Serverless 会阻止可能破坏系统驱动程序完全隔离的功能，包括：
  - UDTs、Hive UDFs 以及任何涉及自定义类的用户定义函数
  - 自定义数据来源
  - 为 Spark 扩展、连接器或元存储提供额外的 jar
  - `ANALYZE TABLE` 命令
- 如果您的 EMR Serverless 应用程序位于具有适用于 Amazon S3 的 VPC 终端节点的私有子网中，并且您附加了终端节点策略来控制访问权限，则在您的任务可以将日志数据发送到托管 A AWS mazon S3 之前，请在您的 VPC 策略中将托管存储中详述的权限添加到 S3 网关终端节点。如需疑难解答请求，请联系 AWS 支持人员。
- 从 Amazon EMR 7.9.0 开始，Spark FGAC 在与 s3a://方案一起使用时支持 S3 AFile 系统。
- 亚马逊 EMR 7.11 支持使用 CTAS 创建托管表。
- Amazon EMR 7.12 支持使用 CTAS 创建托管表和外部表。

Permissions

- 为了强制执行访问控制，EXPLAIN PLAN 和 DDL 操作（例如 DESCRIBE TABLE）不会泄露受限信息。
- 当您向 Lake Formation 注册表位置时，数据访问将使用 Lake Formation 存储的证书，而不是 EMR Serverless 作业运行时角色的 IAM 权限。如果表位置的注册角色配置错误，即使运行时角色对该位置拥有 S3 IAM 权限，任务也会失败。
- 从 Amazon EMR 7.12 开始，你可以在追加模式下使用 DataFrameWriter (V2) 和 Lake Formation 凭证写入现有 Hive 和 Iceberg 表。对于覆盖操作或创建新表时，EMR 使用运行时角色凭据来修改表数据。
- 使用视图或缓存表作为源数据时，存在以下限制（这些限制不适用于 AWS Glue 数据目录视图）：
  - 用于合并、删除和更新操作
    - 支持：使用视图和缓存表作为源表。

    - 不支持：在赋值和条件子句中使用视图和缓存表。
  - 对于创建或替换和替换表作为选择操作：
    - 不支持：使用视图和缓存表作为源表。
- 仅当启用删除向量时，包含 UDFs 源数据的 Delta Lake 表才支持合并、删除和更新操作。

日志和调试

- EMR Serverless 限制访问启用 Lake Formation 的应用程序上的系统驱动程序 Spark 日志。由于系统驱动程序以提升的权限运行，因此系统驱动程序生成的事件和日志可能包含敏感信息。为防止未经授权的用户或代码访问此敏感数据，EMR Serverless 禁止访问系统驱动程序日志。
- 系统配置文件日志始终保存在托管存储中：这是一项强制性设置，无法禁用。这些日志使用客户托管 KMS 密钥或托管 KMS 密钥 AWS 进行安全存储和加密。

Iceberg

使用 Apache Iceberg 时，请查看以下注意事项：

- 您只能在会话目录中使用 Apache Iceberg，而不能使用任意命名的目录。
- 在 Lake Formation 中注册的 Iceberg 表仅支持元数据表 `history`、`metadata_log_entries`、`snapshots`、`files`、`manifests` 和 `refs`。Amazon EMR 会隐藏可能包含敏感数据的列，例如 `partitions`、`path` 和 `summaries`。此限制不适用于未在 Lake Formation 中注册的 Iceberg 表。
- 未在 Lake Formation 中注册的表支持所有 Iceberg 存储过程。任何表都不支持 `register_table` 和 `migrate` 程序。
- 我们建议你使用 Iceberg DataFrameWriter V2 而不是 V1。

问题排查

有关故障排除解决方案，请参阅以下部分：

日志记录

EMR Serverless 使用 Spark 资源配置文件来拆分作业执行。EMR Serverless 使用用户配置文件来运行您提供的代码，而系统配置文件则强制执行 Lake Formation 策略。您可以访问作为用户配置文件运行的任务的日志。

有关调试启用 Lake Formation 的作业的更多信息，请参阅调试作业。

Live UI 和 Spark History Server

Live UI 和 Spark History Server 包含从用户配置文件生成的所有 Spark 事件以及从系统驱动程序生成的编辑事件。

您可以在执行程序选项卡中查看用户和系统驱动程序中的所有任务。但日志链接仅适用于用户配置文件。此外，还会从 Live UI 中编辑一些信息，例如输出记录数。

由于 Lake Formation 权限不足，作业失败

确保您的作业运行时角色有权在您要访问的表上运行 SELECT 和 DESCRIBE。

RDD 作业执行失败

EMR Serverless 目前不支持在启用了 Lake Formation 的作业上进行弹性分布式数据集（RDD）操作。

无法访问 Amazon S3 中的数据文件

确保您已在 Lake Formation 中注册数据湖的位置。

安全验证异常

EMR Serverless 检测到安全验证错误。请联系 AWS 支持人员寻求帮助。

跨账户 AWS 共享 Glue 数据目录和表格

您可以跨账户共享数据库和表，且仍可使用 Lake Formation。有关更多信息，请参阅 Lake Formation 中的跨账户数据共享和如何使用跨账户共享 AWS Glue 数据目录和表格？ AWS Lake Formation。

Spark 原生精细访问控制允许列入名单 API PySpark

为了维护安全性和数据访问控制，Spark 精细访问控制 (FGAC) 限制了某些功能。 PySpark 这些限制是通过以下方式强制执行的：

- 用于阻止函数执行的显式阻塞
- 使函数无法运行的架构不兼容
- 可能引发错误、返回被拒绝访问的消息或在调用时什么都不做的函数

Spark FGAC 不支持以下 PySpark 功能：

- RDD 操作（被 Spark RDDUnsupported 异常阻止）
- Spark Connect（不支持）
- Spark 直播（不支持）

虽然我们已经在 Native Spark FGAC 环境中测试了列出的函数并确认它们可以按预期运行，但我们的测试通常仅涵盖每个 API 的基本用法。具有多种输入类型或复杂逻辑路径的函数可能有未经测试的场景。

对于此处未列出且不明显属于上述不支持的类别的任何函数，我们建议：

- 首先在 gamma 环境或小规模部署中对其进行测试
- 在生产中使用它们之前对其行为进行验证

> **Note**
>
> 如果您看到列出了一个类方法但没有列出其基类，则该方法应该仍然有效，这只是意味着我们尚未明确验证基类构造函数。

PySpark API 被组织成多个模块。下表详细说明了对每个模块中方法的普遍支持。

模块名称	Status	注意
pystark_core	支持	该模块包含主要的 RDD 类，这些函数大多不受支持。
pystark_sql	支持	
pyspark_tes	支持	
pyspark_res	支持	
pyspark_stre	阻止	Spark FGAC 中已禁止使用直播功能。

模块名称	Status	注意
pyspark_mllib	实验性的	该模块包含基于 RDD 的 ML 操作，这些函数大多不受支持。此模块未经过全面测试。
pyspark_ml	实验性的	该模块包含 DataFrame 基于机器学习的操作，这些函数大多受支持。此模块未经过全面测试。
pypark_pandas	支持	
pyspark_pandas_low	支持	
pyspark_con	阻止	Spark FGAC 中禁止使用 Spark Connect。
pyspark_pandas_conn	阻止	Spark FGAC 中禁止使用 Spark Connect。
pyspark_pandas_slow_conn	阻止	Spark FGAC 中禁止使用 Spark Connect。
pyspark_erro	实验性的	此模块未经过全面测试。无法使用自定义错误类。

API 许可名单

为了获得可下载且更易于搜索的列表，可以在原生 FGAC 中允许的 Python 函数中找到包含模块和类的文件。

工作线程间加密

在 Amazon EMR 6.15.0 及更高版本中，请在 Spark 作业运行时中的工作线程之间启用双向 TLS 加密通信。启用后，EMR Serverless 会自动为作业运行下预置的每个工作线程生成并分发唯一的证书。当这些工作线程通信交换控制消息或传输随机排序数据时，会建立双向 TLS 连接，并使用配置的证书来

验证彼此的身份。如果工作线程无法验证其他证书，TLS 握手会失败，EMR Serverless 将中止彼此的连接。

如果将 Lake Formation 与 EMR Serverless 结合使用，默认情况下会启用双向 TLS 加密。

在 EMR Serverless 上启用双向 TLS 加密

要在 Spark 应用程序上启用双向 TLS 加密，请在创建 EMR Serverless 应用程序时将 `spark.ssl.internode.enabled` 设置为 true。如果您使用 AWS 控制台创建 EMR Serverless 应用程序，请选择使用自定义设置，然后展开应用程序配置，然后输入您的。`runtimeConfiguration`

```
aws emr-serverless create-application \
--release-label emr-6.15.0 \
--runtime-configuration '{
  "classification": "spark-defaults",
  "properties": {"spark.ssl.internode.enabled": "true"}
}' \
--type "SPARK"
```

如果要为单个 Spark 作业运行启用双向 TLS 加密，请在使用 `spark-submit` 时将 `spark.ssl.internode.enabled` 设置为 true。

```
--conf spark.ssl.internode.enabled=true
```

使用 KMS CMK 进行磁盘加密

默认情况下，EMR Serverless 使用服务拥有的加密密钥对连接到工作人员的所有磁盘进行加密。您可以选择使用自己的 AWS KMS 客户托管密钥对这些磁盘进行加密（CMKs）。这使您可以更好地控制加密密钥，包括建立和维护密钥策略以及审计密钥使用情况。

您可以在创建应用程序或提交单个作业时配置磁盘加密。在应用程序级别启用后，该应用程序上的所有作业都将继承加密设置。您还可以通过在提交作业时指定磁盘加密配置来覆盖应用程序的默认设置。

> **Note**
>
> EMR 无服务器磁盘加密仅支持对称 KMS 密钥。不支持非对称 KMS 密钥。您必须使用中创建的对称加密 KMS 密钥。 AWS KMS有关的更多信息 AWS KMS，请参阅什么是 AWS KMS？

使用加密上下文

或者，EMR Serverless 使用加密上下文为加密操作提供额外的经过身份验证的数据。加密上下文是一组键值对，可以包含非机密的其他经过身份验证的数据。加密上下文以加密方式绑定到加密数据，因此解密数据需要相同的加密上下文。

在 EMR Serverless 中，您可以在配置磁盘加密时指定自定义加密上下文。此加密上下文包含在 AWS CloudTrail 日志中，可帮助您识别和了解您的 KMS 操作。

> **Note**
>
> 请勿将敏感信息存储在加密环境中，因为这些信息以纯文本形式出现在日志中。 AWS CloudTrail

使用客户托管密钥配置磁盘加密

CreateApplication

要使用您自己的 KMS 密钥加密磁盘，请在创建 EMR Serverless 应用程序时添加`diskEncryptionConfiguration`参数。

```
aws emr-serverless create-application \
  --type TYPE \
  --name APPLICATION_ID \
  --release-label RELEASE_LABEL \
  --region AWS_REGION \
  --disk-encryption-configuration '{
        "encryptionKeyArn": "key-arn",
        "encryptionContext": {
            "key": "value"
        }
  }'
```

UpdateApplication

要更新 KMS 密钥 ARN and/or 加密上下文，请在更新应用程序时使用新值指定`diskEncryptionConfiguration`参数。

```
aws emr-serverless update-application \
  --name APPLICATION_ID \
```

```
  --region AWS_REGION \
  --disk-encryption-configuration '{
        "encryptionKeyArn": "key-arn",
        "encryptionContext": {
            "key": "value"
        }
  }'
```

> **Note**
>
> 要取消设置应用程序上已配置的磁盘加密，请在更新应用程序`diskEncryptionConfiguration`时传递一个空值。

StartJobRun

要使用您自己的 KMS 密钥加密磁盘，请在提交作业运行时使用`diskEncryptionConfiguration`配置。

```
--configuration-overrides '{
        "diskEncryptionConfiguration": {
            "encryptionKeyArn": "key-arn",
            "encryptionContext": {
                "key": "value"
            }
        }
    }'
```

Public Livy 端点

要在通过公共 Livy 端点创建 Spark 会话时使用自己的 KMS 密钥加密磁盘，请在会话的`conf`对象中指定加密配置。

```
data = {
    "kind": "pyspark",
    "heartbeatTimeoutInSecond": 60,
    "conf": {
        "emr-serverless.session.executionRoleArn": "role_arn",
        "spark.emr-serverless.disk.encryptionKeyArn": "key-arn",
        "spark.emr-serverless.disk.encryptionContext": "key1:value1,key2:value2"  #
 Optional
```

```
    }
}

# Send request to create a session with the Livy API endpoint
request = AWSRequest(method='POST', url=endpoint + "/sessions", data=json.dumps(data),
 headers=headers)
```

磁盘加密所需的权限

EMR 无服务器的加密密钥权限

使用自己的加密密钥加密磁盘时，必须为`emr-serverless.amazonaws.com`主体配置以下 KMS 密钥权限：

- `kms:GenerateDataKey`: 生成用于加密磁盘卷的数据密钥
- `kms:Decrypt`: 在访问加密磁盘内容时解密数据密钥

```
{
    "Effect": "Allow",
    "Principal":{
        "Service": "emr-serverless.amazonaws.com"
    },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
    ],
    "Resource": "*",
    "Condition": {
        "StringLike": {
            "aws:SourceArn": "arn:aws:emr-serverless:region:aws-account-id:/
applications/application-id"
        },
        "StringEquals": {
            "kms:EncryptionContext:applicationId": "application-id",
            "aws:SourceAccount": "aws-account-id"
        }
    }
}
```

作为安全最佳实践，建议在 KMS 密钥政策中添加 `aws:SourceArn` 条件键。IAM 全局条件键 `aws:SourceArn` 可确保 EMR Serverless 仅将 KMS 密钥用于应用程序 ARN。此外，包

含aws:SourceAccount条件密钥可以将您的 KMS 密钥的使用限制为来自条件中指定的 AWS 账户 ID 的请求，从而提供另一层安全保护。

作业运行时角色必须在其 IAM 策略中具有以下权限：

```
{
    "Sid": "Enable GDK and Decrypt",
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "kms:GenerateDataKey",
            "kms:Decrypt"
        ],
        "Resource": "key-arn"
    }
}
```

所需的用户权限

提交任务的用户必须具有使用密钥的权限。您可以在 KMS 密钥政策或 IAM 策略中为用户、组或角色指定权限。如果提交作业的用户没有 KMS 密钥权限，EMR Serverless 会拒绝提交作业运行。

示例密钥政策

以下密钥策略提供对kms:DescribeKey、kms:GenerateDataKey和的权限kms:Decrypt：

- kms:DescribeKey：在使用客户管理的 KMS 密钥之前验证其是否已启用和 SYMMETRIC。

```
{
    "Sid": "Enable DescribeKey",
    "Effect": "Allow",
    "Principal":{
        "AWS": "arn:aws:iam::111122223333:user/user-name"
    },
    "Action": [
        "kms:DescribeKey"
    ],
    "Resource": "*"
},
{
    "Sid": "Enable GDK and Decrypt",
```

```
    "Effect": "Allow",
    "Principal":{
        "AWS": "arn:aws:iam::111122223333:user/user-name"
    },
    "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "kms:ViaService": "emr-serverless.region.amazonaws.com",
            "kms:EncryptionContext:key": "value"
        }
    }
}
```

作为安全最佳实践，建议在 KMS 密钥政策中添加 `kms:viaService` 条件键。它将 KMS 密钥的使用限制为仅限来自 emr-serverless 的验证请求。

示例 IAM 策略

以下 IAM 策略提供对`kms:DescribeKey`、`kms:GenerateDataKey`和的权限`kms:Decrypt`。

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "kms:DescribeKey",
            "kms:GenerateDataKey",
            "kms:Decrypt"
        ],
        "Resource": "key-arn"
    }
}
```

监控密钥使用情况

您可以通过以下方式监控客户托管密钥在 EMR Serverless 中的使用情况。 AWS CloudTrail AWS CloudTrail 将所有 API 调用捕获 AWS KMS 为事件，包括来自 EMR 无服务器控制台、EMR Serverless API、CLI 或 SDK 的调用。 AWS AWS

捕获的信息包括您指定的加密上下文，这可以帮助您识别和审计使用您的 KMS 密钥的特定 EMR Serverless 资源。例如，您可能会在中看到与以下内容类似的事件 AWS CloudTrail。有关使用的更多信息 AWS CloudTrail，请参阅《AWS CloudTrail 用户指南》。

GenerateDataKey

EMR Serverless 创建加密磁盘卷时的 GenerateDataKey 操作示例事件


```
{
    "eventVersion": "1.11",
    "userIdentity": {
        "type": "AWSService",
        "principalId": "user",
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2025-07-28T21:43:51Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "ipAddress",
    "userAgent": "userAgent",
    "requestParameters": {
        "encryptionContext": {
            "applicationId": "test"
        },
        "keyId": "arn:aws:kms:region:accountId:key/ffffffff-fffff-aaaaa-eeee-sample",
        "keySpec": "AES_256"
    },
    "responseElements": null,
    "additionalEventData": {
        "keyMaterialId":
 "145c963debe558dfb01848d2a4539da940f3478852f86cfe2f52d5df796a5a02"
    },
    "requestID": "cc9d1c5e-97c4-4a4f-ae7a-e576sample",
    "eventID": "0b0fef09-f28d-4da8-a5a1-17b74sample",
    "readOnly": true,
    "resources": [
        {
            "accountId": "account",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:region:accountId:key/ffffffff-fffff-aaaaa-eeee-sample"
        }
    ],
```

```
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "accountId",
    "eventCategory": "Management"
}
```

Decrypt

EMR Serverless 访问加密数据时解密操作的示例事件。

```
{
    "eventVersion": "1.11",
    "userIdentity": {
        "type": "AWSService",
        "principalId": "user",
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2025-07-28T21:43:51Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "ipAddress",
    "userAgent": "userAgent",
    "requestParameters": {
        "encryptionContext": {
            "applicationId": "test"
        },
        "keyId": "arn:aws:kms:region:accountId:key/ffffffff-fffff-aaaaa-eeee-sample",
        "keySpec": "AES_256"
    },
    "responseElements": null,
    "additionalEventData": {
        "keyMaterialId":
 "145c963debe558dfb01848d2a4539da940f3478852f86cfe2f52d5df796a5a02"
    },
    "requestID": "cc9d1c5e-97c4-4a4f-ae7a-e576sample",
    "eventID": "0b0fef09-f28d-4da8-a5a1-17b74sample",
    "readOnly": true,
    "resources": [
        {
            "accountId": "account",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:region:accountId:key/ffffffff-fffff-aaaaa-eeee-sample"
```

```
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "accountId",
    "eventCategory": "Management"
}
```

了解更多

以下资源提供有关静态数据加密的更多信息。

- 有关 AWS KMS 基本概念的更多信息，请参阅《AWS KMS 开发人员指南》。
- 有关安全最佳实践的更多信息 AWS KMS，请参阅《AWS KMS 开发人员指南》。

在 EMR Serverless 中使用 Secrets Manager 保护数据

AWS Secrets Manager 是一项秘密存储服务，用于保护数据库凭证、API 密钥和其他机密信息。然后，在您的代码中，将硬编码的凭证替换为对 Secrets Manager 的 API 调用。这有助于确保别人在检查您的代码时不会泄露密钥，因为代码中没有密钥。有关概述，请参阅 AWS Secrets Manager 用户指南。

Secrets Manager 使用密 AWS Key Management Service 钥对机密进行加密。有关更多信息，请参阅《AWS Secrets Manager 用户指南》中的密钥加密和解密。

此外，您还可以配置 Secrets Manager 以根据指定的计划自动轮换密钥。这使您能够将长期密钥替换为短期密钥，这有助于显著减少泄露风险。有关更多信息，请参阅《AWS Secrets Manager 用户指南》中的轮换 AWS Secrets Manager 密钥。

Amazon EMR Serverless 与集成， AWS Secrets Manager 因此您可以将数据存储在 Secrets Manager 中，并在配置中使用密钥 ID。

EMR Serverless 如何使用密钥

当您将数据存储在 Secrets Manager 中并在您的 EMR Serverless 配置中使用密钥 ID 时，您不会以纯文本形式将敏感配置数据传递给 EMR Serverless，也不会将其暴露给外部。 APIs如果您指示键值对包含您存储在 Secrets Manager 中的密钥 ID，EMR Serverless 会在将配置数据发送给运行作业的工作线程时检索该密钥。

要指示配置的键值对包含对 Secrets Manager 中存储的密钥的引用，请在配置值中添加 `EMR.secret@` 注释。对于带有密钥 ID 注释的任何配置属性，EMR Serverless 会调用 Secrets Manager 并在作业执行时解析密钥。

如何创建密钥

要创建密钥，请按照《AWS Secrets Manager 用户指南》中创建 AWS Secrets Manager 密钥中的步骤进行操作。在步骤 3 中，选择明文字段，输入敏感值。

在配置分类中提供密钥

以下示例展示了如何在 `StartJobRun` 的配置分类中提供密钥。如果要在应用程序级别配置 Secrets Manager 的分类，请参阅EMR Serverless 的默认应用程序配置。

在示例中，将 *SecretName* 替换为要检索的密钥名称。有关更多信息，请参阅如何创建密钥。

本节内容

指定密钥引用 Spark

Example在 Spark 的外部 Hive 元存储配置中指定密钥引用

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

```
            --conf spark.driver.cores=2
            --conf spark.executor.memory=10G
            --conf spark.driver.memory=6G
            --conf spark.executor.cores=4"
        }
    }' \
    --configuration-overrides '{
        "monitoringConfiguration": {
        "s3MonitoringConfiguration": {
            "logUri": "s3://amzn-s3-demo-bucket/spark/logs/"
        }
    }
}'
```

Example在 spark-defaults 分类中指定外部 Hive 元存储配置的密钥引用

```
{
        "classification": "spark-defaults",
        "properties": {

 "spark.hadoop.javax.jdo.option.ConnectionDriverName":"org.mariadb.jdbc.Driver"
            "spark.hadoop.javax.jdo.option.ConnectionURL":"jdbc:mysql://db-host:db-
port/db-name"
            "spark.hadoop.javax.jdo.option.ConnectionUserName":"connection-user-name"
            "spark.hadoop.javax.jdo.option.ConnectionPassword":
 "EMR.secret@SecretName",
        }
    }
```

指定密钥引用 Hive

Example在 Hive 的外部 Hive 元存储配置中指定密钥引用

```
aws emr-serverless start-job-run \
  --application-id "application-id" \
  --execution-role-arn "job-role-arn" \
    --job-driver '{
        "hive": {
        "query": "s3://amzn-s3-demo-bucket/emr-serverless-hive/query/hive-query.ql",
        "parameters": "--hiveconf hive.exec.scratchdir=s3://amzn-s3-demo-bucket/emr-
serverless-hive/hive/scratch
                    --hiveconf hive.metastore.warehouse.dir=s3://amzn-s3-demo-bucket/
emr-serverless-hive/hive/warehouse
```

```
                    --hiveconf javax.jdo.option.ConnectionUserName=username
                    --hiveconf
 javax.jdo.option.ConnectionPassword=EMR.secret@SecretName
                    --hiveconf
 hive.metastore.client.factory.class=org.apache.hadoop.hive.ql.metadata.SessionHiveMetaStoreCli
                    --hiveconf
 javax.jdo.option.ConnectionDriverName=org.mariadb.jdbc.Driver
                    --hiveconf javax.jdo.option.ConnectionURL=jdbc:mysql://db-host:db-
port/db-name"
        }
    }' \
    --configuration-overrides '{
        "monitoringConfiguration": {
        "s3MonitoringConfiguration": {
            "logUri": "s3://amzn-s3-demo-bucket"
        }
    }
}'
```

Example在 hive-site 分类中指定外部 Hive 元存储配置的密钥引用

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

授予 Amazon EMR 检索密钥的访问权限

要允许 EMR Serverless 从 Secrets Manager 检索密钥值，请在创建密钥时将以下策略语句添加到密钥中。您必须使用客户托管的 KMS 密钥创建自己的密钥，EMR Serverless 才能读取密钥值。有关更多信息，请参阅《AWS Secrets Manager 用户指南》中的 KMS 密钥的权限。

在以下策略中，将 *applicationId* 替换为应用程序 ID。

密钥的资源策略

您必须在 AWS Secrets Manager 中密钥的资源策略中包含以下权限，以允许 EMR Serverless 检索密钥值。为确保只有特定应用程序才能检索此密钥，您可以选择在策略中指定 EMR Serverless 应用程序 ID 作为条件。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue",
        "secretsmanager:DescribeSecret"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:SourceArn": "arn:aws:emr-serverless:*:123456789012:/applications/
*"
        }
      },
      "Sid": "AllowSECRETSMANAGERGetsecretvalue"
    }
  ]
}
```

使用以下客户托管 AWS Key Management Service (AWS KMS) 密钥策略创建您的密钥：

客户管理的密钥 AWS KMS 政策

```
{
    "Sid": "Allow EMR Serverless to use the key for decrypting secrets",
    "Effect": "Allow",
    "Principal": {
        "Service": [
            "emr-serverless.amazonaws.com"
        ]
```

```
    },
    "Action": [
        "kms:Decrypt",
        "kms:DescribeKey"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "kms:ViaService": "secretsmanager.AWS ##.amazonaws.com"
        }
    }
}
```

轮换密钥

轮换是指定期更新密钥。您可以将 AWS Secrets Manager 配置为按照您指定的计划自动轮换密钥。这样，您可以将长期密钥替换为短期密钥。这有助于降低泄露的风险。当作业变为运行状态时，EMR Serverless 会从注释的配置中检索密钥值。如果您或某个进程更新了 Secrets Manager 中的密钥值，则必须提交新作业，这样作业才能获取更新的值。

> **Note**
>
> 已处于运行状态的作业无法获取更新的密钥值。这可能会导致作业失败。

将 Amazon S3 访问权限管控与 EMR Serverless 结合使用

EMR Serverless 的 S3 Access Grants 概述

在 Amazon EMR 6.15.0 及更高版本中，Amazon S3 访问权限管控提供了一种可扩展的访问控制解决方案，可用于增强对 EMR Serverless 中 Amazon S3 数据的访问。如果您的 S3 数据有复杂或大规模的权限配置，请使用访问权限管控来扩展用户、角色和应用程序的 S3 数据权限。

使用 S3 Access Grants 可增强对 Amazon S3 数据的访问，以超出运行时系统角色或 IAM 角色授予的权限，这些权限附加在可访问 EMR Serverless 应用程序的身份上。

有关更多信息，请参阅《Amazon EMR 管理指南》中的 Managing access with S3 Access Grants for Amazon EMR 和《Amazon Simple Storage Service 用户指南》中的使用 S3 Access Grants 管理访问权限。

本节介绍了如何启动 EMR Serverless 应用程序，以使用 S3 Access Grants 提供对 Amazon S3 中数据的访问权限。有关将 S3 Access Grants 与其他 Amazon EMR 部署配合使用的步骤，请参阅以下文档：

- 将 S3 Access Grants 与 Amazon EMR 结合使用
- 将 S3 Access Grants 与 EKS 上的 Amazon EMR 结合使用

利用 S3 Access Grants 启动 EMR Serverless 应用程序，以进行数据管理。

您可以在 EMR Serverless 上启用 S3 Access Grants，并启动 Spark 应用程序。当您的应用程序请求获取 S3 数据时，Amazon S3 会提供限定于特定存储桶、前缀或对象的临时凭证。

1. 为 EMR Serverless 应用程序设置作业执行角色。包括运行 Spark 作业和使用 S3 访问权限管控、`s3:GetDataAccess` 和 `s3:GetAccessGrantsInstanceForPrefix` 所需的 IAM 权限：

```
{
    "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**
>
> 如果您为作业执行指定的 IAM 角色具有直接访问 S3 的额外权限，那么即使用户没有 S3 访问权限管控授予的权限，也能访问该角色允许的数据。

2. 启动 EMR Serverless 应用程序时，请使用 6.15.0 或更高版本的 Amazon EMR 发行版标签和 `spark-defaults` 分类，如下面的示例所示。将 *`red text`* 中的值替换为适合您的使用场景的适当值。

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
```

```
  --job-driver '{
        "sparkSubmit": {
            "entryPoint": "s3://us-east-1.elasticmapreduce/emr-containers/samples/
wordcount/scripts/wordcount.py",
            "entryPointArguments": ["s3://amzn-s3-demo-destination-bucket1/
wordcount_output"],
            "sparkSubmitParameters": "--conf spark.executor.cores=1 --conf
 spark.executor.memory=4g --conf spark.driver.cores=1 --conf spark.driver.memory=4g
 --conf spark.executor.instances=1"
        }
    }' \
  --configuration-overrides '{
    "applicationConfiguration": [{
        "classification": "spark-defaults",
        "properties": {
          "spark.hadoop.fs.s3.s3AccessGrants.enabled": "true",
          "spark.hadoop.fs.s3.s3AccessGrants.fallbackToIAM": "false"
         }
      }]
}'
```

将 S3 Access Grants 与 EMR Serverless 结合使用时的注意事项

有关将 Amazon S3 访问权限管控与 EMR Serverless 结合使用时的重要支持、兼容性和行为信息，请参阅《Amazon EMR 管理指南》中的 Amazon EMR 的 S3 Access Grants 注意事项。

使用记录亚马逊 EMR 无服务器 API 调用 AWS CloudTrail

Amazon EMR Serverless 与一项服务集成，可记录用户 AWS CloudTrail、角色或服务 AWS 在 EMR Serverless 中执行的操作。 CloudTrail 将 EMR Serverless 的所有 API 调用捕获为事件。捕获的调用包括来自 EMR Serverless 控制台的调用以及针对 EMR Serverless API 操作的代码调用。如果您创建跟踪，请允许将 CloudTrail事件持续传输到 Amazon S3 存储桶，包括 EMR Serverless 的事件。如果您未配置跟踪，您仍然可以在事件历史记录中访问 CloudTrail 控制台中的最新事件。使用收集的信息 CloudTrail，您可以确定向 EMR Serverless 发出的请求、发出请求的 IP 地址、谁发出了请求、何时发出请求以及其他详细信息。

要了解更多信息，请参阅《AWS CloudTrail 用户指南》。

EMR 中的无服务器信息 CloudTrail

CloudTrail 在您创建账户 AWS 账户 时已在您的账户上启用。在 EMR Serverless 中发生活动时，该活动会与其他 AWS 服务 CloudTrail 事件一起记录在事件历史记录中。您可以在 AWS 账户账户中访问、搜索和下载最新事件。有关更多信息，请参阅使用事件历史记录查看 CloudTrail 事件。

要持续记录您的事件 AWS 账户，包括 EMR Serverless 的事件，请创建跟踪。跟踪允许 CloudTrail 将日志文件传输到 Amazon S3 存储桶。默认情况下，在控制台中创建跟踪记录时，此跟踪记录应用于所有 AWS 区域。跟踪记录 AWS 分区中所有区域的事件，并将日志文件传送到您指定的 Amazon S3 存储桶。此外，配置其他 AWS 服务以进一步分析和处理 CloudTrail 日志中收集的事件数据。有关更多信息，请参阅以下内容：

- 创建跟踪记录概述
- CloudTrail 支持的服务和集成
- 配置 Amazon SNS 通知 CloudTrail
- 接收来自多个区域的 CloudTrail 日志文件和接收来自多个账户的 CloudTrail 日志文件

所有 EMR 无服务器操作均由 EMR 无服务器 API 参考记录 CloudTrail 并记录在《EM R 无服务器 API 参考》中。例如，调用`StartJobRun`和`CancelJobRun`操作会在 CloudTrail 日志文件中生成条目。`CreateApplication`

每个事件或日志条目都包含有关生成请求的人员信息。身份信息有助于您确定以下内容：

- 请求是使用根证书还是 AWS Identity and Access Management (IAM) 用户凭证发出。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其他 AWS 服务发出。

有关更多信息，请参阅CloudTrail 用户身份元素。

了解 EMR Serverless 日志文件条目

跟踪是一种配置，允许将事件作为日志文件传输到您指定的 Amazon S3 存储桶。 CloudTrail 日志文件包含一个或多个日志条目。事件代表来自任何来源的单个请求，包括有关请求的操作、操作的日期和时间、请求参数等的信息。 CloudTrail 日志文件不是公共 API 调用的有序堆栈跟踪，因此它们不会按任何特定的顺序出现。

以下示例显示了演示该`CreateApplication`操作的 CloudTrail 日志条目。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE:admin",
        "arn": "arn:aws:sts::012345678910:assumed-role/Admin/admin",
        "accountId": "012345678910",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AIDACKCEVSQ6C2EXAMPLE",
                "arn": "arn:aws:iam::012345678910:role/Admin",
                "accountId": "012345678910",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-06-01T23:46:52Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-06-01T23:49:28Z",
    "eventSource": "emr-serverless.amazonaws.com",
    "eventName": "CreateApplication",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "203.0.113.0",
    "userAgent": "PostmanRuntime/7.26.10",
    "requestParameters": {
        "name": "my-serverless-application",
        "releaseLabel": "emr-6.6",
        "type": "SPARK",
        "clientToken": "0a1b234c-de56-7890-1234-567890123456"
    },
    "responseElements": {
        "name": "my-serverless-application",
        "applicationId": "1234567890abcdef0",
        "arn": "arn:aws:emr-serverless:us-west-2:555555555555:/
applications/1234567890abcdef0"
    },
    "requestID": "890b8639-e51f-11e7-b038-EXAMPLE",
    "eventID": "874f89fa-70fc-4798-bc00-EXAMPLE",
```

```
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "012345678910",
    "eventCategory": "Management"
}
```

Amazon EMR Serverless 的合规性验证

EMR Serverless 的安全性和合规性由第三方审计机构作为多个 AWS 合规计划的一部分进行评估，包括：

- 系统和组织控制（SOC）
- 支付卡行业数据安全标准（PCI DSS）
- 联邦风险与授权管理计划（FedRAMP）中级
- 《健康保险流通与责任法案》（HIPAA）

AWS 在 “按合规计划划分的范围内的 AWS 服务” 中提供了经常更新的特定合规计划范围内的AWS 服务列表。

您可以使用第三方审计报告进行下载 AWS Artifact。有关更多信息，请参阅在 Artifac t 中下载 AWS 报告。

有关 AWS 合规计划的更多信息，请参阅合AWS 规计划。

使用 EMR Serverless 时，您的合规责任取决于数据的敏感性、组织的合规目标以及适用的法律法规。如果您在使用 EMR Serverless 时需要遵守 HIPAA、PCI 或 FedRAMP Moderate 等标准， AWS 将提供以下实用资源：

- 《安全与合规性快速入门指南》讨论了架构注意事项以及部署以安全性和合规性为重点的基准环境的步骤。 AWS
- AWS 《客户合规指南》可以帮助您从合规角度了解责任共担模式。这些指南总结了保护 AWS 服务的最佳实践，并将指南映射到跨多个框架的安全控制，包括美国国家标准与技术研究院（NIST）、支付卡行业安全标准委员会（PCI）和国际标准化组织（ISO）。
- AWS Config 可用于评估您的资源配置对内部实践、行业指南和法规的遵循情况。
- AWS 合规资源是一系列可能适用于您所在行业和所在地区的工作簿和指南。

- AWS Sec@@ urity Hub 为您提供内部安全状态的全面视图， AWS 并帮助您检查自己是否符合安全行业标准和最佳实践。
- AWS Audit Manager— 这 AWS 服务 可以帮助您持续审计 AWS 使用情况，从而简化风险管理以及对法规和行业标准的合规性。

Amazon EMR Serverless 的弹性

AWS 全球基础设施是围绕 AWS 区域和可用区构建的。 AWS 区域提供多个物理隔离和隔离的可用区，这些可用区通过低延迟、高吞吐量和高度冗余的网络相连。利用可用区，可设计和操作在可用区之间无中断地自动实现失效转移的应用程序和数据库。与传统的单个或多个数据中心基础设施相比，可用区具有更高的可用性、容错能力和可扩展性。

有关 AWS 区域和可用区的更多信息，请参阅AWS 全球基础设施。

除了 AWS 全球基础设施外，Amazon EMR Serverless 还提供通过 EMRFS 与 Amazon S3 的集成，以帮助支持您的数据弹性和备份需求。

Amazon EMR Serverless 中的基础设施安全性

作为一项托管服务，Amazon EMR 受到 AWS 全球网络安全的保护。有关 AWS 安全服务以及如何 AWS 保护基础设施的信息，请参阅AWS 云安全。要使用基础设施安全的最佳实践来设计您的 AWS 环境，请参阅 S AWS ecurity Pillar Well-Architected Fram ework 中的基础设施保护。

您可以使用 AWS 已发布的 API 调用通过网络访问 Amazon EMR。客户端必须支持以下内容：

- 传输层安全性协议（TLS）。我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 具有完全向前保密（PFS）的密码套件，例如 DHE（临时 Diffie-Hellman）或 ECDHE（临时椭圆曲线 Diffie-Hellman）。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

Amazon EMR Serverless 中的配置和漏洞分析

AWS 处理基本的安全任务，例如客户机操作系统 (OS) 和数据库修补、防火墙配置和灾难恢复。这些流程已通过相应第三方审核和认证。有关更多详细信息，请参阅以下资源：

- Amazon EMR Serverless 的合规性验证
- 责任共担模式
- Amazon Web Services：安全过程概述

EMR Serverless 端点和配额

服务端点

要以编程方式连接到 AWS 服务，请使用终端节点。端点是 AWS Web 服务入口点的 URL。除标准 AWS 终端节点外，有些终端节点还在选定区域 AWS 服务 提供 FIPS 终端节点。下表列出了 EMR Serverless 的服务端点。有关更多信息，请参阅 AWS 服务 端点。

EMR Serverless 服务端点

区域名称	区域	端点	协议
美国东部（俄亥俄州）	us-east-2 （仅限于以下可用区：use2-az1、use2-az2 和 use2-az3）	emr-serverless.us-east-2.amazonaws.com	HTTPS
美国东部（弗吉尼亚州北部）	us-east-1 （仅限于以下可用区：use1-az1、use1-az2、use1-az4、use1-az5 和 use1-az6）	emr-serverless.us-east-1.amazonaws.com emr-serverless-fips.us-east-1.amazonaws.com	HTTPS
美国西部（北加利福尼亚）	us-west-1	emr-serverless.us-west-1.amazonaws.com	HTTPS
美国西部（俄勒冈州）	us-west-2	emr-serverless.us-west-2.amazonaws.com	HTTPS

区域名称	区域	端点	协议
		emr-serverless-fips.us-west-2.amazonaws.com	
非洲（开普敦）	af-south-1	emr-serverless.af-south-1.amazonaws.com	HTTPS
亚太地区（香港）	ap-east-1	emr-serverless.ap-east-1.amazonaws.com	HTTPS
亚太地区（雅加达）	ap-southeast-3	emr-serverless.ap-southeast-3.amazonaws.com	HTTPS
亚太地区（墨尔本）	ap-southeast-4	emr-serverless.ap-southeast-4.amazonaws.com	HTTPS
亚太地区（马来西亚）	ap-southeast-5	emr-serverless.ap-southeast-5.amazonaws.com	HTTPS

区域名称	区域	端点	协议
亚太地区（孟买）	ap-south-1	emr-serverless.ap-south-1.amazonaws.com	HTTPS
亚太地区（大阪）	ap-northeast-3	emr-serverless.ap-northeast-3.amazonaws.com	HTTPS
亚太地区（首尔）	ap-northeast-2	emr-serverless.ap-northeast-2.amazonaws.com	HTTPS
亚太地区（新加坡）	ap-southeast-1	emr-serverless.ap-southeast-1.amazonaws.com	HTTPS
亚太地区（悉尼）	ap-southeast-2	emr-serverless.ap-southeast-2.amazonaws.com	HTTPS
亚太地区（东京）	ap-northeast-1	emr-serverless.ap-northeast-1.amazonaws.com	HTTPS

区域名称	区域	端点	协议
加拿大（中部）	ca-central-1 （仅限于以下可用区：cac1-az1 和 cac1-az2）	emr-serverless.ca-central-1.amazonaws.com	HTTPS
加拿大西部（卡尔加里）	ca-west-1	emr-serverless.ca-west-1.amazonaws.com	HTTPS
欧洲地区（法兰克福）	eu-central-1	emr-serverless.eu-central-1.amazonaws.com	HTTPS
欧洲（苏黎世）	eu-central-2	emr-serverless.eu-central-2.amazonaws.com	HTTPS
欧洲地区（爱尔兰）	eu-west-1	emr-serverless.eu-west-1.amazonaws.com	HTTPS
欧洲地区（伦敦）	eu-west-2	emr-serverless.eu-west-2.amazonaws.com	HTTPS
欧洲地区（米兰）	eu-south-1	emr-serverless.eu-south-1.amazonaws.com	HTTPS

区域名称	区域	端点	协议
欧洲地区（巴黎）	eu-west-3	emr-serverless.eu-west-3.amazonaws.com	HTTPS
欧洲（西班牙）	eu-south-2	emr-serverless.eu-south-2.amazonaws.com	HTTPS
欧洲地区（斯德哥尔摩）	eu-north-1	emr-serverless.eu-north-1.amazonaws.com	HTTPS
以色列（特拉维夫）	il-central-1	emr-serverless.il-central-1.amazonaws.com	HTTPS
中东（巴林）	me-south-1	emr-serverless.me-south-1.amazonaws.com	HTTPS
中东（阿联酋）：	me-central-1	emr-serverless.me-central-1.amazonaws.com	HTTPS
南美洲（圣保罗）	sa-east-1	emr-serverless.sa-east-1.amazonaws.com	HTTPS

区域名称	区域	端点	协议
中国（北京）	cn-north-1 （仅限于以下可用区：cnn1-az1、cnn1-az2）	emr-serverless.cn-north-1.amazonaws.com.cn	HTTPS
AWS GovCloud （美国东部）	us-gov-east-1	emr-serverless.us-gov-east-1.amazonaws.com	HTTPS
AWS GovCloud （美国西部）	us-gov-west-1	emr-serverless.us-gov-west-1.amazonaws.com	HTTPS

服务配额

服务配额，也称为限制，是 AWS 账户 您可以使用的服务资源或操作的最大数量。EMR Serverless 每分钟收集一次服务配额使用指标，并将其发布在 AWS/Usage 命名空间中。

> **Note**
>
> 新 AWS 账户的初始配额较低，随着时间的推移可能会增加。Amazon EMR Serverless 会监控每个账户中的账户使用情况 AWS 区域，然后根据您的使用情况自动增加配额。

下表列出了 EMR Serverless 的服务配额。有关更多信息，请参阅 AWS 服务 配额。

Name	默认限制	是否可调整？	说明
CPUs 每个账户的最大并发 v	16	是	当前可以为CPUs 该账户同时运行的最大 v 数。 AWS 区域

API 限制

下面列出了 AWS 账户每个区域的 API 限制。

资源	默认配额
ListApplications	每秒 10 个事务。每秒突发 50 个事务。
CreateApplication	每秒 1 个事务。每秒突发 25 个事务。
DeleteApplication	每秒 1 个事务。每秒突发 25 个事务。
GetApplication	每秒 10 个事务。每秒突发 50 个事务。
UpdateApplication	每秒 1 个事务。每秒突发 25 个事务。
ListJobRuns	每秒 1 个事务。每秒突发 25 个事务。
StartJobRun	每秒 1 个事务。每秒突发 25 个事务。
GetDashboardForJobRun	每秒 1 个事务。每秒突发 2 个事务。
CancelJobRun	每秒 1 个事务。每秒突发 25 个事务。
GetJobRun	每秒 10 个事务。每秒突发 50 个事务。
StartApplication	每秒 1 个事务。每秒突发 25 个事务。
StopApplication	每秒 1 个事务。每秒突发 25 个事务。

其他考虑因素

以下列表包含 EMR Serverless 的其他注意事项。

- EMR Serverless 有以下版本： AWS 区域
  - 美国东部（俄亥俄州）
  - 美国东部（弗吉尼亚州北部）
  - 美国西部（北加利福尼亚）
  - 美国西部（俄勒冈州）
  - 非洲（开普敦）
  - 亚太地区（香港）
  - 亚太地区（雅加达）
  - 亚太地区（孟买）
  - 亚太地区（大阪）
  - 亚太地区（首尔）
  - 亚太地区（新加坡）
  - 亚太地区（悉尼）
  - 亚太地区（东京）
  - 加拿大（中部）
  - 欧洲地区（法兰克福）
  - 欧洲地区（爱尔兰）
  - 欧洲地区（伦敦）
  - 欧洲地区（米兰）
  - 欧洲地区（巴黎）
  - 欧洲（西班牙）
  - 欧洲地区（斯德哥尔摩）
  - 中东（巴林）
  - 中东（阿联酋）：
  - 南美洲（圣保罗）
  - AWS GovCloud（美国东部）
  - AWS GovCloud（美国西部）

有关与这些区域关联的端点列表，请参阅服务端点。

- 作业运行的默认超时为 12 小时。您可以使用 `startJobRun` API 或 AWS SDK 中的`executionTimeoutMinutes`属性更改此设置。如果希望作业运行永不超时，可将 `executionTimeoutMinutes` 设置为 0。例如，如果您有一个流处理应用程序，请将 `executionTimeoutMinutes` 设置为 0，以允许流处理作业持续运行。
- `getJobRun`API 中的`billedResourceUtilization`属性显示了为作业运行计费的聚合 vCPU、内存和存储空间。 AWS 计费资源包括工作线程的 1 分钟最低使用量，以及每个工作线程超过 20GB 的额外存储空间。这些资源不包括空闲预初始化工作线程的使用情况。
- 如果没有 VPC 连接，则任务可以访问相同的 VPC 中的某些 AWS 服务 终端节点 AWS 区域。这些服务包括 Amazon S3、 AWS Glu AWS e、Lake Formation、Amazon CloudWatch Logs AWS KMS AWS Security Token Service Amazon DynamoDB、、、和 AWS Secrets Manager。您可以启用 VPC 连接以通过 AWS PrivateLink 访问其他 AWS 服务 ，但这不是必须的。要访问外部服务，请使用 VPC 创建应用程序。
- EMR Serverless 不支持 HDFS。工作线程上的本地磁盘是 EMR Serverless 在作业运行期间用于随机排序和处理数据的临时存储。

Amazon EMR Serverless 发行版

Amazon EMR 发行版是一组来自大数据生态系统的开源应用程序。每个发行版都包含大数据应用程序、组件和功能，您可以从中选择，在运行作业时部署和配置 Amazon EMR Serverless。

在 Amazon EMR 6.6.0 及更高版本中，部署 EMR Serverless。此部署选项不适用于早期 Amazon EMR 发行版。提交作业时，请指定以下受支持的发行版之一。

主题

- EMR Serverless6.7.0
- EMR Serverless6.6.0

AWS runtime for Apache Spark（emr-spark-8.0 预览版）

下表列出了可用的应用程序版本AWS runtime for Apache Spark（emr-spark-8.0-preview）。

应用程序版本信息

应用程序	版本
Spark	4.0.1-amzn-0

AWS runtime for Apache Spark(emr-spark-8.0 预览版) 发行说明

- 预览版 — 这是以 Apache Spark 4.0. AWS runtime for Apache Spark 1 为特色的预览版。此预览版仅在 EMR Serverless 上可用。
- 地区供货情况-此预览版适用于所有提供 EMR Serverless 的 AWS 区域，但中国和 AWS GovCloud（美国）地区除外。
- 应用程序版本信息-此版本附带以下应用程序版本：
  - AWS 适用于 Java 的 SDK 2.35.5, 1.12.792
  - Python 3.9，3.11, 3.12
  - Scala 2.13.16
  - AmazonCloudWatchAgent 1.300034.0-amzn-0
  - 三角洲 4.0.0-amzn-0-spark
  - 冰山 1.10.0-amzn-spark-0
  - 对于支持 C orretto 17（JDK 17）的应用程序，此版本默认附带亚马逊 Corretto 17（基于 OpenJDK 构建）。
- 预览限制-此预览版中不提供以下功能：
  - 交互和集成功能：不支持 SageMaker Unified Studio、EMR Studio 集成、Spark Connect、JupyterEnterpriseGateway Livy 等。
  - 表格格式和访问控制：不支持 Hudi、Delta 通用格式和带有行级或列级筛选和运算符的细粒度访问控制 (FGAC)。 DDL/DML
  - 数据连接器: spark-sql-kinesis、emr-dynamodb 和 spark-redshift 连接器不可用。

  - 历史服务器：Pers ition Spark 历史服务器在此预览版中不可用。用户仍然可以访问实时 Spark 用户界面来实时监控和调试活动的无服务器作业。
  - 特殊功能：实体化视图不可用。
- 预览功能-您可以在此预览版中测试以下功能。不建议将此预览版用于生产工作负载：
  - SQL 功能：具有更严格的类型处理的 ANSI SQL 模式、用于链接操作的 SQL PIPE 语法 (|>)、半结构化 JSON 数据的 VARIANT 数据类型、带有控制流语句和会话变量的 SQL 脚本以及 SQL 用户定义的函数。
  - 流媒体增强功能：带有 transformWithState 运算符的任意状态处理 API v2、用于可查询流媒体状态的状态数据源读取器（实验性），以及通过改进 RocksDB 变更日志检查点功能增强的状态存储。
  - 表格格式支持：Apache Iceberg v3 支持变体数据类型、 AWS S3 表格集成以及与 Iceberg、Delta Lake 和 Hive 表 AWS Lake Formation 的完整表访问权限 (FTA)。
- 其他文档——有关其他 Apache Spark 文档，请参阅 A pache Spark 4.0.1 版本文档。

开始使用

要开始使用 Apache Spark 4.0.1 预览版，请使用 CLI 创建一个 EMR 无服务器应用程序： AWS

```
aws emr-serverless create-application --type spark \
  --release-label emr-spark-8.0-preview \
  --region us-east-1 --name spark4-preview
```

EMR Serverless7.12.0

下表列出了 EMR Serverless 7.12.0 中可用的应用程序版本。

应用程序	版本
Apache Spark	3.5.6
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless 7.12.0 版本说明

- 新特征
  - 适用于 EMR 无服务器的无服务器存储 — Amazon EMR 无服务器引入了无服务器存储，EMR 版本为 7.12 及更高版本，无需为 Apache Spark 工作负载配置本地磁盘。EMR Serverless 可自动处理中间数据操作，例如随机播放，无需支付存储费用。无服务器存储将存储与计算分离，允许 Spark 在空闲时立即释放工作人员，而不是让他们保持活动状态以保留临时数据。要了解更多信息，请参阅无服务器存储。
  - 冰山物化视图 ——从亚马逊 EMR 7.12.0 开始，亚马逊 EMR Spark 支持创建和管理 Iceberg 物化视图 (MV)
  - Hudi Full Table Acces s——从亚马逊 EMR 7.12.0 开始，亚马逊 EMR 现在支持根据你在 Lake Formation 中定义的策略，在 Apache Spark 中对 Apache Hudi 进行全表访问 (FTA) 控制。当作业角色具有完整表访问权限时，此功能允许对您的 Amazon EMR Spark 作业进行 Lake Formation 注册表的读写操作。
  - 冰山版本升级 ——亚马逊 EMR 7.12.0 支持 Apache Iceberg 版本 1.10

EMR Serverless7.11.0

下表列出了 EMR Serverless 7.11.0 中可用的应用程序版本。

应用程序	版本
Apache Spark	3.5.6
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless 7.11.0 版本说明

- 最长任务执行时间-从本版本起，BATCH 作业的运行`executionTimeoutMinutes`中`StartJobRun`最大值为 7 天。`executionTimeoutMinutes`不能再将批处理作业运行设置为`0`即无超时。

EMR Serverless7.10.0

下表列出了 EMR Serverless 7.10.0 中可用的应用程序版本。

应用程序	版本
Apache Spark	3.5.5
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless 7.10.0 版本说明

- EMR Serverless 的指标：监控指标经过重组，将重点放在 `ApplicationName` 和 `JobName` 维度上。今后将不再更新较旧的指标。有关更多信息，请参阅监控 EMR Serverless 应用程序和作业。

EMR Serverless7.9.0

下表列出了 EMR Serverless 7.9.0 中可用的应用程序版本。

应用程序	版本
Apache Spark	3.5.5
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless7.8.0

下表列出了 EMR Serverless 7.8.0 中可用的应用程序版本。

应用程序	版本
Apache Spark	3.5.4

应用程序	版本
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless7.7.0

下表列出了 EMR Serverless 7.7.0 中可用的应用程序版本。

应用程序	版本
Apache Spark	3.5.3
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless7.6.0

下表列出了 EMR Serverless 7.6.0 中可用的应用程序版本。

应用程序	版本
Apache Spark	3.5.3
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless7.5.0

下表列出了 EMR Serverless 7.5.0 中可用的应用程序版本。

应用程序	版本
Apache Spark	3.5.2
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless7.4.0

下表列出了 EMR Serverless 7.4.0 中可用的应用程序版本。

应用程序	版本
Apache Spark	3.5.2
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless7.3.0

下表列出了 EMR Serverless 7.3.0 中可用的应用程序版本。

应用程序	版本
Apache Spark	3.5.1
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless 7.3.0 发行说明

- 使用 EMR Serverless 实现作业并发和排队：在 Amazon EMR 7.3.0 或更高版本上，当创建新的 EMR Serverless 应用程序时，默认启用作业并发和队列。有关更多信息，请参阅the section called “作业并发和排队”，其中详细介绍了如何开始使用并发和排队，还包含功能注意事项列表。

EMR Serverless7.2.0

下表列出了 EMR Serverless 7.2.0 中可用的应用程序版本。

应用程序	版本
Apache Spark	3.5.1
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless 7.2.0 发布说明

- 带有 EMR Serverless 的 Lak e Formation — 你现在可以使用 AWS Lake Formation 对由 S3 支持的数据目录表应用精细的访问控制。此功能允许您为 EMR Serverless Spark 作业中的读取查询配置表、行、列和单元格级访问控制。有关更多信息，请参阅 the section called “适用于 FGAC 的 Lake Formation” 和 the section called “注意事项和限制”。

EMR Serverless7.1.0

下表列出了 EMR Serverless 7.1.0 中可用的应用程序版本。

应用程序	版本
Apache Spark	3.5.0
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless7.0.0

下表列出了 EMR Serverless 7.0.0 中可用的应用程序版本。

应用程序	版本
Apache Spark	3.5.0
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless6.15.0

下表列出了 EMR Serverless 6.15.0 中可用的应用程序版本。

应用程序	版本
Apache Spark	3.4.1
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless 6.15.0 发布说明

- TLS 支持：在 Amazon EMR Serverless 6.15.0 及更高版本中，请在 Spark 作业运行时在工作线程之间启用双向 TLS 加密通信。启用后，EMR Serverless 会自动为每个工作线程生成一个唯一证书，并在作业运行时提供给工作线程，工作线程可在 TLS 握手过程中使用该证书来相互验证，并建立一个加密通道来安全处理数据。有关双向 TLS 加密的更多信息，请参阅工作线程间加密。

EMR Serverless6.14.0

下表列出了 EMR Serverless 6.14.0 中可用的应用程序版本。

应用程序	版本
Apache Spark	3.4.1
Apache Hive	3.1.3

应用程序	版本
Apache Tez	0.10.2

EMR Serverless6.13.0

下表列出了 EMR Serverless 6.13.0 中可用的应用程序版本。

应用程序	版本
Apache Spark	3.4.1
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless6.12.0

下表列出了 EMR Serverless 6.12.0 中可用的应用程序版本。

应用程序	版本
Apache Spark	3.4.0
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless6.11.0

下表列出了 EMR Serverless 6.11.0 中可用的应用程序版本。

应用程序	版本
Apache Spark	3.3.2

应用程序	版本
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless 6.11.0 发布说明

- 访问其他账户中的 S3 资源：在 6.11.0 及更高版本中，您可以配置多个 IAM 角色，以便在从 EMR Serverless 访问不同 AWS 账户中的 Amazon S3 存储桶时代入。

EMR Serverless6.10.0

下表列出了 EMR Serverless 6.10.0 中可用的应用程序版本。

应用程序	版本
Apache Spark	3.3.1
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless 6.10.0 发布说明

- 在 6.10.0 或更高版本的 EMR Serverless 应用程序中，`spark.dynamicAllocation.maxExecutors` 属性的默认值为 `infinity`。早期版本默认为 `100`。有关更多信息，请参阅Spark 作业属性。

EMR Serverless6.9.0

下表列出了 EMR Serverless 6.9.0 中可用的应用程序版本。

应用程序	版本
Apache Spark	3.3.0

应用程序	版本
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless 6.9.0 发布说明

- Amazon EMR 发行版 6.9.0 及更高版本包含适用于 Apache Spark 的 Amazon Redshift 集成。本地集成之前是一种开源工具，现在是 Spark 连接器，您可以将其用于构建 Apache Spark 应用程序，这些应用程序可在 Amazon Redshift 和 Amazon Redshift Serverless 中读取和写入数据。有关更多信息，请参阅 在 Amazon EMR Serverless 上使用适用于 Apache Spark 的 Amazon Redshift 集成。
- EMR Serverless 版本 6.9.0 增加了对 AWS Graviton2 (arm64) 架构的支持。您可以使用`create-application`和的`architecture`参数`update-application` APIs 来选择 arm64 架构。有关更多信息，请参阅Amazon EMR Serverless 架构选项。
- 现在，您可以直接从 EMR Serverless Spark 和 Hive 应用程序导出、导入、查询和连接 Amazon DynamoDB 表。有关更多信息，请参阅使用 Amazon EMR Serverless 连接到 DynamoDB。

已知问题

- 如果您使用适用于 Apache Spark 的 Amazon Redshift 集成，并且具有 Parquet 格式的时间、timetz、时间戳或 timestamptz（精度为微秒），连接器会将时间值舍入为最接近的毫秒值。解决方法是使用文本卸载格式 `unload_s3_format` 参数。

EMR Serverless6.8.0

下表列出了 EMR Serverless 6.8.0 中可用的应用程序版本。

应用程序	版本
Apache Spark	3.3.0
Apache Hive	3.1.3
Apache Tez	0.9.2

EMR Serverless6.7.0

下表列出了 EMR Serverless 6.7.0 中可用的应用程序版本。

应用程序	版本
Apache Spark	3.2.1
Apache Hive	3.1.3
Apache Tez	0.9.2

特定于引擎的更改、增强和解决的问题

下表列出了特定于引擎的新功能。

更改	描述
功能	Tez 调度程序现在支持抢占 Tez 任务，而不是抢占容器

EMR Serverless6.6.0

下表列出了 EMR Serverless 6.6.0 中可用的应用程序版本。

应用程序	版本
Apache Spark	3.2.0
Apache Hive	3.1.2
Apache Tez	0.9.2

EMR Serverless 初始发行说明

- EMR Serverless 支持 Spark 配置分类 `spark-defaults`。此分类会更改 Spark 的 `spark-defaults.conf` XML 文件中的值。配置分类允许您自定义应用程序。有关更多信息，请参阅配置应用程序。
- EMR Serverless 支持 Hive 配置分类 `hive-site`、`tez-site`、`emrfs-site` 和 `core-site`。此分类将分别更改 Hive 的 `hive-site.xml` 文件、Tez 的 `tez-site.xml` 文件、Amazon EMR 的 EMRFS 设置或 Hadoop 的 `core-site.xml` 文件中的值。配置分类允许您自定义应用程序。有关更多信息，请参阅配置应用程序。

特定于引擎的更改、增强和解决的问题

- 下表列出了 Hive 和 Tez 逆向移植。

  Hive 和 Tez 更改

更改	描述
逆向移植	TEZ-4430：修复了 `tez.task.launch.cmd-opts` 属性的问题
逆向移植	HIVE-25971：修复了由于打开缓存线程池而导致 Tez 任务关闭延迟的问题

文档历史记录

下表描述了自 EMR Serverless 上次发布以来文档的重要更改。如需对此文档更新的通知，您可以订阅 RSS 源。

变更	说明	日期
新的预览版	AWS runtime for Apache Spark（emr-spark-8.0 预览版）	2025 年 11 月 21 日
新版本	EMR Serverless7.2.0	2024 年 7 月 25 日
新版本	EMR Serverless7.1.0	2024 年 4 月 17 日
更新为现有策略。	在 Amazon EMRServerless ServiceRolePolicy 政策中EC2PolicyStatement 添加了新的SidCloudWatchPolicyStatement 和。	2024 年 1 月 25 日
新版本	EMR Serverless7.0.0	2023 年 12 月 29 日
新版本	EMR Serverless6.15.0	2023 年 11 月 17 日
新特征	配置多个 IAM 角色，以便在从 EMR Serverless（6.11 及更高版本）访问不同账户中的 Amazon S3 存储桶时代入	2023 年 10 月 18 日
新版本	EMR Serverless6.14.0	2023 年 10 月 17 日
新特征	EMR Serverless 的默认应用程序配置	2023 年 9 月 25 日
更新为默认 Hive 属性	更新了 hive.driver.disk 、hive.tez.disk.size 、hive.tez.auto.reducer.paral	2023 年 9 月 12 日

	lelism 和 tez.group ing.min-size Hive 作业属性的默认值。	
新版本	EMR Serverless6.13.0	2023 年 9 月 11 日
新版本	EMR Serverless6.12.0	2023 年 7 月 21 日
新版本	EMR Serverless6.11.0	2023 年 6 月 8 日
服务相关角色策略更新	更新了 AmazonEMR ServerlessServiceR olePolicy SLR 角色，以在 "AWS/Usage" 命名空间中发布账户级使用情况。	2023 年 4 月 20 日
EMR Serverless 正式发布（GA）	这是 EMR Serverless 的首次公开发布。	2022 年 6 月 1 日

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。