亚马逊云科技

Amazon EMR on EKS 开发指南

Amazon EMR

Amazon EMR: Amazon EMR on EKS 开发指南

Amazon Web Services 文档中描述的 Amazon Web Services 服务或功能可能因区域而异。要查看适用于中国区域的差异，请参阅 中国的 Amazon Web Services 服务入门 (PDF)。

Table of Contents

什么是 Amazon EMR on EKS？ 1
Amazon EMR on EKS 架构 2
了解 Amazon EMR on EKS 概念和术语 3
Kubernetes 命名空间 3
虚拟集群 3
任务运行 4
Amazon EMR 容器 4
将工作提交到 Amazon EMR on EKS 虚拟集群时会发生什么 4
开始使用 Amazon EMR on EKS 6
运行 Spark 应用程序 7
最佳实践 12
安全性 12
Pyspark 作业提交 12
存储 12
元数据仓集成 12
调试 13
排查 Amazon EMR on EKS 中的问题 13
节点放置 13
性能 13
成本优化 13
使用 Amazon Outposts 13
自定义 Docker 镜像 14
如何自定义 Docker 镜像 14
先决条件 15
步骤 1：从 Amazon Elastic Container Registry（Amazon ECR）中检索基础镜像 15
步骤 2：自定义基础镜像 16
步骤 3：（可选但建议）验证自定义镜像 17
步骤 4：发布自定义镜像 18
步骤 5：使用自定义镜像在 Amazon EMR 中提交 Spark 工作负载 19
为交互式端点自定义 Docker 映像 21
使用多架构镜像 23
如何选择基础映像 URI 的详细信息 24
Amazon ECR 注册账户 25
自定义映像的注意事项 27

运行 Flink 任务 28
Flink Kubernetes Operator 28
设置 29
安装部署 Flink Kubernetes Operator 30
运行 Flink 应用程序 31
运行 Flink 应用程序的安全角色权限 35
卸载 Operator 37
Flink Native Kubernetes 38
设置 38
入门 39
安全要求 41
为 Flink 和 FluentD 自定义 Docker 映像 42
先决条件 42
从 Amazon Elastic Container Registry 中检索基础映像 43
自定义基础映像 43
发布自定义映像 44
提交 Flink 工作负载 45
监控 45
使用 Amazon Managed Service for Prometheus 46
使用 Flink UI 47
使用监视配置 48
Flink 如何支持高可用性和作业弹性 53
使用高可用性 53
优化重启时间 59
正常停用 64
使用 Autoscaler 67
Autoscaler 参数自动调整 68
Amazon EMR on EKS 上 Flink 作业的维护和故障排除 77
维护 Flink 应用程序 77
故障排除 78
支持的发行版 81
运行 Spark 任务 83
StartJobRun 83
设置 84
使用 StartJobRun 提交任务运行 107
使用任务提交者分类 109

使用亚马逊 EMR 容器默认分类 114
Spark Operator 117
设置 117
入门 118
垂直弹性伸缩 122
卸载 126
使用监控配置监控 Spark 127
安全性 133
spark-submit 143
设置 143
入门 144
安全性 145
Apache Livy 150
设置 151
入门 151
运行 Spark 应用程序 156
卸载 158
安全性 158
安装属性 168
排查常见的环境变量格式错误 172
管理任务运行 173
使用 CLI 进行管理 173
运行 Spark SQL 脚本 178
任务运行状态 181
查看控制台中的任务 181
任务运行常见错误 182
使用任务模板 187
创建并使用作业模板启动作业运行 187
定义任务模板参数 189
控制对任务模板的访问权限 191
使用 Pod 模板 192
常见场景 192
利用 Amazon EMR on EKS 启用 Pod 模板 194
Pod 模板字段 196
边车容器注意事项 199
使用重试策略 201

设置重试策略 201
检索策略状态 203
监控任务 204
查找驱动程序日志 204
使用 Spark 事件日志轮替 205
使用 Spark 容器日志轮换 206
使用垂直自动扩展功能 207
设置 208
入门 210
配置 212
监控建议 218
卸载 219
运行交互式工作负载 220
交互式端点概述 220
交互式端点先决条件 222
Amazon CLI 222
eksctl 222
Amazon EKS 集群 222
授予集群访问权限 223
激活服务账户的 IAM 角色 223
创建 IAM 任务执行角色 223
授予用户访问权限 223
通过 Amazon EMR 注册 Amazon EKS 集群 224
负载均衡器控制器 224
创建交互式端点 224
创建交互式端点 224
指定自定义参数 225
..... 226
交互式端点参数 227
配置交互式端点的设置 228
监控 Spark 作业 228
自定义容器组（Pod）模板 229
将 JEG Pod 部署到节点组 230
JEG 配置选项 234
修改 PySpark参数 234
自定义内核映像 235

- 监控交互式端点 ... 236
  - 示例 ... 238
- 使用自托管式 Jupyter notebook ... 239
  - 创建安全组 ... 239
  - 创建交互式端点 ... 239
  - 获取网关服务器 URL ... 240
  - 获取身份验证令牌 ... 240
  - 部署 Notebook ... 241
  - 清理 ... 246
- 使用 CLI 命令获取有关交互式端点的信息 ... 247
  - ... 247
  - 列出交互式端点 ... 248
  - 删除交互式端点 ... 250

上传数据 ... 251
- 先决条件 ... 251
- 入门 ... 251

监控任务 ... 253
- 通过 Amazon Events CloudWatch 监控作业 ... 253
- 通过事件在 EKS 上自动执行 Amazon EMR CloudWatch ... 254
- 示例：设置调用 Lambda 的规则 ... 255
- 使用 Amazon CloudWatch Events 使用重试策略监控作业的驱动程序窗格 ... 255

管理虚拟集群 ... 256
- 创建虚拟集群 ... 256
- 列出虚拟集群 ... 257
- 描述虚拟集群 ... 258
- 删除虚拟集群 ... 258
- 虚拟集群状态 ... 258

教程 ... 259
- 使用 Delta Lake ... 259
- 使用 Iceberg ... 260
  - 用于目录集成的 Spark 会话配置 ... 261
- 使用 PyFlink ... 262
- 在 Flink Amazon 中使用 Glue ... 263
- 使用 Apache Hudi ... 266
  - 提交 Apache Hudi 作业 ... 266
- 使用 Spark RAPIDS ... 269

在 Redshift 上使用 Spark ... 273
启动 Spark 应用程序 ... 274
对 Amazon Redshift 进行身份验证 ... 274
对 Amazon Redshift 进行读取和写入 ... 277
注意事项 ... 279
使用 Volcano ... 279
概览 ... 280
安装 ... 280
提交：Spark Operator ... 281
提交：`spark-submit` ... 283
使用 YuniKorn ... 284
概览 ... 284
创建集群 ... 284
安装 YuniKorn ... 286
提交：Spark Operator ... 287
提交：`spark-submit` ... 289
安全性 ... 12
最佳实践 ... 293
采用最低特权原则 ... 293
终端节点的访问控制列表 ... 293
获取自定义镜像的最新安全更新 ... 293
限制 Pod 凭证访问 ... 293
隔离不受信任的应用程序代码 ... 294
基于角色的访问控制（RBAC）权限 ... 294
限制对节点组 IAM 角色或实例配置文件凭证的访问 ... 294
数据保护 ... 295
静态加密 ... 295
传输中加密 ... 297
身份和访问管理 ... 298
受众 ... 298
使用身份进行身份验证 ... 299
使用策略管理访问 ... 301
Amazon EMR on EKS 如何与 IAM 配合使用 ... 303
使用服务相关角色 ... 309
Amazon EMR on EKS 的托管式策略 ... 312
将任务执行角色与 Amazon EMR on EKS 结合使用 ... 313

基于身份的策略示例 315
基于标签的访问控制策略 318
故障排除 320
在 EKS 上使用 Amazon EMR 和 Lake Formati Amazon on 322
EKS 上的亚马逊 EMR 如何与 Lake Formation 配合使用 Amazon 322
在 EKS 上使用亚马逊 EMR 启用 Lake Formation 324
注意事项和限制 331
故障排除 334
日志记录和监控 335
加密日志 335
CloudTrail 日志 338
S3 访问权限管控 341
概览 341
启动集群 341
注意事项 343
合规性验证 343
恢复能力 343
基础设施安全性 343
配置和漏洞分析 344
接口 VPC 端点 344
为 Amazon EMR on EKS 创建 VPC 终端节点策略 345
跨账户访问 347
先决条件 348
如何访问跨账户 Amazon S3 存储桶或 DynamoDB 表 348
为资源添加标签 352
标签基本知识 352
标记资源 352
标签限制 353
使用标签 Amazon CLI 和在 EKS API 上使用亚马逊 EMR 354
故障排除 13
PVC 任务失败 356
验证 356
修补 356
手动修补 360
垂直自动扩展失败 362
403 禁止错误 363

未找到命名空间 ... 363
Docker 凭证错误 ... 363
Spark 运算符故障 ... 364
Helm 图表安装失败 ... 364
不支持的文件系统异常 ... 364
服务端点和限额 ... 366
服务端点 ... 366
服务配额 ... 368
查看配额并申请增加配额 ... 370
发行版 ... 371
7.9.0 版本 ... 372
发行版 ... 372
发行说明 ... 374
更改 ... 375
emr-7.9.0-最新 ... 375
emr-7.9.0-20250425 ... 376
emr-7.9.0-flink-latest ... 376
emr-7.9.0-flink-20250425 ... 376
7.8.0 版本 ... 376
发行版 ... 376
发行说明 ... 378
更改 ... 379
emr-7.8.0-最新 ... 380
emr-7.8.0-20250228 ... 380
emr-7.8.0-flink-latest ... 380
emr-7.8.0-flink-20250228 ... 380
7.7.0 版本 ... 381
发行版 ... 381
发行说明 ... 382
更改 ... 384
emr-7.7.0-最新 ... 384
emr-7.0-20250131 ... 384
emr-7.0-flink-latest ... 384
emr-7.0-flink-20250131 ... 385
7.6.0 版本 ... 385
发行版 ... 385

发行说明 387
特征 388
更改 388
emr-7.6.0-最新 388
emr-7.6.0-20241213 389
emr-7.6.0-flink-latest 389
emr-7.6.0-flink-20241213 389
7.5.0 版本 389
发行版 389
发行说明 389
7.4.0 版本 390
发行版 390
发行说明 390
7.3.0 发行版 390
发行版 391
发行说明 392
特征 394
更改 394
emr-7.3.0-latest 394
emr-7.3.0-29240920 394
emr-7.3.0-flink-latest 395
emr-7.3.0-flink-29240920 395
7.2.0 发行版 395
发行版 395
发行说明 397
特征 398
emr-7.2.0-latest 399
emr-7.2.0-20240610 399
emr-7.2.0-flink-latest 399
emr-7.2.0-flink-20240610 399
7.1.0 发行版 400
发行版 400
发行说明 401
特征 403
emr-7.1.0-latest 403
emr-7.1.0-20240321 403

- emr-7.1.0-flink-latest 404
- emr-7.1.0-flink-20240321 404
- 7.0.0 发行版 404
  - 发行版 404
  - 发行说明 406
  - 特征 407
  - 更改 407
  - emr-7.0.0-latest 408
  - emr-7.0.0-2024321 408
  - emr-7.0.0-20231211 408
  - emr-7.0.0-flink-latest 408
  - emr-7.0.0-flink-2024321 409
  - emr-7.0.0-flink-20231211 409
- 6.15.0 发行版 409
  - 发行版 409
  - 发行说明 411
  - 特征 412
  - emr-6.15.0-latest 412
  - emr-6.15.0-20240105 413
  - emr-6.15.0-20231109 413
  - emr-6.15.0-flink-latest 413
  - emr-6.15.0-flink-20240105 413
  - emr-6.15.0-flink-20231109 414
- 6.14.0 发行版 414
  - 发行版 414
  - 发行说明 415
  - 特征 417
  - emr-6.14.0-latest 417
  - emr-6.14.0-20231005 417
- 6.13.0 版本 417
  - 发行版 417
  - 发行说明 419
  - 特征 420
  - emr-6.13.0-latest 420
  - emr-6.13.0-20230814 421
- 6.12.0 版本 421

发行版 421
发行说明 422
特征 423
emr-6.12.0-latest 423
emr-6.12.0-20240321 424
emr-6.12.0-20230701 424
6.11.0 版本 424
发行版 424
发行说明 425
特征 426
emr-6.11.0-latest 426
emr-6.11.0-20230905 427
emr-6.11.0-20230509 427
6.10.0 版本 427
emr-6.10.0-latest 430
emr-6.10.0-20230905 430
emr-6.10.0-20230624 430
emr-6.10.0-20230421 430
emr-6.10.0-20230403 430
emr-6.10.0-20230220 431
6.9.0 版本 431
emr-6.9.0-latest 433
emr-6.9.0-20230905 434
emr-6.9.0-20230624 434
emr-6.9.0-20221108 434
6.8.0 版本 434
emr-6.8.0-latest 437
emr-6.8.0-20230905 438
emr-6.8.0-20230624 438
emr-6.8.0-20221219 438
emr-6.8.0-20220802 438
6.7.0 版本 439
emr-6.7.0-latest 440
emr-6.7.0-20240321 440
emr-6.7.0-20230624 441
emr-6.7.0-20221219 441

emr-6.7.0-20220630 441
6.6.0 版本 441
emr-6.6.0-最新版本 443
emr-6.6.0-20240321 443
emr-6.6.0-20230624 443
emr-6.6.0-20221219 443
emr-6.6.0-20220411 444
6.5.0 版本 444
emr-6.5.0-最新版本 445
emr-6.5.0-20240321 445
emr-6.5.0-20221219 446
emr-6.5.0-20220802 446
emr-6.5.0-20211119 446
6.4.0 版本 446
emr-6.4.0-latest 447
emr-6.4.0-20240321 448
emr-6.4.0-20221219 448
emr-6.4.0-20210830 448
6.3.0 版本 448
emr-6.3.0-latest 449
emr-6.3.0-20240321 450
emr-6.3.0-20220802 450
emr-6.3.0-20211008 450
emr-6.3.0-20210802 450
emr-6.3.0-20210429 451
6.2.0 版本 451
emr-6.2.0-latest 452
emr-6.2.0-20240321 452
emr-6.2.0-20220802 453
emr-6.2.0-20211008 453
emr-6.2.0-20210802 453
emr-6.2.0-20210615 453
emr-6.2.0-20210129 454
emr-6.2.0-20201218 454
emr-6.2.0-20201201 454
5.36.0 版本 454

emr-5.36.0-latest ... 455
emr-5.36.0-20240321 ... 456
emr-5.36.0-20221219 ... 456
emr-5.36.0-20220620 ... 456
emr-5.36.0-20220525 ... 456
5.35.0 版本 ... 457
emr-5.35.0-latest ... 458
emr-5.35.0-20240321 ... 458
emr-5.35.0-20221219 ... 458
emr-5.35.0-20220802 ... 458
emr-5.35.0-20220307 ... 459
5.34 版本 ... 459
emr-5.34.0-最新版本 ... 460
emr-5.34.0-20240321 ... 460
emr-5.34.0-20220802 ... 460
emr-5.34.0-20211208 ... 461
5.33.0 版本 ... 461
emr-5.33.0-latest ... 462
emr-5.33.0-20240321 ... 462
emr-5.33.0-20221219 ... 463
emr-5.33.0-20220802 ... 463
emr-5.33.0-20211008 ... 463
emr-5.33.0-20210802 ... 463
emr-5.33.0-20210615 ... 464
emr-5.33.0-20210323 ... 464
5.32.0 版本 ... 464
emr-5.32.0-latest ... 465
emr-5.32.0-20240321 ... 466
emr-5.32.0-20220802 ... 466
emr-5.32.0-20211008 ... 466
emr-5.32.0-20210802 ... 466
emr-5.32.0-20210615 ... 467
emr-5.32.0-20210129 ... 467
emr-5.32.0-20201218 ... 467
emr-5.32.0-20201201 ... 467
文档历史记录 ... 468

.... cdlxx

什么是 Amazon EMR on EKS？

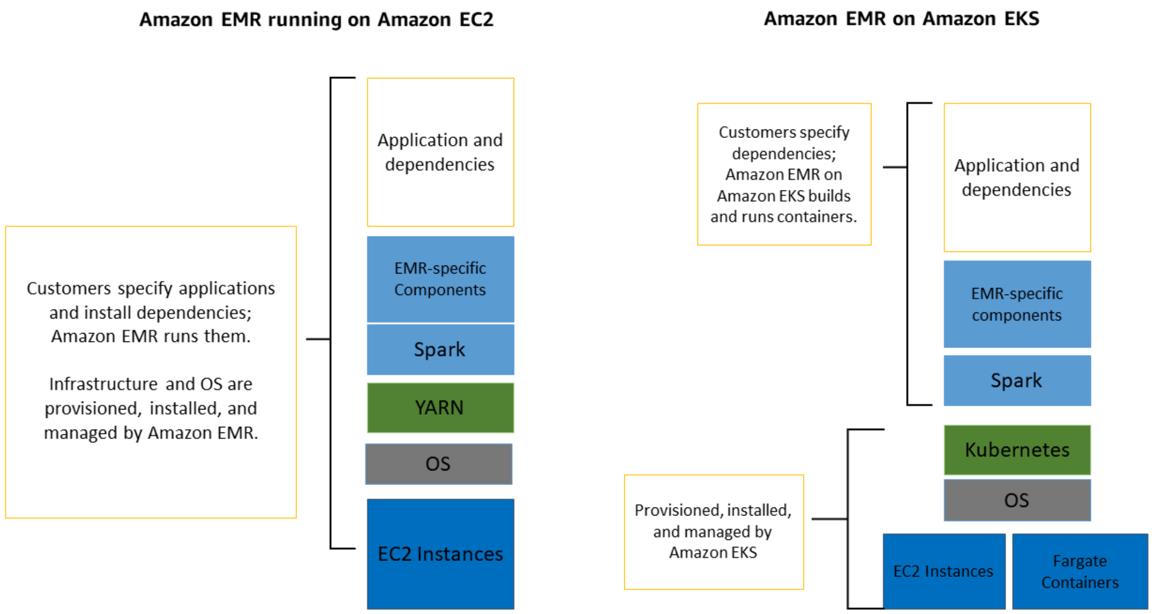
Amazon EMR on EKS 为 Amazon EMR 提供了部署选项，可让您在 Amazon Elastic Kubernetes Service（Amazon EKS）上运行开源大数据框架。借助此部署选项，您可以专注于运行分析工作负载，同时 Amazon EMR on EKS 为开源应用程序构建、配置和管理容器。

如果您已经使用 Amazon EMR，您现在可以在同一 Amazon EKS 集群上使用其它类型的应用程序来运行基于 Amazon EMR 的应用程序。此部署选项还提高了资源利用率，并简化了跨多个可用区的基础设施管理。如果您已在 Amazon EKS 上运行大数据框架，您现在可以使用 Amazon EMR 自动调配和管理，并更快地运行 Apache Spark。

Amazon EMR on EKS 可让您的团队更加高效地协作，并以更轻松、经济实惠的方式来处理海量数据：

- 您可以在公共资源池上运行应用程序，而无需预置基础设施。您可以使用 Amazon EMR Studio 和软件开发 Amazon 工具包，也可以 Amazon CLI 开发、提交和诊断在 EKS 集群上运行的分析应用程序。您可以使用自行托管式 Apache Airflow 或 Amazon Managed Workflows for Apache Airflow（MWAA），在 Amazon EMR on EKS 上运行计划任务。
- 基础设施团队可以集中管理通用计算平台，将 Amazon EMR 工作负载与其它基于容器的应用程序整合起来。您可以使用常用 Amazon EKS 工具简化基础设施管理，并利用共享集群来处理需要不同版本开源框架的工作负载。您还可以通过自动化 Kubernetes 集群管理和操作系统修补来减少运营开销。借助 Amazon EC2 和 Amazon Fargate，您可以启用多个计算资源来满足性能、运营或财务要求。

下图展示了 Amazon EMR 的两种不同的部署模型。



主题

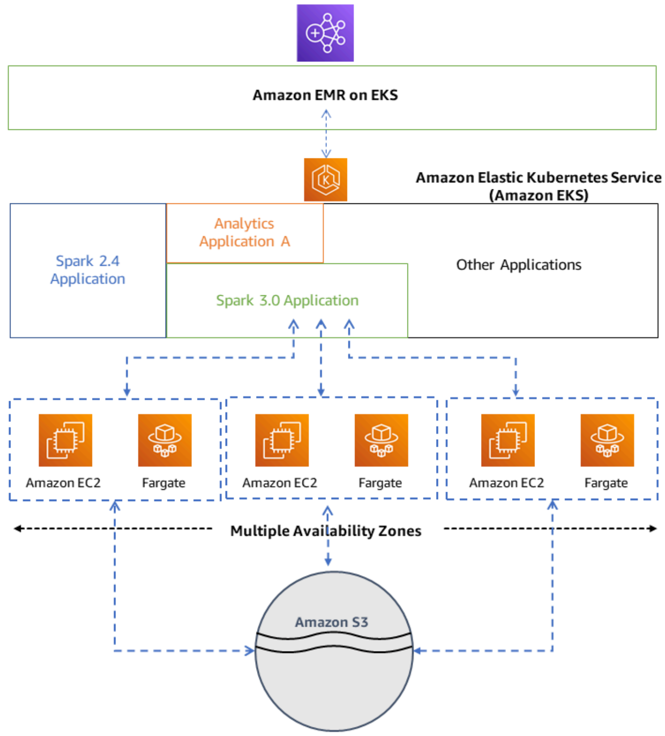
- [Amazon EMR on EKS 架构](#)
- [了解 Amazon EMR on EKS 概念和术语](#)
- [将工作提交到 Amazon EMR on EKS 虚拟集群时会发生什么](#)

Amazon EMR on EKS 架构

Amazon EMR on EKS 会松散地将应用程序与它们运行的基础设施结合起来。每个基础设施层都为后续层提供编排。向 Amazon EMR 提交任务时，您的任务定义将包含其所有特定于应用程序的参数。Amazon EMR 使用这些参数指示 Amazon EKS 要部署哪些 Pod 和容器。然后，Amazon EKS 将来自亚马逊的计算资源上线 EC2 ，这些资源是运行任务所 Amazon Fargate 必需的。

通过这种松散的服务耦合，您可以同时运行多个安全隔离任务。您还可以使用不同的计算后端对同一任务进行基准测试，或将您的任务分布到多个可用区，以提高可用性。

下图说明了 EKS 上的 Amazon EMR 如何与其他 Amazon 服务配合使用。



了解 Amazon EMR on EKS 概念和术语

Amazon EMR on EKS 为 Amazon EMR 提供了部署选项，可让您在 Amazon Elastic Kubernetes Service（Amazon EKS）上运行开源大数据框架。本主题提供了一些常见术语的背景信息，包括命名空间、虚拟集群和作业运行，这些是提交进行处理的工作单元。

Kubernetes 命名空间

Amazon EKS 使用 Kubernetes 命名空间在多位用户和多个应用程序之间划分集群资源。这些命名空间是多租户环境的基础。Kubernetes 命名空间可以使用 Amazon EC2 或 Amazon Fargate 作为计算提供商。这种灵活度为您提供了不同的性能和成本选项，供您运行任务。

虚拟集群

虚拟集群是 Amazon EMR 注册的 Kubernetes 命名空间。Amazon EMR 使用虚拟集群运行任务和主机终端节点。同一个物理集群可以支持多个虚拟集群。但是，每个虚拟集群都映射到 EKS 集群上的命名空间。虚拟集群不会创建任何可增加您账单的活动资源，以及需要在服务之外进行生命周期管理的活动资源。

任务运行

作业运行是你在 EKS 上提交给 Amazon EMR 的一个工作单元，例如 Spark jar、 PySpark 脚本或 sparkSQL 查询。一个任务可以有多个任务运行。提交任务运行时，需要包括以下信息：

- 任务运行时所在的虚拟集群。
- 用来标识任务的任务名称。
- 执行角色 – 一种范围限定 IAM 角色，可运行任务并允许您指定任务可以访问哪些资源。
- Amazon EMR 发行版标注，可指定要使用的开源应用程序版本。
- 提交任务时要使用的构件，例如 spark-submit 参数。

默认情况下，日志将上载到 Spark 历史记录服务器，并可通过 Amazon Web Services Management Console访问。您还可以将事件日志、执行日志和指标推送到 Amazon S3 和 Amazon CloudWatch。

Amazon EMR 容器

Amazon EMR 容器是在 Amazon EMR on EKS 的 API 名称。以下情况使用 `emr-containers` 前缀：

- 用于 Amazon EMR on EKS 的 CLI 命令中的前缀。例如，`aws emr-containers start-job-run`。
- 用于 Amazon EMR on EKS 的 IAM policy 操作之前的前缀。例如，`"Action": [ "emr-containers:StartJobRun"]`。有关更多信息，请参阅 Amazon EMR on EKS 的策略操作。
- 在 Amazon EMR on EKS 服务终端节点中使用的前缀。例如，`emr-containers.us-east-1.amazonaws.com`。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

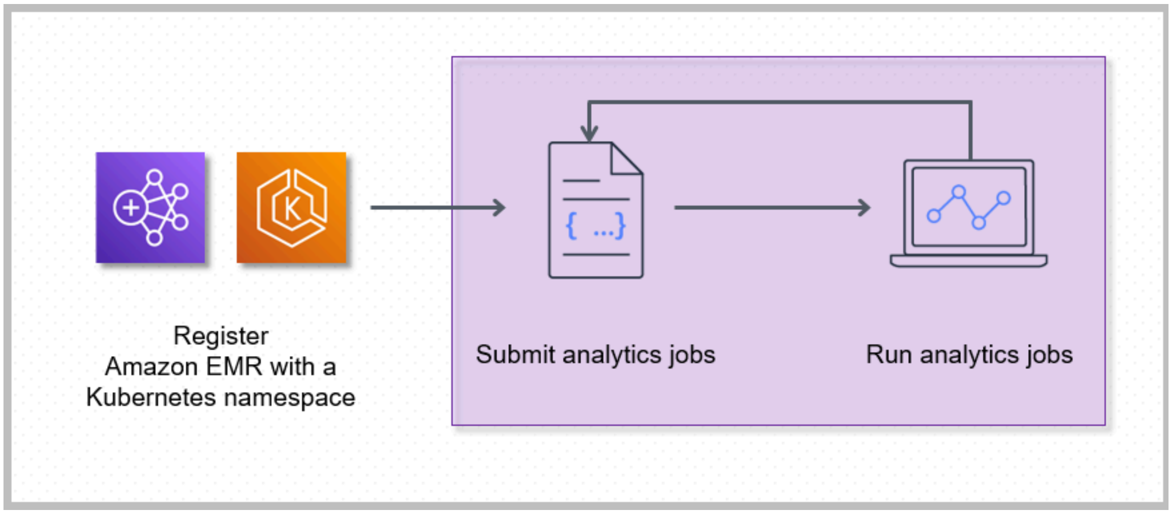
将工作提交到 Amazon EMR on EKS 虚拟集群时会发生什么

使用 Kubernetes 命名空间在 Amazon EKS 上注册 Amazon EMR，以此创建虚拟集群。然后，Amazon EMR 可以在该命名空间上运行分析工作负载。当您使用 Amazon EMR on EKS 将 Spark 提交提交到虚拟集群时，Amazon EMR on EKS 会请求 Amazon EKS 上的 Kubernetes 计划程序来安排 Pod。

以下步骤和图说明了 Amazon EMR on EKS 工作流：

- 使用现有的 Amazon EKS 集群，或通过使用 eksctl 命令行实用程序或 Amazon EKS 控制台来创建 Amazon EKS 集群。

- 通过在 EKS 集群上使用命名空间注册 Amazon EMR 来创建虚拟集群。
- 使用 Amazon CLI 或 SDK 将您的任务提交到虚拟集群。



对于您运行的每个任务，Amazon EMR on EKS 会创建包含 Amazon Linux 2 基础镜像、Apache Spark 和相关依赖项的容器。每个任务都在下载容器的 Pod 中运行，并开始运行 Pod。任务终止后，Pod 也会终止。如果容器的镜像之前已部署到节点，则会使用缓存镜像并绕过下载。日志或指标转发服务器等 Sidecar 容器可部署到 Pod 中。任务终止后，您仍可以使用 Amazon EMR 控制台中的 Spark 应用程序 UI 对其进行调试。

开始使用 Amazon EMR on EKS

本主题通过在虚拟集群上部署 Spark 应用程序，帮助您开始使用 Amazon EMR on EKS。其中包括设置正确权限和启动作业的步骤。开始之前，请确保您已完成设置 Amazon EMR on EKS中的步骤。这可以帮助您在创建虚拟集群之前获得诸如 Amazon CLI 设置之类的工具。有关其他可以帮助您入门的模板，请参阅我们的 EMR 容器最佳实践指南。 GitHub

您将需要安装步骤中的以下信息：

- 使用 Amazon EMR 注册的 Amazon EKS 集群 和 Kubernetes 命名空间的虚拟集群 ID

> **Important**
>
> 创建 EKS 集群时，请使用 m5.xlarge 作为实例类型，或使用具有较高 CPU 和内存的任何其他实例类型。与 m5.xlarge 相比，使用具有较低 CPU 或内存的实例类型可能会由于集群中可用资源不足而导致任务失败。

- 用于执行任务的 IAM 角色名称
- Amazon EMR 发行版的发行版标签（例如：`emr-6.4.0-latest`）
- 用于日志记录和监控的地址目标：
  - Amazon CloudWatch 日志组名称和日志流前缀
  - 用于存储事件和容器日志的 Amazon S3 位置

> **Important**
>
> EKS 任务上的 Amazon EMR 使用亚马逊 CloudWatch 和亚马逊 S3 作为监控和记录的目标目标。您可以通过查看发送到这些地址的任务日志来监控任务进度并排查故障。要想启用日志记录，与执行任务的 IAM 角色关联的 IAM policy 必须具有访问目标资源所需的权限。如果 IAM 策略没有所需的权限，则在运行此示例任务之前更新任务执行角色的信任策略，您必须按照配置任务运行以使用 Amazon S3 日志和将任务运行配置为使用 CloudWatch 日志中概述的步骤操作。

运行 Spark 应用程序

执行以下步骤，即可在 Amazon EMR on EKS 上运行简单的 Spark 应用程序。适用于 Spark Python 应用程序的应用程序 `entryPoint` 文件位于 `s3://REGION.elasticmapreduce/emr-containers/samples/wordcount/scripts/wordcount.py`。*REGION*是您在 EKS 虚拟集群上的 Amazon EMR 所在的区域，例如。*us-east-1*

1. 使用所需的权限更新任务执行角色的 IAM policy，如以下策略语句所示：

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

```
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:*:*:*"
            ]
        },
        {
            "Sid": "WriteToCloudwatchLogs",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:*:*:log-group:my_log_group_name:log-
stream:my_log_stream_prefix/*"
            ]
        }
    ]
}
```

- 本策略的第一个语句`ReadFromLoggingAndInputScriptBuckets`授予`ListBucket`和`GetObjects`访问以下 Amazon S3 存储桶：
  - *`REGION`*`.elasticmapreduce`-应用程序文件所在的存储桶`entryPoint`。
  - *`amzn-s3-demo-destination-bucket`*-您为输出数据定义的存储桶。
  - *`amzn-s3-demo-logging-bucket`*-您为日志数据定义的存储桶。
- 本策略的第二个语句 `WriteToLoggingAndOutputDataBuckets` 授予任务将数据分别写入输出和日志记录存储桶的权限。
- 第三条语句`DescribeAndCreateCloudwatchLogStream`授予任务描述和创建 Amazon CloudWatch 日志的权限。
- 第四条语句`WriteToCloudwatchLogs`授予将日志写入名为的 CloudWatch 日志流*`my_log_group_name`*下名为 Amazon 日志组的权限*`my_log_stream_prefix`*。

2. 要运行 Spark Python 应用程序，请使用以下命令。将所有可替换的*`red italicized`*值替换为适当的值。*`REGION`*是您在 EKS 虚拟集群上的 Amazon EMR 所在的区域，例如。*`us-east-1`*

```
aws emr-containers start-job-run \
```

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

此任务的输出数据将出现在`s3://`*`amzn-s3-demo-destination-bucket`*`/wordcount_output`。

您还可以为任务运行创建具有指定参数的 JSON 文件。然后运行指向 JSON 文件路径的 `start-job-run` 命令。有关更多信息，请参阅 [使用 `StartJobRun` 提交任务运行]。有关配置任务运行参数的更多详细信息，请参阅[配置任务运行的选项]。

3. 要运行 Spark SQL 应用程序，请使用以下命令。将所有*`red italicized`*值替换为适当的值。*`REGION`*是您在 EKS 虚拟集群上的 Amazon EMR 所在的区域，例如。*`us-east-1`*

```
aws emr-containers start-job-run \
--virtual-cluster-id cluster_id \
--name sample-job-name \
--execution-role-arn execution-role-arn \
--release-label emr-6.7.0-latest \
--job-driver '{
```

```
  "sparkSqlJobDriver": {
    "entryPoint": "s3://query-file.sql",
    "sparkSqlParameters": "--conf spark.executor.instances=2 --
conf spark.executor.memory=2G --conf spark.executor.cores=2 --conf
 spark.driver.cores=1"
  }
}' \
--configuration-overrides '{
  "monitoringConfiguration": {
    "cloudWatchMonitoringConfiguration": {
      "logGroupName": "my_log_group_name",
      "logStreamNamePrefix": "my_log_stream_prefix"
    },
    "s3MonitoringConfiguration": {
       "logUri": "s3://amzn-s3-demo-logging-bucket"
    }
  }
}'
```

下面显示了一个示例 SQL 查询文件。您必须有一个外部文件存储，例如 S3，其中存储表的数据。

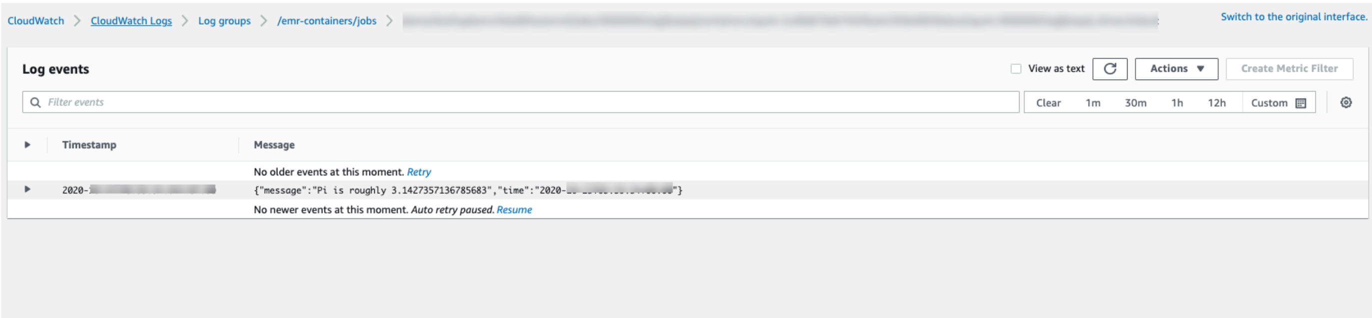
```
CREATE DATABASE demo;
CREATE EXTERNAL TABLE IF NOT EXISTS demo.amazonreview( marketplace string,
 customer_id string, review_id  string, product_id  string, product_parent  string,
 product_title  string, star_rating  integer, helpful_votes  integer, total_votes
  integer, vine  string, verified_purchase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

此任务的输出将在 S3 或驱动程序的 stdout 日志中可用 CloudWatch，具体取决于所配置`monitoringConfiguration`的。

4. 您还可以为任务运行创建具有指定参数的 JSON 文件。然后运行指向 JSON 文件路径的 start-job-run 命令。有关更多信息，请参阅“提交任务运行”。有关配置任务运行参数的更多详细信息，请参阅“配置任务运行的选项”。

   要监控任务进度或调试故障，您可以检查上传到 Amazon S3 的 CloudWatch 日志、日志或两者。请参阅 Amazon S3 中的日志路径[配置任务运行以使用 S3 日志](#)，有关[配置任务运行以使用日志的 Cloudwatch CloudWatch 日志](#)。要在日志中查看 CloudWatch 日志，请按照以下说明进行操作。

- 打开 CloudWatch 控制台，网址为https://console.aws.amazon.com/cloudwatch/。
- 在导航窗格中，选择 Logs (日志)。然后选择 Log groups (日志组)。
- 选择 Amazon EMR on EKS 日志组，然后查看已上载的录入事件。



Important

作业配置了默认的重试策略。有关如何修改或禁用配置的信息，请参阅使用作业重试策略。

EKS 最佳实践指南上的 Amazon EMR 链接 GitHub

我们通过开源社区协作编写了 Amazon EMR on EKS 最佳实践指南，以便我们可以快速迭代，并为创建和运行虚拟集群的各个方面提供建议。我们建议您在这些部分中使用 Amazon EMR on EKS 最佳实践指南。选择每个部分中的链接以转到该 GitHub站点。

安全性

> **Note**
>
> 有关 Amazon EMR on EKS 安全性的更多信息，请参阅 Amazon EMR on EKS 安全最佳实践。

加密最佳实践：如何对静态数据和传输中的数据进行加密。

管理网络安全描述了如何在连接 Amazon RDS 和 Amazon Redshift 等 Amazon Web Services 服务 中托管的数据源时为 Amazon EMR on EKS 的容器组（pod）配置安全组。

使用 Amazon 密钥管理器存储机密。

Pyspark 作业提交

Pyspark 作业提交：使用 zip、egg、wheel 和 pex 等打包格式为 PySpark 应用程序指定不同类型的打包。

存储

使用 EBS 卷：如何对需要 EBS 卷的作业使用静态和动态预置。

使用 Amazon FSx 获取 Lustre 卷：如何为需要亚马逊 FSx 提供 Luster 卷的任务使用静态和动态配置。

使用实例存储卷：如何使用实例存储卷进行作业处理。

元数据仓集成

使用 Hive 元数据仓：提供使用 Hive 元数据仓的不同方式。

使用 Amazon Glue：提供了不同的方式来配置 Amazon Glue 目录。

调试

使用 Spark 调试：如何更改日志级别。

在驱动程序容器器组（pod）上连接到 Spark UI。

如何通过 Amazon EMR on EKS 使用自托管 Spark 历史记录服务器。

排查 Amazon EMR on EKS 中的问题

故障排除。

节点放置

将 Kubernetes 节点选择器用于 `single-az` 和其他使用案例。

使用 Fargate 节点放置。

性能

使用动态资源分配（DRA）。

Amazon VPC 容器网络接口插件（CNI）、Cluster Autoscaler 和 Core DNS 的 EKS 最佳实践。

成本优化

使用竞价型实例：Amazon EC2 竞价型实例最佳实践以及如何使用 Spark 节点停用功能。

使用 Amazon Outposts

使用 EKS 在 EKS 上运行亚马逊 EMR Amazon Outposts

为 Amazon EMR on EKS 自定义 Docker 镜像

您可以使用 Amazon EMR on EKS 自定义 Docker 镜像。自定义 Amazon EMR on EKS 运行时的镜像具有以下优势：

- 将应用程序依赖项和运行时环境打包到单个不可改变容器中，以提高可移植性并简化每个工作负载的依赖项管理。
- 安装并配置针对您的工作负载进行优化的软件包。这些软件包可能不会在 Amazon EMR 运行时的公开分配中广泛使用。
- 将 Amazon EMR on EKS 与组织中当前建立的构建、测试和部署流程（包括本地开发和测试）集成在一起。
- 应用已建立的安全过程（如镜像扫描），以满足组织内的合规性和监管要求。

主题

- 如何自定义 Docker 镜像
- 如何选择基础映像 URI 的详细信息
- 自定义映像的注意事项

如何自定义 Docker 镜像

请按照以下步骤为 Amazon EMR on EKS 自定义 Docker 映像。这些步骤展示了如何获取基础映像、自定义和发布映像，以及如何使用映像提交工作负载。

- 先决条件
- 步骤 1：从 Amazon Elastic Container Registry（Amazon ECR）中检索基础镜像
- 步骤 2：自定义基础镜像
- 步骤 3：（可选但建议）验证自定义镜像
- 步骤 4：发布自定义镜像
- 步骤 5：使用自定义镜像在 Amazon EMR 中提交 Spark 工作负载

> **Note**
>
> 在自定义 Docker 映像时，可能要考虑的其他选项包括为交互式端点进行自定义（以确保拥有所需的依赖项），或使用多架构容器映像：
>
> - 为交互式端点自定义 Docker 映像
> - 使用多架构镜像

先决条件

- 请完成 Amazon EMR on EKS 的设置 Amazon EMR on EKS步骤。
- 在您的环境中安装 Docker。有关更多信息，请参阅获取 Docker。

步骤 1：从 Amazon Elastic Container Registry（Amazon ECR）中检索基础镜像

基础映像包含用于访问其他 Amazon 服务的 Amazon EMR 运行时系统和连接器。对于 Amazon EMR 6.9.0 及更高版本，您可以从 Amazon ECR Public Gallery 获取基础映像。浏览图库以找到映像链接，然后将映像拉到本地工作区。例如，对于 Amazon EMR 7.0 发行版，以下 `docker pull` docker pull 命令将为您提供最新的标准基础映像。您可以将 `emr-7.8.0:latest` 替换为 `emr-7.8.0-spark-rapids:latest`，检索带 Nvidia RAPIDS Accelerator 的映像。您也可以将 `emr-7.8.0:latest` 替换为 `emr-7.8.0-java11:latest`，使用 Java 11 运行时系统检索映像。

```
docker pull public.ecr.aws/emr-on-eks/spark/emr-7.8.0:latest
```

如果您想检索 Amazon EMR 6.9.0 或更早版本的基础映像，或者如果您希望从每个区域中的 Amazon ECR 注册账户中检索，请使用以下步骤：

1. 选择基础镜像 URI。镜像 URI 遵循格式（*ECR-registry-account*.dkr.ecr.*Region*.amazonaws.com/spark/*container-image-tag*），示例如下。

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

   要选择您所在区域的基础镜像，请参阅如何选择基础映像 URI 的详细信息。

2. 登录存储基础镜像的 Amazon ECR 存储库。*us-west-2*用*895885662937*所选的容器镜像标签 Amazon 替换。

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

3. 将基础镜像拉入本地 Workspace。*emr-6.6.0:latest*用所选的容器镜像标签替换 emr-6.0:

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

步骤 2：自定义基础镜像

请按照以下步骤自定义您从 Amazon ECR 中拉取的基础映像。

1. 在您的本地 Workspace 上创建新的 `Dockerfile`。
2. 编辑您刚刚创建的 `Dockerfile` 并添加以下内容。该 `Dockerfile` 使用您从 `895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.6.0:latest` 中提取的容器镜像。

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

3. 将命令添加到 `Dockerfile` 以自定义基础镜像。例如，添加命令来安装 Python 库，如以下 `Dockerfile` 所示。

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

4. 在创建 `Dockerfile` 的同一目录中，请运行以下命令构建 Docker 镜像。为 Docker 镜像提供一个名称，例如 emr6.5_custom。*emr6.6_custom*

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

步骤 3：（可选但建议）验证自定义镜像

我们建议您在发布自定义镜像之前测试它的兼容性。您可以使用 Amazon EMR on EKS 自定义镜像 CLI 检查您的镜像是否具有在 Amazon EMR on EKS 上运行所需要的文件结构和正确配置。

> **Note**
>
> Amazon EMR on EKS 的自定义镜像 CLI 无法确认您的镜像没有错误。在删除基础镜像的依赖项时要格外小心。

可以执行以下步骤，验证您的自定义镜像。

1. 下载并安装 Amazon EMR on EKS 自定义镜像 CLI。有关更多信息，请参阅 Amazon EMR on EKS 自定义镜像 CLI 安装指南。
2. 运行以下命令以测试安装。

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

   下面是此类输出的示例。

   ```
   Amazon EMR on EKS Custom Image CLI
   Version: x.xx
   ```
3. 运行以下命令以验证您的自定义镜像。

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

   - `-i` 指定需要验证的本地镜像 URI。它可以是镜像 URI，也可以是为镜像定义的任何名称或标签。
   - `-r` 为基础镜像指定确切的发布版本，例如 `emr-6.6.0-latest`。
   - `-t` 指定镜像类型。如果这是 Spark 镜像，请输入 `spark`。默认值为 `spark`。当前 Amazon EMR on EKS 自定义镜像 CLI 版本仅支持 Spark 运行时镜像。

   如果您成功运行命令并且自定义镜像满足所有必需的配置和文件结构，则返回的输出会显示所有测试结果，如以下示例所示。

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

如果自定义镜像不满足所需的配置或文件结构，则会出现错误消息。返回的输出提供了错误配置或文件结构相关信息。

步骤 4：发布自定义镜像

将新的 Docker 镜像发布到您的 Amazon ECR 注册表。

1. 运行以下命令创建用于存储 Docker 镜像的 Amazon ECR 存储库。为存储库提供一个名称，例如 emr6.2_custom_rep *emr6.6_custom_repo* o。将 *us-west-2* 替换为您的区域。

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

   有关更多信息，请参阅《Amazon ECR 用户指南》中的创建存储库。

2. 运行以下命令对您的默认注册表进行身份验证。

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

有关更多信息，请参阅《Amazon ECR 用户指南》中的对您的默认注册表进行身份验证。

3. 标记镜像并将其发布到您创建的 Amazon ECR 存储库。

   标记镜像。

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

   推送镜像。

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

   有关更多信息，请参阅《Amazon ECR 用户指南》中的将镜像推送到 Amazon ECR。

步骤 5：使用自定义镜像在 Amazon EMR 中提交 Spark 工作负载

构建和发布自定义镜像后，您可以使用自定义镜像提交 Amazon EMR on EKS 任务。

首先，创建 start-job-run-request start.imag `spark.kubernetes.container.image` e 参数来引用自定义镜像，如下面的 JSON 文件所示。

> **Note**
>
> 您可以使用 `local://` 方案来引用自定义镜像中的可用文件，如下面所示的 JSON 片段中的 `entryPoint` 参数。您也可以使用 `local://` 方案来引用应用程序依赖项。使用 `local://` 方案引用的所有文件和依赖项必须已存在于自定义镜像的指定路径。

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

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

您还可以使用 applicationConfiguration 属性来引用自定义映像，如以下示例所示。

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

然后运行 `start-job-run` 命令提交任务。

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

将上面 JSON 示例中的 emr6.0-lates *emr-6.6.0-latest* t 替换为您的 Amazon EMR 发行版的版本。我们强烈建议您使用 `-latest` 发布版本，以确保所选版本包含最新的安全更新。有关 Amazon EMR 发行版本和相应映像标签的更多信息，请参阅 如何选择基础映像 URI 的详细信息。

> **Note**
>
> 您可以使用 `spark.kubernetes.driver.container.image` 和 `spark.kubernetes.executor.container.image` 为驱动程序和执行程序 Pod 指定不同的镜像。

为交互式端点自定义 Docker 映像

您还可以为交互式端点自定义 Docker 映像，以便运行自定义的基础内核映像。这有助于确保在从 EMR Studio 运行交互式工作负载时拥有所需的依赖项。

1. 按照上述第 1-4 步进行操作，以自定义 Docker 镜像。对于 Amazon EMR 6.9.0 版本及更高版本，您可以从 Amazon ECR Public Gallery 获取基础映像 URI。对于 Amazon EMR 6.9.0 之前的版本，您可以在每个 Amazon Web Services 区域的 Amazon ECR 注册账户中获取映像，唯一的区别是 Dockerfile 中的基础映像 URI。基础映像 URI 遵循以下格式：

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

   您需要在基本镜像 URI 中使用 `notebook-spark`，而不是 `spark`。基本镜像包含 Spark 运行时，以及随之运行的 Notebook 内核。有关选择区域和容器镜像标签的更多信息，请参阅 如何选择基础映像 URI 的详细信息。

   > **Note**
   >
   > 目前仅支持覆盖基础镜像，除 Amazon 提供的基础镜像之外，不支持引入其他类型的全新内核。

2. 创建可与自定义映像配合使用的交互式端点。

首先，使用以下内容创建名为 `custom-image-managed-endpoint.json` 的 JSON 文件。

```
{
    "name": "endpoint-name",
    "virtualClusterId": "virtual-cluster-id",
    "type": "JUPYTER_ENTERPRISE_GATEWAY",
    "releaseLabel": "emr-6.6.0-latest",
    "executionRoleArn": "execution-role-arn",
    "certificateArn": "certificate-arn",
    "configurationOverrides": {
        "applicationConfiguration": [
            {
                "classification": "jupyter-kernel-overrides",
                "configurations": [
                    {
                        "classification": "python3",
                        "properties": {
                            "container-image": "123456789012.dkr.ecr.us-
west-2.amazonaws.com/custom-notebook-python:latest"
                        }
                    },
                    {
                        "classification": "spark-python-kubernetes",
                        "properties": {
                            "container-image": "123456789012.dkr.ecr.us-
west-2.amazonaws.com/custom-notebook-spark:latest"
                        }
                    }
                ]
            }
        ]
    }
}
```

然后，使用该 JSON 文件中指定的配置创建交互式端点，如以下示例所示。

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

有关更多信息，请参阅为虚拟集群创建交互式端点。

3. 通过 EMR Studio 连接到交互式端点。有关更多信息，请参阅从 Studio 连接。

使用多架构镜像

Amazon EMR on EKS 支持适用于 Amazon Elastic Container Registry (Amazon ECR) 的多架构容器镜像。有关更多信息，请参阅适用于 Amazon ECR 的多架构容器镜像简介。

Amazon EMR on EMR on EMR on EMR on EKS 和非基于 Amazon Graviton EC2 的 EC2 实例。non-Graviton-based EC2 基于 Graviton 的镜像与非基于 Graviton 的镜像存储在 Amazon EMR 中的相同镜像存储库中。 non-Graviton-based

例如，要检查 Docker 清单列表中是否有 6.6.0 镜像，请运行下面的命令。

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

下面是输出。`arm64` 架构适用于 Graviton 实例。`amd64` 适用于非 Graviton 实例。

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

请执行以下步骤以创建多架构镜像：

1. 创建包含以下内容的 `Dockerfile`，以便您能拉取 `arm64` 映像。

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

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

2. 按照适用于 Amazon ECR 的多架构容器器镜像简介中的说明操作，构建一个多架构镜像。

> **Note**
>
> 您必须在 `arm64` 实例上创建 `arm64` 映像。同样，您必须在 `amd64` 实例上构建 `amd64` 映像。

   您还可以使用 Docker `buildx` 命令构建多架构映像，而无需基于每种特定实例类型上进行构建。有关更多信息，请参阅利用多 CPU 架构支持。

3. 构建多架构镜像后，您可以使用相同的 `spark.kubernetes.container.image` 参数并指向该映像来提交任务。在同时包括 Amazon 基于 Graviton 和非基于 Graviton 的 non-Graviton-based EC2 EC2 实例的异构集群中，实例将根据拉取该镜像的实例架构确定正确的架构映像。

如何选择基础映像 URI 的详细信息

> **Note**
>
> 对于 Amazon EMR 6.9.0 版本及更高版本，您可以从 Amazon ECR Public Gallery 中检索基础映像，因此您无需按照本页上的说明构造基础映像 URI。要查找基础映像的容器映像标签，请参阅 Amazon EMR on EKS 相应版本的发布说明页面。

所选基础 Docker 映像存储在 Amazon Elastic Container Registry（Amazon ECR）。映像 URI 遵循此格式：*`ECR-registry-account`*`.dkr.ecr.`*`Region`*`.amazonaws.com.cn/spark/`*`container-image-tag`*

以下示例展示了此格式：

```
895885662937.dkr.ecr.us-west-2.amazonaws.com.cn/spark/emr-7.8.0:latest
```

适用于交互式端点的映像 URI 遵循此格式：*ECR-registry-account*.dkr.ecr.*Region*.amazonaws.com.cn/notebook-spark/*container-image-tag*，如以下示例所示。您需要在基本镜像 URI 中使用 notebook-spark，而不是 spark。

```
895885662937.dkr.ecr.us-west-2.amazonaws.com.cn/notebook-spark/emr-7.8.0:latest
```

同样，对于交互式端点的非 Spark python3 映像，映像 URI 为 *ECR-registry-account*.dkr.ecr.*Region*.amazonaws.com.cn/notebook-python/*container-image-tag*。以下示例 URI 格式正确：

```
895885662937.dkr.ecr.us-west-2.amazonaws.com.cn/notebook-python/emr-7.8.0:latest
```

要查找基础映像的容器映像标签，请参阅 Amazon EMR on EKS 相应版本的发布说明页面。

按区域划分的 Amazon ECR 注册账户

为了避免高网络延迟，您应该从最近的拉取基础映像 Amazon Web Services 区域。根据下表，选择与您从中拉取映像区域对应的 Amazon ECR 注册账户。

区域	Amazon ECR 注册账户	
ap-east-1	736135916053	
ap-northeast-1	059004520145	
ap-northeast-2	996579266876	
ap-northeast-3	705689932349	
ap-southeast-3	946962994502	
ap-south-1	235914868574	
ap-south-2	691480105545	
ap-southeast-1	671219180197	

区域	Amazon ECR 注册账户
ap-southeast-2	038297999601
ca-central-1	351826393999
eu-central-1	107292555468
eu-central-2	314408114945
eu-north-1	830386416364
eu-west-1	483788554619
eu-west-2	118780647275
eu-west-3	307523725174
eu-south-1	238014973495
eu-south-2	350796622945
il-central-1	395734710648
me-south-1	008085056818
me-central-1	818935616732
sa-east-1	052806832358
us-gov-west-1	299385240661
us-gov-east-1	299393998622
us-east-1	755674844232
us-east-2	711395599931
us-west-1	608033475327
us-west-2	895885662937

区域	Amazon ECR 注册账户
af-south-1	358491847878
cn-north-1	068337069695
cn-northwest-1	068420816659

自定义映像的注意事项

自定义 Docker 映像时，您可精细地选择精确的任务运行时间。使用此功能时，请考虑以下最佳实践。其中包括安全、配置和安装映像方面的注意事项：

- 安全性是 Amazon 和您的共同责任。您负责对添加到映像中的二进制文件进行安全修补。按照Amazon EMR on EKS 安全最佳实践中的说明操作，特别是获取自定义镜像的最新安全更新和采用最低特权原则。
- 自定义基础映像时，请务必将 Docker 用户更改为 `hadoop:hadoop`，以确保不使用根用户运行任务。
- Amazon EMR on EKS 会在运行时将文件挂载到映像的配置之上，例如 `spark-defaults.conf`。要覆盖这些配置文件，我们建议您在提交任务期间使用 `applicationOverrides` 参数，而不是仅直接在自定义映像中修改文件。
- Amazon EMR on EKS 会在运行时挂载某些文件夹。对这些文件夹所做的任何修改都不可用于容器中。如果要为自定义映像添加应用程序或其依赖项，我们建议您选择不属于以下预定义路径的目录：
  - `/var/log/fluentd`
  - `/var/log/spark/user`
  - `/var/log/spark/apps`
  - `/mnt`
  - `/tmp`
  - `/home/hadoop`
- 您可以将自定义映像上载到任何兼容 Docker 的存储库，例如 Amazon ECR、Docker Hub 或私有企业存储库。有关使用所选 Docker 存储库配置 Amazon EKS 集群身份验证的更多信息，请参阅从私有仓库拉取镜像。

使用 Amazon EMR on EKS 运行 Flink 任务

Amazon EMR 版本 6.13.0 及更高版本支持带 Apache Flink 的 Amazon EMR on EKS 或 Flink Kubernetes 运算符作为 Amazon EMR on EKS 的任务提交模型。借助带 Apache Flink 的 Amazon EMR on EKS，您可以在自己的 Amazon EKS 集群上使用 Amazon EMR 发行版运行时系统来部署和管理 Flink 应用程序。在 Amazon EKS 集群中部署 Flink Kubernetes 运算符后，您可以直接向此运算符提交 Flink 应用程序。运算符管理 Flink 应用程序的生命周期。

主题

- 设置和使用 Flink Kubernetes Operator
- 使用 Flink Native Kubernetes
- 为 Flink 和 FluentD 自定义 Docker 映像
- 监视 Flink Kubernetes Operator 和 Flink 任务
- Flink 如何支持高可用性和作业弹性
- 在 Flink 应用程序中使用 Autoscaler
- Amazon EMR on EKS 上 Flink 作业的维护和故障排除
- 支持将 Amazon EMR on EKS 与 Apache Flink 结合使用的发行版

设置和使用 Flink Kubernetes Operator

以下页面旨在介绍如何在 Amazon EMR on EKS 上设置并使用 Flink Kubernetes Operator 来运行 Flink 任务。提供的主题包括所需的先决条件、如何设置环境以及在 Amazon EMR on EKS 上运行 Flink 应用程序。

主题

- 设置 Amazon EMR on EKS 的 Flink Kubernetes Operator
- 安装 Amazon EMR on EKS 的 Flink Kubernetes Operator
- 运行 Flink 应用程序
- 运行 Flink 应用程序的安全角色权限
- 卸载 Amazon EMR on EKS 的 Flink Kubernetes Operator

设置 Amazon EMR on EKS 的 Flink Kubernetes Operator

在 Amazon EKS 上安装 Flink Kubernetes Operator 之前，请执行下述任务来完成设置。如果已注册 Amazon Web Services（Amazon）并且一直在使用 Amazon EKS，您基本上就准备好使用 Amazon EMR on EKS 了。完成以下任务，在 Amazon EKS 上设置好 Flink Operator。跳过已完成的先决条件，转到下一个先决条件。

- 安装或更新到最新版本的 Amazon CLI — 如果您已经安装了 Amazon CLI，请确认您安装的是最新版本。
- 设置 kubectl 和 eksctl：eksctl 是用来与 Amazon EKS 通信的命令行工具。
- Install Helm – Kubernetes 的 Helm 包管理器可帮助您在 Kubernetes 集群上安装和管理应用程序。
- 开始使用 Amazon EKS - eksctl：按照以下步骤在 Amazon EKS 中创建带有节点的新 Kubernetes 集群。
- 选择 Amazon EMR 发行版标签（6.13.0 或更高版本）：Amazon EMR 6.13.0 及更高版本支持 Flink Kubernetes Operator。
- 在 Amazon EKS 集群上启用服务账户的 IAM 角色（IRSA）。
- 创建任务执行角色。
- 更新任务执行角色的信任策略。
- 创建 Operator 执行角色。此为可选步骤。Flink 任务和 Operator 可以使用相同的角色。如果想为 Operator 设置不同的 IAM 角色，可以单独创建一个角色。
- 更新 Operator 执行角色的信任策略。必须为要用于 Amazon EMR Flink Kubernetes Operator 服务账户的角色显式添加一个信任策略条目。可以按照如下示例格式进行操作：

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
           "Effect": "Allow",
           "Principal": {
               "Federated": "arn:aws:iam::ACCOUNT_ID:oidc-provider/OIDC_PROVIDER"
           },
           "Action": "sts:AssumeRoleWithWebIdentity",
           "Condition": {
               "StringLike": {
                   "OIDC_PROVIDER:sub": "system:serviceaccount:NAMESPACE:emr-
containers-sa-flink-operator"
               }
```

```
            }
        }
    ]
}
```

安装 Amazon EMR on EKS 的 Flink Kubernetes Operator

本主题通过准备 Flink 部署，帮助您开始在 Amazon EKS 上使用 Flink Kubernetes Operator。

安装 Kubernetes Operator

按照以下步骤安装 Apache Flink 版 Kubernetes Operator。

1. 如果尚未执行此操作，请完成 the section called “设置” 中的步骤。
2. 安装 cert-manager*cert-manager*（每个 Amazon EKS 集群安装一次）以允许添加 webhook 组件。

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

3. 安装 Helm 图表。

```
export VERSION=7.8.0 # The Amazon EMR release version
export NAMESPACE=The Kubernetes namespace to deploy the operator

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

输出示例：

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

4. 等待部署完成并验证图表安装情况。

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

5. 部署完成后，您应该会看到如下消息。

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

6. 使用以下命令查看部署的 Operator。

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

示例输出如下，其中应用程序版本 `x.y.z-amzn-n` 将与您的 Amazon EMR on EKS 发行版的 Flink Operator 版本对应。有关更多信息，请参阅 支持将 Amazon EMR on EKS 与 Apache Flink 结合使用的发行版。

```
NAME                              NAMESPACE     REVISION     UPDATED
               STATUS       CHART                                   APP VERSION

flink-kubernetes-operator     $NAMESPACE    1           2023-02-22 16:43:45.24148
 -0500 EST     deployed     flink-kubernetes-operator-emr-7.8.0     x.y.z-amzn-n
```

运行 Flink 应用程序

在 Amazon EMR 6.13.0 及更高版本中，您可以在应用程序模式下在 EKS 上的 Amazon EMR 上使用 Flink Kubernetes Operator 来运行 Flink 应用程序。在 Amazon EMR 6.15.0 及更高版本中，您还可以在会话模式中运行 Flink 应用程序。本页介绍您使用 EKS 上的 Amazon EMR 运行 Flink 应用程序时可用的两种方法。

> **Note**
>
> 提交 Flink 作业时，必须使用一个 Amazon S3 存储桶来存储高可用性元数据。如果不想使用此功能，可以将其禁用。系统会默认启用该功能。

先决条件 – 在使用 Flink Native Kubernetes Operator 运行 Flink 应用程序之前，请先完成 the section called “设置” 和 the section called “安装 Kubernetes Operator” 中的步骤。

Application mode

在 Amazon EMR 6.13.0 及更高版本中，您可以在应用程序模式下在 EKS 上的 Amazon EMR 上使用 Flink Kubernetes Operator 来运行 Flink 应用程序。

1. 创建一个名为 basic-example-app-cluster.yaml 的 FlinkDeployment 定义文件，如下例所示。如果您已激活并使用其中一个 Amazon Web Services 区域，务必取消注释并配置。fs.s3a.endpoint.region

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: basic-example-app-cluster
spec:
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
    #fs.s3a.endpoint.region: OPT_IN_AWS_REGION_NAME
    state.checkpoints.dir: CHECKPOINT_S3_STORAGE_PATH
    state.savepoints.dir: SAVEPOINT_S3_STORAGE_PATH
  flinkVersion: v1_17
  executionRoleArn: JOB_EXECUTION_ROLE_ARN
  emrReleaseLabel: "emr-6.13.0-flink-latest" # 6.13 or higher
  jobManager:
    storageDir: HIGH_AVAILABILITY_STORAGE_PATH
    resource:
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
  job:
    # if you have your job jar in S3 bucket you can use that path as well
    jarURI: local:///opt/flink/examples/streaming/StateMachineExample.jar
    parallelism: 2
    upgradeMode: savepoint
    savepointTriggerNonce: 0
  monitoringConfiguration:
    cloudWatchMonitoringConfiguration:
       logGroupName: LOG_GROUP_NAME
```

2. 使用如下命令提交 Flink 部署。此操作还会创建一个名为 basic-example-app-cluster 的 FlinkDeployment 对象。

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

3. 访问 Flink UI。

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

4. 打开 localhost:8081 即可在本地查看 Flink 任务。
5. 清理任务。记得清理为此任务创建的 S3 构件，例如检查点、保存点元数据和 CloudWatch 日志。 CloudWatch

有关通过 Flink Kubernetes Operator 向 Flink 提交应用程序的更多信息，请在 GitHub 上参阅 apache/fl ink-kubernetes-operat or examples apache/flink-kubernetes-operator GitHub

Session mode

在 Amazon EMR 6.15.0 及更高版本中，您可以在会话模式下在 EKS 上的 Amazon EMR 上使用 Flink Kubernetes Operator 来运行 Flink 应用程序。

1. 创建一个名为 basic-example-app-cluster.yaml 的 FlinkDeployment 定义文件，如下例所示。如果您已激活并使用其中一个 Amazon Web Services 区域，务必取消注释并配置。fs.s3a.endpoint.region

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

```
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
  monitoringConfiguration:
    s3MonitoringConfiguration:
       logUri:
    cloudWatchMonitoringConfiguration:
       logGroupName: LOG_GROUP_NAME
```

2. 使用如下命令提交 Flink 部署。此操作还会创建一个名为 `basic-example-session-cluster` 的 `FlinkDeployment` 对象。

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

3. 使用以下命令确认会话集群 `LIFECYCLE` 是 `STABLE`：

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

该输出应该类似于以下示例：

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

4. 使用以下示例内容创建 `FlinkSessionJob` 自定义定义资源文件 `basic-session-job.yaml`：

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkSessionJob
metadata:
  name: basic-session-job
spec:
  deploymentName: basic-session-deployment
  job:
    # If you have your job jar in an S3 bucket you can use that path.
    # To use jar in S3 bucket, set
    # OPERATOR_EXECUTION_ROLE_ARN (--set emrContainers.operatorExecutionRoleArn=
$OPERATOR_EXECUTION_ROLE_ARN)
    # when you install Spark operator
```

```
    jarURI: https://repo1.maven.org/maven2/org/apache/flink/flink-examples-
streaming_2.12/1.16.1/flink-examples-streaming_2.12-1.16.1-TopSpeedWindowing.jar
    parallelism: 2
    upgradeMode: stateless
```

5. 使用如下命令提交 Flink 会话作业。此操作将会创建一个 `FlinkSessionJob` 对象 `basic-session-job`。

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

6. 使用以下命令确认会话集群 `LIFECYCLE` 是 `STABLE`，`JOB STATUS` 是 `RUNNING`：

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

   该输出应该类似于以下示例：

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

7. 访问 Flink UI。

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

8. 打开 `localhost:8081` 即可在本地查看 Flink 任务。
9. 清理任务。记得清理为此任务创建的 S3 构件，例如检查点、保存点元数据和 CloudWatch 日志。 CloudWatch

运行 Flink 应用程序的安全角色权限

本主题介绍了部署和运行 Flink 应用程序的安全角色。管理部署以及创建和管理作业需要两个角色：Operator 和 Job。本主题介绍了两个角色并列出了其权限。

基于角色的访问控制

要部署 Operator 并运行 Flink 任务，必须创建两个 Kubernetes 角色：一个 Operator 角色和一个任务角色。安装 Operator 时 Amazon EMR 会默认创建这两个角色。

Operator 角色

我们使用 Operator 角色管理 flinkde `flinkdeployments` ployments， JobManager 为每个 Flink 任务和其他资源（例如服务）创建并管理 JobManager。

Operator 角色的默认名称为 `emr-containers-sa-flink-operator` 并且需要以下权限。

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

```
  resources:
  - flinkdeployments
  - flinkdeployments/status
  - flinksessionjobs
  - flinksessionjobs/status
  verbs:
  - '*'
- apiGroups:
  - networking.k8s.io
  resources:
  - ingresses
  verbs:
  - '*'
- apiGroups:
  - coordination.k8s.io
  resources:
  - leases
  verbs:
  - '*'
```

任务角色

JobManager 使用作业角色 ConfigMaps 为每个作业创建和管理 TaskManagers 和。

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

卸载 Amazon EMR on EKS 的 Flink Kubernetes Operator

按照如下步骤卸载 Flink Kubernetes Operator。

1. 删除 Operator。

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

2. 删除 Helm 未卸载的 Kubernetes 资源。

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

3. （可选）删除 cert-manager。

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

使用 Flink Native Kubernetes

Amazon EMR 6.13.0 及更高版本都支持 Flink Native Kubernetes 作为命令行工具，而您可以使用该工具向 Amazon EMR on EKS 集群提交和执行 Flink 应用程序。

主题

- 设置 Amazon EMR on EKS 的 Flink Native Kubernetes
- Amazon EMR on EKS 的 Flink Native Kubernetes 入门
- 原生 Kubernetes 的 Flink JobManager 服务账号安全要求

设置 Amazon EMR on EKS 的 Flink Native Kubernetes

先执行以下任务来完成设置，才能在 Amazon EMR on EKS 上使用 Flink CLI 运行应用程序。如果已注册 Amazon Web Services（Amazon）并且一直在使用 Amazon EKS，您基本上就准备好使用 Amazon EMR on EKS 了。跳过已完成的先决条件，转到下一个先决条件。

- 安装或更新到最新版本的 Amazon CLI — 如果您已经安装了 Amazon CLI，请确认您安装的是最新版本。
- 开始使用 Amazon EKS - eksctl：按照以下步骤在 Amazon EKS 中创建带有节点的新 Kubernetes 集群。

- 选择 Amazon EMR 基础映像 URI（6.13.0 或更高版本）– Amazon EMR 6.13.0 及更高版本都支持 Flink Kubernetes 命令。
- 确认 JobManager 服务帐号具有创建和监视 TaskManager pod 的相应权限。有关更多信息，请参阅原生 Kubernetes 的 Flink JobManager 服务账号安全要求。
- 设置本地 Amazon 凭证配置文件。
- 为 Amazon EKS 集群创建或更新 kubeconfig 文件，如果要在该集群上运行 Flink 应用程序。

Amazon EMR on EKS 的 Flink Native Kubernetes 入门

这些步骤展示了如何配置和运行 Flink 应用程序并为其设置服务账户。Flink Native Kubernetes 用于在运行的 Kubernetes 集群上部署 Flink。

配置和运行 Flink 应用程序

Amazon EMR 6.13.0 及更高版本都支持 Flink Native Kubernetes 在 Amazon EKS 集群上运行 Flink 应用程序。要运行 Flink 应用程序，请按照下述步骤操作：

1. 在使用 Flink Native Kubernetes 命令运行 Flink 应用程序之前，请先完成 the section called “设置” 中的步骤。
2. 下载并安装 Flink。
3. 设置以下环境变量的值。

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

4. 创建服务账户来管理 Kubernetes 资源。

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

5. 运行 `run-application` CLI 命令。

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

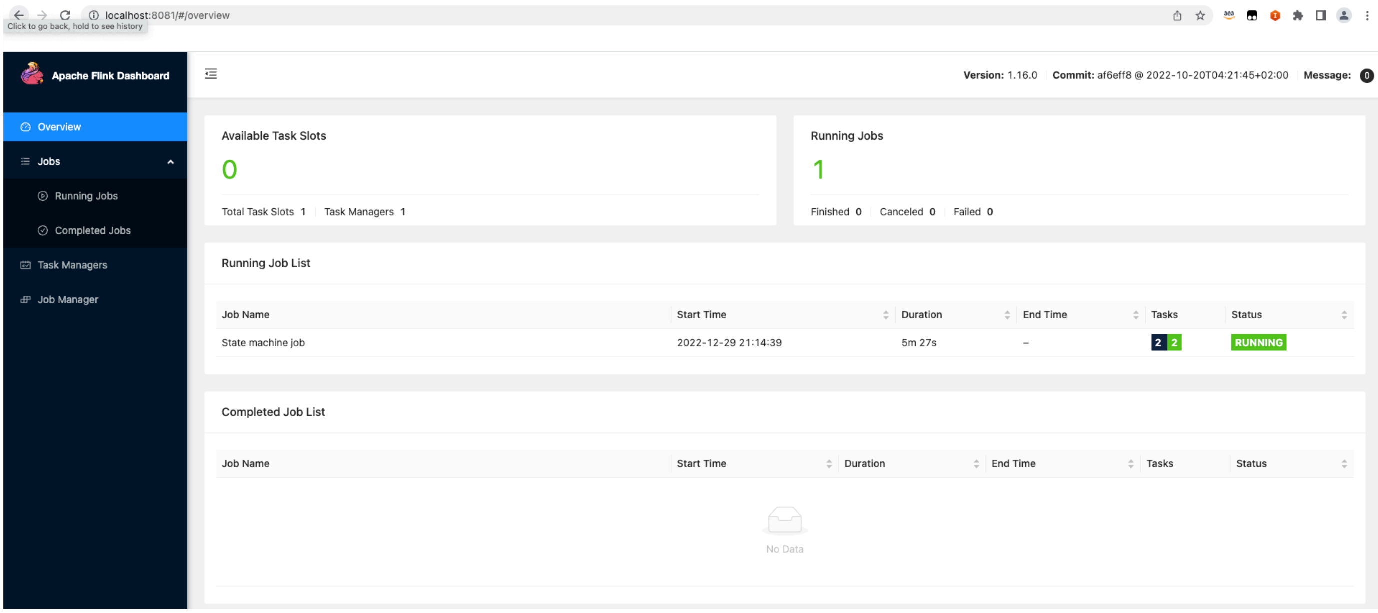
6. 检查创建好的 Kubernetes 资源。

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

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

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

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

7. 端口转发到 8081。

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

8. 在本地访问 Flink UI。



9. 删除 Flink 应用程序。

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

有关向 Flink 提交应用程序的更多信息，请参阅 Apache Flink 文档中的 [Native Kubernetes](https://nightlies.apache.org/flink/flink-docs-master/docs/deployment/resource-providers/native_kubernetes/)。

原生 Kubernetes 的 Flink JobManager 服务账号安全要求

Flink JobManager 容器使用 Kubernetes 服务账号访问 Kubernetes API 服务器来创建和监视 Pod。TaskManager JobManager 服务帐号必须具有适当的权限才能创建/删除 TaskManager pod，并允许 to TaskManager watch 领导者 ConfigMaps 检索集群 ResourceManager 中 JobManager 和的地址。

以下规则适用于此服务账户。

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

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

为 Flink 和 FluentD 自定义 Docker 映像

按照以下步骤，使用 Apache Flink 或 FluentD 映像为 Amazon EMR on EKS 自定义 Docker 映像。其中包括获取基础映像、自定义基础映像、发布基础映像和提交工作负载的技术指导。

主题

- 先决条件
- 步骤 1：从 Amazon Elastic Container Registry 中检索基础映像
- 步骤 2：自定义基础镜像
- 步骤 3：发布自定义映像
- 步骤 4：使用自定义映像在 Amazon EMR 中提交 Flink 工作负载

先决条件

在自定义 Docker 映像之前，确保您已完成以下先决条件：

- 已完成设置 Amazon EMR on EKS 的 Flink Kubernetes Operator 步骤。

- 已在环境中安装 Docker。有关更多信息，请参阅获取 Docker。

步骤 1：从 Amazon Elastic Container Registry 中检索基础映像

基础映像包含访问其他 Amazon Web Services 服务所需的 Amazon EMR 运行时和连接器。如果在 Flink 6.14.0 或更高版本中使用 Amazon EMR on EKS，可从 Amazon ECR 公开映像浏览馆获取基础映像。浏览图库以找到映像链接，然后将映像拉到本地 Workspace。例如，对于 Amazon EMR 6.14.0 发行版，以下 `docker pull` 命令将返回最新的标准基础映像。将 `emr-6.14.0:latest` 替换为所需的发行版。

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

以下是 Flink 库映像和 Fluentd 库映像的链接：

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

步骤 2：自定义基础镜像

下列步骤描述了如何自定义从 Amazon ECR 中拉取的基础映像。

1. 在您的本地 Workspace 上创建新的 `Dockerfile`。
2. 编辑 `Dockerfile` 并添加以下内容。该 `Dockerfile` 使用您从 `public.ecr.aws/emr-on-eks/flink/emr-7.9.0-flink:latest` 中拉取的容器映像。

    ```
    FROM public.ecr.aws/emr-on-eks/flink/emr-7.9.0-flink:latest
    USER root
    ### Add customization commands here ####
    USER hadoop:hadoop
    ```

    如果您使用的是 `Fluentd`，请使用以下配置。

    ```
    FROM public.ecr.aws/emr-on-eks/fluentd/emr-7.9.0:latest
    USER root
    ### Add customization commands here ####
    USER hadoop:hadoop
    ```

3. 将命令添加到 `Dockerfile` 以自定义基础镜像。以下命令演示了如何安装 Python 库。

```
FROM public.ecr.aws/emr-on-eks/flink/emr-7.9.0-flink:latest
USER root
RUN pip3 install --upgrade boto3 pandas numpy // For python 3
USER hadoop:hadoop
```

4. 在创建 DockerFile 的同一目录中，运行以下命令构建 Docker 映像。您在 -t 标志后提供的字段是您自定义的映像名称。

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

步骤 3：发布自定义映像

现在，将新的 Docker 映像发布到您的 Amazon ECR 注册表。

1. 运行以下命令创建一个 Amazon ECR 存储库，来存储您的 Docker 映像。为存储库命名，例如 emr_custom_repo.。有关更多信息，请参阅《Amazon Elastic Container Registry 用户指南》中的创建存储库。

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

2. 运行以下命令对您的默认注册表进行身份验证。有关更多信息，请参阅《Amazon Elastic Container Registry 用户指南》中的对您的默认注册表进行身份验证。

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

3. 推送镜像。有关更多信息，请参阅《Amazon Elastic Container Registry 用户指南》中的将映像推送到 Amazon ECR。

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

步骤 4：使用自定义映像在 Amazon EMR 中提交 Flink 工作负载

要使用自定义映像，请对 FlinkDeployment 规范进行以下更改。为此，请在部署规范的 spec.image 行中输入您的映像。

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

要在 Fluentd 作业中使用自定义映像，请在部署规范的 monitoringConfiguration.image 行中输入您的映像。

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

监视 Flink Kubernetes Operator 和 Flink 任务

本节旨在介绍在 Amazon EMR on EKS 上监控 Flink 任务的几种方法。其中包括将 Flink 与 Amazon Managed Service for Prometheus 集成，使用提供作业状态和指标的 Flink Web 控制面板，或者使用监控配置将日志数据发送到 Amazon S3 和 Amazon CloudWatch。

主题

- 使用 Amazon Managed Service for Prometheus 监控 Flink 作业
- 使用 Flink UI 监控 Flink 作业
- 使用监控配置监控 Flink Kubernetes Operator 和 Flink 作业

使用 Amazon Managed Service for Prometheus 监控 Flink 作业

您可以将 Apache Flink 与 Amazon Managed Service for Prometheus（管理门户）集成。Amazon Managed Service for Prometheus 支持从在 Amazon EKS 上运行的集群中摄取 Amazon Managed Service for Prometheus 服务器的指标。Amazon Managed Service for Prometheus 可与已 Amazon EKS 集群上运行的 Prometheus 服务器配合使用。运行集成了 Amazon EMR Flink Operator 的 Amazon Managed Service for Prometheus 会自动部署并配置 Prometheus 服务器，使其与 Amazon Managed Service for Prometheus 集成。

1. 创建 Amazon Managed Service for Prometheus Workspace。此工作空间用作摄取端点。稍后需要远程写入 URL。
2. 设置服务账户的 IAM 角色。

   要使用这种入门方法，请使用运行 Prometheus 服务器的 Amazon EKS 集群中服务账户的 IAM 角色。此类角色又称为服务角色。

   如果尚无此类角色，请设置服务角色从 Amazon EKS 集群中摄取指标。

   请创建一个名为 `amp-iamproxy-ingest-role` 的 IAM 角色，然后再继续操作。
3. 使用 Amazon Managed Service for Prometheus 安装 Amazon EMR Flink Operator。

您已经拥有了一个 Amazon Managed Service for Prometheus Workspace，一个专用于 Amazon Managed Service for Prometheus 的 IAM 角色，以及所需的权限，现在可以安装 Amazon EMR Flink 运算符。

创建 `enable-amp.yaml` 文件。此文件允许您使用自定义配置来覆盖 Amazon Managed Service for Prometheus 的设置。务必使用您自己的角色。

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

```
        queueConfig:
            maxSamplesPerSend: 1000
            maxShards: 200
            capacity: 2500
```

使用 [Helm Install --set](https://helm.sh/docs/helm/helm_install/) 命令将覆盖传递给 `flink-kubernetes-operator` 图表。

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

此命令会自动在端口 9999 的 Operator 上安装 Prometheus 报告程序。未来的任何 `FlinkDeployment` 也会在 9249 上公开一个 `metrics` 端口。

- Flink Operator 指标会出现在 Prometheus 中的 `flink_k8soperator_` 标签下。
- Flink Task Manager 指标会出现在 Prometheus 的 `flink_taskmanager_` 标签下。
- Flink Job Manager 指标会出现在 Prometheus 的 `flink_jobmanager_` 标签下。

使用 Flink UI 监控 Flink 作业

要监控正在运行的 Flink 应用程序的运行状况和性能，请使用 Flink Web 控制面板。此仪表板提供有关任务状态 TaskManagers、数量以及该任务的指标和日志的信息。借助它还可以查看并修改 Flink 作业配置，以及通过提交或取消作业与 Flink 集群进行交互。

要访问 Kubernetes 上正在运行的 Flink 应用程序的 Flink Web 控制面板，请按照以下步骤操作：

1. 使用`kubectl port-forward`命令将本地端口转发到 Flink 应用程序容器中运行 Flink Web 控制面板的 TaskManager 端口。默认情况下，此端口为 8081。*deployment-name*替换为上面的 Flink 应用程序部署名称。

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

   输出示例：

   ```
   kubectl get deployments -n flink-namespace
   NAME                         READY   UP-TO-DATE   AVAILABLE   AGE
   ```

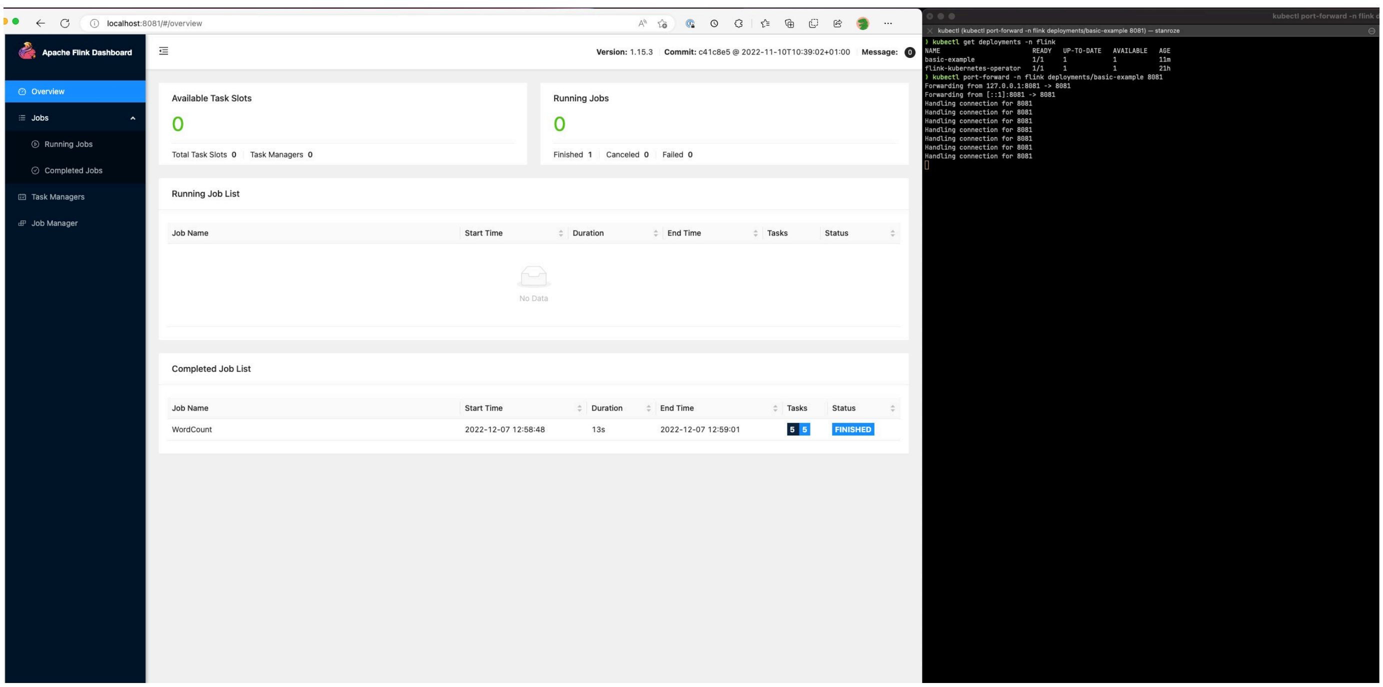
```
basic-example                1/1      1         1         11m
flink-kubernetes-operator    1/1      1         1         21h
```

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

2. 如果要在本地使用其他端口，请使用:8081 *local-port* 参数。

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

3. 在 Web 浏览器中导航到 `http://localhost:8081`（如果使用的是自定义本地端口，则导航到 `http://localhost:`*`local-port`*）来访问 Flink Web 控制面板。此仪表板显示有关正在运行的 Flink 应用程序的信息，例如作业的状态 TaskManagers、数量以及该作业的指标和日志。



使用监控配置监控 Flink Kubernetes Operator 和 Flink 作业

监控配置使您可以轻松地将 Flink 应用程序和操作员日志的日志存档设置为 S3 和/或 CloudWatch（您可以选择其中一个或两个）。这样做会向 JobManager 你的 TaskManager 和容器添加一个 FluentD 边车，然后将这些组件的日志转发到你配置的接收器。

> **Note**
>
> 必须为 Flink Operator 和 Flink 任务（服务账户）设置服务账户的 IAM 角色，才能使用此功能，因为它需要与其他 Amazon Web Services 服务进行交互。您必须按照 设置 Amazon EMR on EKS 的 Flink Kubernetes Operator，使用 IRSA 进行设置。

Flink 应用程序日志

您可以通过下列方式来定义此配置。

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: basic-example
spec:
  image: FLINK IMAGE TAG
  imagePullPolicy: Always
  flinkVersion: v1_17
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
  executionRoleArn: JOB EXECUTION ROLE
  jobManager:
    resource:
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
  job:
    jarURI: local:///opt/flink/examples/streaming/StateMachineExample.jar
  monitoringConfiguration:
    s3MonitoringConfiguration:
      logUri: S3 BUCKET
    cloudWatchMonitoringConfiguration:
      logGroupName: LOG GROUP NAME
      logStreamNamePrefix: LOG GROUP STREAM PREFIX
    sideCarResources:
      limits:
        cpuLimit: 500m
        memoryLimit: 250Mi
```

```
    containerLogRotationConfiguration:
        rotationSize: 2GB
        maxFilesToKeep: 10
```

配置选项如下。

- `s3MonitoringConfiguration` – 用来设置转发到 S3 的配置密钥
  - `logUri`（必需）– 用来存储日志的 S3 存储桶路径。
  - 上传日志后，S3 中的路径如下所示。
    - 未启用日志轮换：

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

    - 已启用日志轮换。您可以同时使用轮换文件和当前文件（不带日期戳的文件）。

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

      以下格式是递增数字。

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

  - 使用此转发器需要以下 IAM 权限。

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

- `cloudWatchMonitoringConfiguration`— 用于设置转发的配置密钥 CloudWatch。
  - `logGroupName`（必填）— 要向其发送 CloudWatch 日志的日志组的名称（如果该组不存在，则自动创建该组）。
  - `logStreamNamePrefix`（可选）– 要向其发送日志的日志流的名称。默认值是空字符串。格式如下所示：

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

- 使用此转发器需要以下 IAM 权限。

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

- `sideCarResources`（可选）– 用于在启动的 Fluentbit Sidecar 容器上设置资源限制的配置密钥。
  - `memoryLimit`（可选）– 默认值为 512Mi。根据自身需求进行调整。
  - `cpuLimit`（可选）– 此选项没有默认值。根据自身需求进行调整。
- `containerLogRotationConfiguration`（可选）– 控制容器日志的轮换行为。该功能默认已启用。
  - `rotationSize`（必需）– 指定日志轮换的文件大小。可行值的范围从 2KB 到 2GB 不等。rotationSize 参数的数字单位部分以整数形式传递。由于不支持十进制值，您可以指定 1.5GB 的轮换大小，例如值 1500MB。默认值为 2GB。
  - `maxFilesToKeep`（必需）– 指定轮换后要在容器中保留的最大文件数。最小值为 1，最大值为 50。默认值为 10。

Flink Operator 日志

我们还可以使用 Helm 图表安装 `values.yaml` 文件中的以下选项，为 Operator 启用日志存档。您可以启用 S3 或两者。 CloudWatch

```
monitoringConfiguration:
  s3MonitoringConfiguration:
    logUri: "S3-BUCKET"
    totalFileSize: "1G"
    uploadTimeout: "1m"
```

```
  cloudWatchMonitoringConfiguration:
    logGroupName: "flink-log-group"
    logStreamNamePrefix: "example-job-prefix-test-2"
  sideCarResources:
    limits:
      cpuLimit: 1
      memoryLimit: 800Mi
  memoryBufferLimit: 700M
```

monitoringConfiguration 下的可用配置选项如下。

- s3MonitoringConfiguration – 设置此选项以存档到 S3。
- logUri（必填项）– 用来存储日志的 S3 存储桶路径。
- 以下是上传日志后 S3 存储桶路径的示例格式。
  - 未启用日志轮换。

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

  - 已启用日志轮换。您可以同时使用轮换文件和当前文件（不带日期戳的文件）。

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

    以下格式索引是递增数字。

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

- cloudWatchMonitoringConfiguration— 用于设置转发的配置密钥 CloudWatch。
  - logGroupName（必填）— 您要向其发送 CloudWatch 日志的日志组的名称。如果日志组不存在，则会自动创建。
  - logStreamNamePrefix（可选）– 要向其发送日志的日志流的名称。默认值是空字符串。中的格式 CloudWatch 如下：

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

- sideCarResources（可选）– 用于在启动的 Fluentbit Sidecar 容器上设置资源限制的配置密钥。
  - memoryLimit（可选）– 内存限制。根据自身需求进行调整。默认值为 512Mi。
  - cpuLimit – CPU 限制。根据自身需求进行调整。无默认值。

- `containerLogRotationConfiguration`（可选）– 控制容器日志的轮换行为。该功能默认已启用。
  - `rotationSize`（必需）– 指定日志轮换的文件大小。可行值的范围从 2KB 到 2GB 不等。rotationSize 参数的数字单位部分以整数形式传递。由于不支持十进制值，您可以指定 1.5GB 的轮换大小，例如值 1500MB。默认值为 2GB。
  - `maxFilesToKeep`（必需）– 指定轮换后要在容器中保留的最大文件数。最小值为 1，最大值为 50。默认值为 10。

Flink 如何支持高可用性和作业弹性

以下部分概述了 Flink 如何使作业更加可靠和高度可用。Flink 通过内置的高可用性功能和各种故障恢复功能来实现这一点。

主题

- 为 Flink Operator 和 Flink 应用程序使用高可用性（HA）
- 使用 EKS 上的 Amazon EMR 优化 Flink 任务重启时间以进行任务恢复和扩展操作
- 使用 EKS 上的 Amazon EMR 上的 Flink 正常停用竞价型实例

为 Flink Operator 和 Flink 应用程序使用高可用性（HA）

本主题介绍了如何配置高可用性以及如何在几种不同的用例中使用。包括在使用 Job Manager 和 Flink Native Kubernetes 时。

Flink Operator 高可用性

我们为 Flink Operator 启用了高可用性，这样就可以使用备用 Flink Operator 进行故障转移，从而在发生故障时最大限度地减少 Operator 控制回路中的停机时间。默认会启用“高可用性”，启动 Operator 副本的默认数量为 2。您可以在 Helm 图表的 `values.yaml` 文件中配置副本字段。

以下字段支持自定义：

- `replicas`（可选，默认值为 2）：将此数字设置为大于 1 会创建其他备用 Operator，从而更快地恢复任务。
- `highAvailabilityEnabled`（可选，默认值为 true）：控制是否要启用 HA。将此参数指定为 true 可启用多可用区部署支持，并设置正确的 `flink-conf.yaml` 参数。

在 `values.yaml` 文件中设置以下配置可以为 Operator 禁用 HA。

```
...
imagePullSecrets: []

replicas: 1

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

多可用区部署

我们在多个可用区中创 Operator Pod。这是一个软约束，如果不同可用区中没有足够的资源，您的 Operator Pod 将被调度到同一可用区中。

确定主副本

如果启用了 HA，则副本使用租约来确定哪个 JMs 是领导者，并使用 K8s Lease 进行领导者选举。您可以描述租约并查看 .Spec.Holder Identity 字段来确定当前主副本

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

Flink-S3 交互

配置访问凭证

请确保已为 IRSA 配置了相应的 IAM 权限来访问 S3 存储桶。

从 S3 应用程序模式获取任务 jar

Flink Operator 也支持从 S3 获取应用程序 jar。您只需在 FlinkDeployment 规范中提供 JarurI 的 S3 位置即可。

您也可以使用此功能下载其他工件，例如 PyFlink 脚本。生成的 Python 脚本放在路径 `/opt/flink/usrlib/` 下。

以下示例演示了如何将此功能用于作 PyFlink 业。注意 jarURI 和 args 字段。

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

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

Flink S3 连接器

Flink 随附两个 S3 连接器（如下所示）。以下各节旨在介绍何时使用哪个连接器。

检查点：Presto S3 连接器

- 将 S3 方案设置为 s3p://
- 用于检查点到 s3 的推荐连接器。有关更多信息，请参阅 Apache 文档中特定于 S3 的内容。

示例 FlinkDeployment 规范：

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: basic-example
```

```
spec:
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
    state.checkpoints.dir: s3p://<BUCKET-NAME>/flink-checkpoint/
```

读取/写入 S3：Hadoop S3 连接器

- 将 S3 方案设置为 `s3://` 或 `s3a://`
- 用于从 S3 读取和写入文件的推荐连接器（仅限实现 Flinks Filesystem 接口的 S3 连接器）。
- 默认在 `flink-conf.yaml` 文件中设置 `fs.s3a.aws.credentials.provider`，即 `com.amazonaws.auth.WebIdentityTokenCredentialsProvider`。如果完全覆盖默认值 `flink-conf`，并且正在与 S3 进行交互，请务必使用此提供程序。

示例 FlinkDeployment 规范

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

Flink Job Manager

Flink Deployments 的高可用性 (HA) 允许任务继续取得进展，即使遇到暂时性错误并 JobManager 导致崩溃。任务将重新启动，但会从上次成功启用 HA 的检查点开始。如果未启用 HA，Kubernetes 将重启你的 JobManager，但你的作业将以全新的作业开始并失去其进度。配置 HA 后，我们可以让 Kubernetes 将 HA 元数据存储在永久存储中，以便在中出现暂时故障时参考， JobManager 然后从上次成功的检查点恢复我们的作业。

Flink 任务会默认启用 HA（副本计数设置为 2，这需要您提供 S3 存储位置来永久存储 HA 元数据）。

HA 配置

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

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

以下是 Job Manager 中上述 HA 配置的描述（在 .spec.jobManager 下定义）：

- `highAvailabilityEnabled`（可选，默认值为 true）：如果不想启用 HA，也不想使用提供的 HA 配置，请将其设置为 `false` 。您仍然可以操作“replicas”字段来手动配置 HA。
- `replicas`（可选，默认值为 2）：将此数字设置为大于 1 会创建其他待机状态 JobManagers ，从而可以更快地恢复作业。如果禁用 HA，则必须将副本计数设置为 1，否则会不断收到验证错误（如果未启用 HA，则仅支持 1 个副本）。
- `storageDir`（必需）：由于默认使用副本计数 2，我们必须提供永久 storageDir。目前，此字段仅接受 S3 路径作为存储位置。

Pod 区域

如果您启用 HA，我们还会尝试将 Pod 放在同一个可用区中，这样可以提高性能（通过将 pod 放在同一个可用区中来减少网络延迟 AZs）。这是一个尽力而为的过程，即如果在调度了大多数 Pod 的可用区中没有足够的资源，那么剩余 Pod 仍会被调度，但最终可能会出现在该可用区之外的节点上。

确定主副本

如果启用了 HA，则副本使用租约来确定哪个 JMs 是领导者，并使用 K8s Configmap 作为数据存储来存储此元数据。如果要确定主副本，可以查看 Configmap 的内容，在数据下查看密钥

`org.apache.flink.k8s.leader.restserver`，找到带 IP 地址的 K8s Pod。您也可以使用以下 bash 命令。

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

Flink 作业 - 本机 Kubernetes

Amazon EMR 6.13.0 及更高版本都支持 Flink 本机 Kubernetes 在 Amazon EKS 集群上以高可用性模式运行 Flink 应用程序。

> **Note**
>
> 提交 Flink 作业时，必须使用一个 Amazon S3 存储桶来存储高可用性元数据。如果不想使用此功能，可以将其禁用。系统会默认启用该功能。

要开启 Flink 高可用性功能，请在运行 `run-application` CLI 命令时提供以下 Flink 参数。参数在示例的下方定义。

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

- **`Dhigh-availability.storageDir`** – 您要存储作业的高可用性元数据的 Amazon S3 存储桶。

  `Dkubernetes.jobmanager.replicas` – 要创建的 Job Manager 容器组（pod）的数量，以大于 1 的整数表示。

  `Dkubernetes.cluster-id` – 标识 Flink 集群的唯一 ID。

使用 EKS 上的 Amazon EMR 优化 Flink 任务重启时间以进行任务恢复和扩展操作

当任务失败或发生扩展操作时，Flink 会尝试从上一次完成的检查点重新执行任务。重启过程可能需要一分钟或更长时间才能执行，具体取决于检查点状态的大小以及并行任务的数量。重启期间，可以累积作业的积压任务。但是，Flink 可以通过一些方法来优化执行图的恢复和重启速度，从而提高作业稳定性。

本页介绍了 Amazon EMR Flink 对竞价型实例执行任务恢复或扩展操作过程中缩短作业重启时间的一些方法。竞价型实例是未使用的计算容量，以折扣价提供。该实例具有独特的行为，偶尔发生中断，因此了解 Amazon EMR on EKS 如何处理这些行为非常重要，包括 Amazon EMR on EKS 如何执行停用和作业重启。

主题

- 任务本地恢复
- 通过 Amazon EBS 卷挂载实现的任务本地恢复
- 基于日志的通用增量检查点
- 精细恢复
- 自适应计划程序中的组合重启机制

任务本地恢复

> **Note**
>
> EKS 上的 Amazon EMR 6.14.0 及更高版本上的 Flink 支持任务本地恢复。

使用 Flink 检查点，每个任务都会生成其状态的快照，Flink 会将该快照写入分布式存储（如 Amazon S3）。在恢复的情况下，任务会从分布式存储中恢复其状态。分布式存储提供容错能力，并且可以在重新扩展期间重新分配状态，因为它可供所有节点访问。

但远程分布式存储也有一个缺点：所有任务都必须通过网络从远程位置读取其状态。在任务恢复或扩展操作期间，这可能会导致大规模状态的恢复时间很长。

通过任务本地恢复可以解决恢复时间长这一问题。任务将其在检查点上的状态写入任务本地的辅助存储（例如本地磁盘）。它们还将状态存储在主存储中，或者存储在 Amazon S3 中（在本例中）。恢复期

间，计划程序将任务计划在任务之前运行所在的同一个任务管理器上，这样它们就可以从本地状态存储中恢复，而不是从远程状态存储中读取。有关更多信息，请参阅 Apache Flink 文档中的任务本地恢复。

我们对示例作业进行的基准测试表明，启用任务本地恢复后，恢复时间已从几分钟缩短到几秒。

要启用任务本地恢复，请在 `flink-conf.yaml` 文件中设置以下配置。指定检查点间隔值，以毫秒为单位。

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

通过 Amazon EBS 卷挂载实现的任务本地恢复

> **Note**
>
> EKS 上的 Amazon EMR 6.15.0 及更高版本上的 Flink 支持 Amazon EBS 的任务本地恢复。

使用 EKS 上的 Amazon EMR 上的 Flink，你可以自动将亚马逊 EBS 卷配置到 TaskManager 用于任务本地恢复的 pod。默认的叠加挂载随附 10GB 的卷，足以满足状态较低的作业。状态较大的作业可以启用自动 EBS 卷挂载选项。这些区域有：TaskManager Pod 是在创建容器器时自动创建和挂载的，在删除容器器时会被移除。

按照以下步骤为 EKS 上的 Amazon EMR 中的 Flink 启用自动 EBS 卷挂载：

1. 导出您以下变量的值，您将在接下来的步骤中使用它们。

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

2. 为集群创建或更新 `kubeconfig` YAML 文件。

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

3. 在 Amazon EKS 集群上为 Amazon EBS Container Storage Interface（CSI）驱动程序创建 IAM 服务账户。

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

4. 使用以下命令创建 Amazon EBS CSI 驱动程序：

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

5. 使用以下命令创建 Amazon EBS 存储类：

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

然后应用该类：

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

6. Helm 使用创建服务账户的选项安装 Amazon EMR Flink Kubernetes Operator。这将创建在 Flink 部署中使用的 `emr-containers-sa-flink`。

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

```
    --set rbac.jobRoleBinding.create=true
```

7. 要提交 Flink 作业并启用 EBS 卷的自动预调配以进行任务本地恢复，请在 `flink-conf.yaml` 文件中设置以下配置。调整作业状态大小的大小限制。将 `serviceAccount` 设置为 `emr-containers-sa-flink`。指定检查点间隔值，以毫秒为单位。并省略 `executionRoleArn`。

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

当您准备删除 Amazon EBS CSI 驱动程序插件时，请使用以下命令：

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

基于日志的通用增量检查点

> **Note**
>
> EKS 上的 Amazon EMR 6.14.0 及更高版本上的 Flink 支持基于日志的通用增量检查点。

Flink 1.16 中添加了基于日志的通用增量检查点功能，以提高检查点的速度。较快的检查点间隔通常会导致恢复工作减少，因为恢复后需要重新处理的事件较少。有关更多信息，请参阅 Apache Flink 博客上的[使用基于日志的通用增量检查点提高检查点的速度和稳定性]()。

对于示例作业，我们的基准测试表明，使用基于日志的通用增量检查点时，检查点时间从几分钟缩短到几秒。

要启用基于日志的通用增量检查点，请在 `flink-conf.yaml` 文件中设置以下配置。指定检查点间隔值，以毫秒为单位。

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

精细恢复

> **Note**
>
> EKS 上的 Amazon EMR 6.14.0 及更高版本上的 Flink 支持对默认计划程序的精细恢复支持。EKS 上的 Amazon EMR 6.15.0 及更高版本上的 Flink 提供自适应计划程序的精细恢复支持。

当任务在执行过程中失败时，Flink 会重置整个执行图，并从上次完成的检查点触发完整的重新执行。这比仅重新执行失败的任务更昂贵。精细恢复仅重新启动失败的任务与管道连接的组件。在以下示例中，作业图有 5 个顶点（A 到 E）。顶点之间的所有连接都使用逐点分布进行管道化处理，作业的 `parallelism.default` 设置为 2。

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

在本示例中，总共有 10 个任务在运行。(a1到e1) 的第一条管道在 TaskManager (TM1)，第二条管道（a2到e2）在另一条管道上运行 TaskManager (TM2).

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

有两个管道连接的组件：a1 # e1 和 a2 # e2。如果其中一个TM1或TM2失败，则故障仅影响管道中的 5 个任务，其中 TaskManager 正在运行。重启策略仅会启动受影响的管道化组件。

精细恢复仅适用于完全并行的 Flink 作业。keyBy() 或 redistribute() 操作不支持。有关更多信息，请参阅 Flink 改进提案 Jira 项目中的 FLIP-1：从任务失败中进行精细恢复。

要启用精细恢复，请在 flink-conf.yaml 文件中设置以下配置。

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

自适应计划程序中的组合重启机制

> **Note**
>
> EKS 上的 Amazon EMR 6.15.0 及更高版本上的 Flink 支持自适应计划程序中的组合重启机制。

自适应计划程序可以根据可用插槽调整作业的并行度。如果没有足够的插槽来适应配置的作业并行度，它将自动降低并行度。如果有新的插槽可用，则任务将再次纵向扩展到配置的作业并行度。当没有足够的可用资源时，自适应计划程序将避免作业停机。这是 Flink Autoscaler 支持的计划程序。出于这些原因，我们建议在 Amazon EMR Flink 中使用自适应计划程序。但是，自适应计划程序可能会在短时间内进行多次重启，每添加一个新资源就会重启一次。这可能导致作业性能下降。

在 Amazon EMR 6.15.0 及更高版本中，Flink 在自适应计划程序中具有组合重启机制，可在添加第一个资源时打开一个重启窗口，然后等到配置的默认 1 分钟窗口间隔时。当有足够的资源可用来运行具有配置并行性的作业时，或者当间隔超时时，它会执行一次重启。

对于示例作业，我们的基准测试表明，当您使用自适应计划程序和 Flink Autoscaler 时，此功能处理的记录比默认行为多 10%。

要启用组合重启机制，请在 flink-conf.yaml 文件中设置以下配置。

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

使用 EKS 上的 Amazon EMR 上的 Flink 正常停用竞价型实例

使用 EKS 上的 Amazon EMR 的 Flink 可以在任务恢复或扩展操作期间缩短作业的重启时间。

概览

EKS 上的 Amazon EMR 发行版 6.15.0 及更高版本支持在带有 Apache Flink 的 EKS 上的 Amazon EMR 中正常停用竞价型实例上的任务管理器。作为此功能的一部分，带有 Flink 的 EKS 上的 Amazon EMR 提供了以下功能：

- Just-in-time check pointing — Flink 流式传输作业可以响应竞价型实例中断，对正在运行的作业执行 just-in-time (JIT) 检查点，并防止在这些竞价型实例上安排其他任务。默认和自适应计划程序支持 JIT 检查点。
- 组合重启机制 – 组合重启机制会在作业达到目标资源并行度或当前配置窗口结束后尽最大努力尝试重新启动作业。这样做还可以防止由于多次竞价型实例终止而导致作业连续重启。组合重启机制仅适用于自适应计划程序。

这些功能具有以下优势：

- 您可以利用竞价型实例来运行任务管理器并降低集群开支。
- 竞价型实例任务管理器的活动性提高使弹性得到提高，作业计划的效率提升。
- 您的 Flink 作业将有更长的正常运行时间，因为竞价型实例终止后的重启次数将会减少。

正常停用的工作原理

考虑以下示例：您预调配了运行 Apache Flink 的 EKS 上的 Amazon EMR，为作业管理器指定了按需节点，为任务管理器指定了竞价型实例节点。终止前两分钟，任务管理器收到中断通知。

在这种情况下，作业管理器将处理竞价型实例中断信号，阻止在竞价型实例上计划其他任务，并为流式传输作业启动 JIT 检查点。

然后，只有在当前重启间隔窗口内有足够的新资源可用来满足当前作业并行度之后，作业管理器才会重新启动任务图。重启窗口间隔是根据竞价型实例替换持续时间、新的任务管理器容器组（pod）的创建以及向作业管理器的注册来决定的。

先决条件

要使用正常停用，请在运行 Apache Flink 的 Amazon EMR on EKS 集群上创建并运行流处理作业。启用在至少一个竞价型实例上计划的自适应计划程序和任务管理器，如以下示例所示。您应该为作业管理器使用按需节点，并且只要至少有一个竞价型实例，您就可以将按需节点用于任务管理器。

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

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

配置

本部分涵盖了您可以根据自己停用需求指定的大多数配置。

键	描述	默认值	可接受值
cluster.taskmanager.graceful-decommission.enabled	启用任务管理器的正常停用。	true	true, false
jobmanager.adaptiv	在自适应计划程序中启用组合重启机制。	false	true, false

键	描述	默认值	可接受值
e-scheduler.combined-restart.enabled			
jobmanager.adaptive-scheduler.combined-restart.window-interval	为作业执行合并重启的合并重启窗口间隔。不含单位的整数将被解释为毫秒。	1m	示例：30、60s、3m、1h。

在 Flink 应用程序中使用 Autoscaler

Operator Autoscaler 可以从 Flink 任务中收集指标，并自动调整任务顶点级别的并行度，从而帮助缓解反向压力。配置可能类似于如下示例：

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  ...
spec:
  ...
  flinkVersion: v1_18
  flinkConfiguration:
    job.autoscaler.enabled: "true"
    job.autoscaler.stabilization.interval: 1m
    job.autoscaler.metrics.window: 5m
    job.autoscaler.target.utilization: "0.6"
    job.autoscaler.target.utilization.boundary: "0.2"
    job.autoscaler.restart.time: 2m
    job.autoscaler.catch-up.duration: 5m
    pipeline.max-parallelism: "720"
  ...
```

此配置使用最新版本的 Amazon EMR 的默认值。如果使用其他版本，值可能会有所不同。

> **Note**
>
> 从 Amazon EMR 7.2.0 开始，无需在配置中包含前缀 `kubernetes.operator`。如果使用 7.1.0 或更低版本，则必须在每次配置前使用前缀。例如，必须指定 `kubernetes.operator.job.autoscaler.scaling.enabled`。

以下是 Autoscaler 的配置选项。

- `job.autoscaler.scaling.enabled`：指定是否允许 Autoscaler 执行顶点缩放。默认值为 `true`。如果禁用此配置，Autoscaler 只会收集指标并评估每个顶点的建议并行度，但不会升级作业。
- `job.autoscaler.stabilization.interval` – 不会执行新扩展的稳定期。默认值为 5 分钟。
- `job.autoscaler.metrics.window` – 扩展指标聚合窗口大小。窗口越大，就越流畅、越稳定，但 Autoscaler 对负载突变做出反应的速度可能会变慢。默认值为 15 分钟。建议使用 3 到 60 分钟之间的值进行实验。
- `job.autoscaler.target.utilization` – 提供稳定任务性能和一定负载波动缓冲能力的目标顶点利用率。默认值为 `0.7`，即任务顶点的目标利用率/负载率为 70%。
- `job.autoscaler.target.utilization.boundary` – 目标顶点利用率边界，用作额外的缓冲，避免在负载波动时立即扩展。默认值为 `0.3`，即在触发缩放操作之前，允许与目标利用率有 30% 的偏差。
- `ob.autoscaler.restart.time` – 重新启动应用程序的预计时间。默认值为 5 分钟。
- `job.autoscaler.catch-up.duration` – 赶上进度的预计时间，即在扩展操作完成后完全处理积压工作的时间。默认值为 5 分钟。通过缩短赶上进度的持续时间，Autoscaler 必须为扩展操作预留更多额外容量。
- `pipeline.max-parallelism` – Autoscaler 可以使用的最大并行度。如果该限值高于 Flink 配置中设定的最大并行度或直接在每个 Operator 上设定的最大并行度，则 Autoscaler 会忽略此限值。默认值为 -1。请注意，Autoscaler 将并行度计算为最大并行度数的除数，因此建议选择具有大量除数的最大并行度设置，而非依赖 Flink 提供的默认设置。建议对此配置使用 60 的倍数，例如 120、180、240、360、720。

有关详细配置的参考页面，请参阅 [Autoscaler configuration](#)。

Autoscaler 参数自动调整

本节介绍了各个 Amazon EMR 版本的自动调整行为。还详细介绍了不同的自动扩缩配置。

> **Note**
>
> Amazon EMR 7.2.0 及更高版本使用开源配置 `job.autoscaler.restart.time-tracking.enabled` 来实现重新缩放时间估计。重新缩放时间估计与 Amazon EMR 自动调整的功能相同，因此无需为重启时间手动分配经验值。
>
> 如果运行的是 Amazon EMR 7.1.0 或更低版本，您仍然可以使用 Amazon EMR 自动调整。

7.2.0 and higher

Amazon EMR 7.2.0 及更高版本测量会应用自动扩缩决策所需的实际重启时间。在 7.1.0 及更低版本中，必须使用配置 `job.autoscaler.restart.time` 来手动配置估计的最长重启时间。通过使用配置 `job.autoscaler.restart.time-tracking.enabled`，您只需输入第一次缩放的重启时间。之后，Operator 会记录实际的重启时间，并将其用于后续缩放。

要启用此跟踪，请使用以下命令：

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

以下是重新缩放时间估计的相关配置。

配置	必需	默认值	描述
job.autoscaler.restart.time-tracking.enabled	否	False	指示 Flink Autoscaler 是否应随着时间的推移自动调整配置，以优化缩放决策。请注意，Autoscaler 只能自动调整 Autoscaler 参数 `restart.time` 。
job.autoscaler.restart.time	否	5m	Amazon EMR on EKS 使用的预期重启时间，直到 Operator 可根据之前的缩放确定实际重启时间。
job.autoscaler.restart.time-tracking.limit	否	15m	当 `job.autoscaler.restart.time-tracking`

配置	必需	默认值	描述
			.enabled 设置为 true 时观察到的最长重启时间。

下面是一个示例部署规范，可用来尝试重新缩放时间估计：

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

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

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

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

  executionRoleArn: <JOB ARN>
  emrReleaseLabel: emr-7.9.0-flink-latest
  jobManager:
    highAvailabilityEnabled: false
    storageDir: s3://<s3_bucket>/flink/autoscaling/ha/
    replicas: 1
    resource:
      memory: "1024m"
      cpu: 0.5
  taskManager:
    resource:
      memory: "1024m"
      cpu: 0.5
```

```
  job:
    jarURI: s3://<s3_bucket>/some-job-with-back-pressure
    parallelism: 1
    upgradeMode: stateless
```

要模拟背压，请使用以下部署规范。

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

将以下 Python 脚本上传到 S3 存储桶。

```
import logging
import sys
import time
import random

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

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

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

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

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

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

要验证重缩时间估计是否有效，请确保已启用 Flink Operator 的 DEBUG 级别日志记录。下面的示例演示了如何更新 Helm 图表文件 `values.yaml`。然后重新安装更新后的 Helm 图表并再次运行 Flink 作业。

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

获取主容器器组（pod）的名称。

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

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

运行以下命令，获取指标评估中使用的实际重启时间。

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

您应该会看到类似下面的日志。请注意，只有第一次缩放才会使用 `job.autoscaler.restart.time`。后续缩放使用观察到的重启时间。

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

7.0.0 and 7.1.0

开源内置 Flink Autoscaler 使用大量指标来做出最佳缩放决策。但在计算时使用的默认值适用于大多数工作负载，对于给定作业来说可能不是最佳值。Amazon EMR on EKS 版本的 Flink Operator 中添加的自动调整功能会查看针对特定捕获指标观察到的历史趋势，然后相应地尝试计算为给定作业定制的最佳值。

配置	必需	默认值	描述
kubernetes.operator.job.autoscaler.autotune.enable	否	False	指示 Flink Autoscaler 是否应随着时间的推移自动调整配置，以优化 Autoscaler 缩放决策。目前，Autoscaler 只能自动调整 Autoscaler 参数 `restart.time` 。
kubernetes.operator.job.autoscaler.autotune.metrics.history.max.count	否	3	指示 Autoscaler 在 Amazon EMR on EKS 指标配置映射中保留的 Amazon EMR on EKS 历史指标数。
kubernetes.operator.job.autoscaler.autotune.metrics.restart.count	否	3	指示 Autoscaler 在开始计算给定作业的平均重启时间之前执行的重启次数。

要启用自动调整，必须完成以下操作：

- 将 `kubernetes.operator.job.autoscaler.autotune.enable:` 设置为 `true`

- 将 `metrics.job.status.enable:` 设置为 `TOTAL_TIME`
- 按照在 Flink 应用程序中使用 Autoscaler 的设置启用自动扩缩功能。

下面是一个示例部署规范，可用来尝试自动调整。

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

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

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

    jobmanager.scheduler: adaptive

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

  executionRoleArn: <JOB ARN>
  emrReleaseLabel: emr-6.14.0-flink-latest
  jobManager:
    highAvailabilityEnabled: true
    storageDir: s3://<S3_bucket>/flink/autoscaling/ha/
    replicas: 1
    resource:
      memory: "1024m"
      cpu: 0.5
  taskManager:
```

```
    resource:
      memory: "1024m"
      cpu: 0.5
  job:
    jarURI: s3://<S3_bucket>/some-job-with-back-pressure
    parallelism: 1
    upgradeMode: last-state
```

要模拟背压，请使用以下部署规范。

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

将以下 Python 脚本上传到 S3 存储桶。

```
import logging
import sys
import time
import random

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

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

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

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

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

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

要验证 Autosutiner 是否正在运行，请使用以下命令。请注意，必须对 Flink Operator 使用您自己的主容器器组（pod）信息。

首先，获取主容器器组（pod）的名称。

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

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

获取主容器器组（pod）的名称后，您可以运行以下命令。

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

您应该会看到类似下面的日志。

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

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

Amazon EMR on EKS 上 Flink 作业的维护和故障排除

以下部分概述了如何维护长时间运行的 Flink 作业，并就如何排除 Flink 作业中的一些常见问题提供了指导。

维护 Flink 应用程序

主题

- 升级模式

Flink 应用程序通常设计为长时间运行，例如运行数周、数月甚至数年。与所有长期运行的服务一样，Flink 流式处理应用程序需要维护。这包括错误修复、改进与迁移到更高版本的 Flink 集群。

当 `FlinkDeployment` 和 `FlinkSessionJob` 资源的规范发生变化时，您需要升级正在运行的应用程序。为此，操作员停止正在运行的作业（除非已暂停），并使用最新的规范重新部署该作业，而对于有状态的应用程序，使用上次运行的状态。

用户可以通过 `JobSpec` 的 `upgradeMode` 设置来控制如何在有状态的应用程序停止和恢复时管理状态。

升级模式

可选介绍

无状态

无状态应用程序将从空白状态升级。

上一状态

在任一应用程序状态（即使是失败的作业）下快速升级都不需要正常运行的作业，因为它总是使用最后一次成功的检查点。如果 HA 元数据丢失，可能需要进行手动恢复。要限制在选取最新检查点时作业可能回退的时间，您可以配置 `kubernetes.operator.job.upgrade.last-state.max.allowed.checkpoint.age`。如果检查点早于配置的值，则对于运行正常的作业，系统将获取其保存点。在会话模式下，不支持此功能。

保存点

> 使用保存点进行升级，以获得最大的安全性，并可用作备份/分叉点。升级过程中，系统将创建保存点。请注意，Flink 作业必须正在运行，才能创建保存点。如果作业处于不健康状态，则将使用最后一个检查点（除非 kubernetes.operator.job.upgrade）。 last-state-fallback.enabled 设置为 false）。如果最后一个检查点不可用，则作业升级将失败。

故障排除

本部分介绍如何对 Amazon EMR on EKS 的问题进行故障排除。有关如何排查一般性 Amazon EMR 问题的信息，请参阅《Amazon EMR 管理指南》中的 集群问题排查。

- 对使用 PersistentVolumeClaims (PVC) 的作业进行故障排除
- 对 Amazon EMR on EKS 垂直自动扩展进行问题排查
- 对 Amazon EMR on EKS Spark 运算符进行故障排除

Amazon EMR on EKS 上的 Apache Flink 问题排查

安装 Helm 图表时未找到资源映射

安装 Helm 图表时可能会遇到以下错误消息。

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

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

ensure CRDs are installed first].
```

要解决此错误，请安装 cert-manager 以启用添加 webhook 组件。你必须安装 cert-manager 到您使用的每个 Amazon EKS 集群。

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

Amazon Web Services 服务 访问被拒绝错误

如果你看到 访问被拒绝 错误，请确认 Helm 图表`values.yaml`文件中的 IAM 角色具有正确的权限。`operatorExecutionRoleArn`此外，请确保 `FlinkDeployment` 规范中 `executionRoleArn` 下的 IAM 角色具有正确的权限。

`FlinkDeployment` 卡滞

如果您的 `FlinkDeployment` 卡在“被囚禁”状态，请按照以下步骤强制删除部署：

1. 编辑部署运行。

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

2. 移除此终结器。

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

3. 删除部署。

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

在选择加入模式下运行 Flink 应用程序时出现了 s3a AWSBad RequestException 问题 Amazon Web Services 区域

如果您以可选方式运行 Flink 应用程序 Amazon Web Services 区域，则可能会看到以下错误：

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

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

```
(Service: S3, Status Code: 400, Request ID: ABCDEFGHIJKLMNOP, Extended Request ID:
 AHl42uDNaTUFOus/5IIVNvSakBcMjMCH7dd37ky0vE6jhABCDEFGHIJKLMNOPQRST==)
```

要修复这些错误，请在 FlinkDeployment 定义文件中使用以下配置。

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

我们还建议您使用 SDKv2 凭证提供商：

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

如果您想使用 SDKv1 凭证提供商，请确保您的 SDK 支持您的选择加入区域。有关更多信息，请参阅aws-sdk-java GitHub 存储库。

如果您在选择加入的区域运行 Flink SQL 语句时收到 S3 AWSBadRequestException，请务必在 Flink 配置规范中设置配置 fs.s3a.endpoint.region: *OPT_IN_AWS_REGION_NAME*。

在 CN 区域运行 Flink 会话作业 AWSBadRequestException 时为 S3A

对于 Amazon EMR 发行版 6.15.0-7.2.0 版本，在中国区域运行 Flink 会话作业时可能会收到以下错误消息。包括中国（北京）和中国（宁夏）：

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

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

```
                    Request ID: ABCDEFGH, Extended Request ID:
ABCDEFGH","additionalMetadata":{}}]}
```

人们已经意识到这个问题。团队正在努力修补这些发行版的 Flink Operator。但在我们完成补丁之前，要修复这个错误，需要下载 Flink Operator Helm 图表，将其解压（提取压缩文件），然后在 Helm 图表中更改配置。

具体步骤如下：

1. 更改为（特别是将目录更改为）Helm 图表的本地文件夹，然后运行以下命令行，拉取 Helm 图表并将其解压。

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

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

2. 进入 Helm 图表文件夹，找到 `templates/flink-operator.yaml` 文件。
3. 在中找到`flink-operator-config` ConfigMap 并添加以下`fs.s3a.endpoint.region`配置`flink-conf.yaml`。例如：

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

4. 安装本地 Helm 图表 并运行作业。

支持将 Amazon EMR on EKS 与 Apache Flink 结合使用的发行版

以下 Amazon EMR on EKS 发行版支持使用 Apache Flink。有关所有可用发行版的信息，请参阅 Amazon EMR on EKS 版本。

发行版标签	Java	Flink	Flink Operator
emr-7.2.0-flink-latest	17	1.18.1	-
emr-7.2.0-flink-k8 s-operator-latest	11	-	1.8.0
emr-7.1.0-flink-latest	17	1.18.1	-
emr-7.1.0-flink-k8 s-operator-latest	11	-	1.6.1
emr-7.0.0-flink-latest	11	1.18.0	-
emr-7.0.0-flink-k8 s-operator-latest	11	-	1.6.1
emr-6.15.0-flink-latest	11	1.17.1	-
emr-6.15.0-flink-k8 s-operator-latest	11	-	1.6.0
emr-6.14.0-flink-latest	11	1.17.1	-
emr-6.14.0-flink-k8 s-operator-latest	11	-	1.6.0
emr-6.13.0-flink-latest	11	1.17.0	-
emr-6.13.0-flink-k8 s-operator-latest	11	-	1.5.0

使用 Amazon EMR on EKS 运行 Spark 作业

作业运行是你在 EKS 上提交给 Amazon EMR 的一个工作单元，例如 Spark jar、 PySpark 脚本或 sparkSQL 查询。本主题概述了如何使用管理任务运行 Amazon CLI、使用 Amazon EMR 控制台查看任务运行以及解决常见任务运行错误。

请注意，你无法在 EKS 上的 Amazon EMR 上运行 IPv6 Spark 作业

> **Note**
>
> 使用 Amazon EMR on EKS 提交任务运行之前，必须完成 设置 Amazon EMR on EKS 中的步骤。

主题

- 使用 StartJobRun 运行 Spark 任务
- 使用 Spark Operator 运行 Spark 任务
- 使用 Spark-submit 运行 Spark 任务
- 将 Apache Livy 与 Amazon EMR on EKS 结合使用
- 管理 Amazon EMR on EKS 任务运行
- 使用任务模板
- 使用 Pod 模板
- 使用作业重试策略
- 使用 Spark 事件日志轮替
- 使用 Spark 容器日志轮换
- 使用垂直自动扩展功能处理 Amazon EMR Spark 任务

使用 `StartJobRun` 运行 Spark 任务

本节包括详细的设置步骤，让您的环境做好运行 Spark 作业的准备，然后提供使用指定参数提交作业运行的 step-by-step说明。

主题

- 设置 Amazon EMR on EKS

- 使用 StartJobRun 提交任务运行
- 使用任务提交者分类
- 使用亚马逊 EMR 容器默认分类

设置 Amazon EMR on EKS

要开始设置 Amazon EMR on EKS，请完成以下任务。如果您已注册了 Amazon Web Services（Amazon）并且一直在使用 Amazon EKS，您几乎已准备好使用 Amazon EMR on EKS 了。跳过任何已经完成的任务。

> **Note**
>
> 您也可以按照 Amazon EMR on EKS 研讨会中的说明来设置所有必要的资源，以便在 EKS 上在 Amazon EMR on EKS 上运行 Spark 任务。该研讨会还通过使用 CloudFormation 模板创建入门所需的资源来实现自动化。有关其他模板和最佳实践，请参阅我们的 EMR 容器最佳实践指南。 GitHub

1. 安装或更新到最新版本的 Amazon CLI
2. 设置 kubectl 和 eksctl
3. 开始使用 Amazon EKS - eksctl
4. 在 EKS 上启用 Amazon EMR 的集群访问权限
5. 为 EKS 集群启用 IAM 角色
6. 授予用户访问 Amazon EMR on EKS 的权限
7. 通过 Amazon EMR 注册 Amazon EKS 集群

启用 Amazon EMR on EKS 的集群访问

下面几节介绍了启用集群访问的几种方法。第一种是使用 Amazon EKS 集群访问管理（CAM），后者展示了如何手动启用集群访问。

使用 EKS 访问条目启用集群访问（推荐）

> **Note**
>
> `aws-auth` ConfigMap 已弃用。管理 Kubernetes 访问权限的推荐方法 APIs 是访问条目。

Amazon EMR 已与 Amazon EKS 集群访问管理（CAM）集成，因此您可以自动配置必要的 AuthN 和 AuthZ 策略，以便在 Amazon EKS 集群的命名空间中运行 Amazon EMR Spark 作业。从 Amazon EKS 集群命名空间创建虚拟集群时，Amazon EMR 会自动配置所有必要的权限，因此您无需在当前工作流中添加任何额外步骤。

> **Note**
>
> 只有新的 Amazon EMR on EKS 虚拟集群支持 Amazon EMR 与 Amazon EKS CAM 集成。您无法迁移现有虚拟集群以使用此集成。

先决条件

- 确保你运行的是 2.15.3 或更高版本的 Amazon CLI
- Amazon EKS 集群必须为 1.23 或更高版本。

设置

要在 Amazon EMR 和 Amazon EKS AccessEntry 的 API 操作之间设置集成，请确保您已完成以下各项：

- 确保将 Amazon EKS 集群的 `authenticationMode` 设置为 `API_AND_CONFIG_MAP`。

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

  如果还未设置，请将 `authenticationMode` 设置为 `API_AND_CONFIG_MAP`。

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

  有关身份验证模式的更多信息，请参阅集群身份验证模式。

- 确保用于运行 CreateVirtualCluster 和 DeleteVirtualCluster API 操作的 IAM 角色也具有以下权限：

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

概念和术语

以下是 Amazon EKS CAM 的相关术语和概念列表。

- 虚拟集群（VC）：在 Amazon EKS 中创建的命名空间的逻辑表示。这是指向 Amazon EKS 集群命名空间的 1:1 链接。您可以使用虚拟集群在指定命名空间内的 Amazon EKS 集群上运行 Amazon EMR 工作负载。
- 命名空间：在单个 EKS 集群中隔离资源组的机制。
- 访问策略：允许 EKS 集群中的 IAM 角色访问和操作的权限。
- 访问条目：使用角色 arn 创建的条目。您可以将访问条目关联到访问策略，以便在 Amazon EKS 集群中分配特定权限。
- EKS 访问条目集成虚拟集群：使用 Amazon EKS 中的访问条目 API 操作创建的虚拟集群。

使用 **aws-auth** 启用集群访问

您必须通过执行以下操作来允许 Amazon EMR on EKS 访问集群中的特定命名空间：创建 Kubernetes 角色、将角色绑定到 Kubernetes 用户以及将 Kubernetes 用户映射为服务关联角色 [AWSServiceRoleForAmazonEMRContainers](#)。当 IAM 身份映射命令与 `emr-containers` 一起作为服务名称时，这些操作在 `eksctl` 中自动执行。您可以使用以下命令轻松地执行这些操作。

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

*`my_eks_cluster`*替换为您的 Amazon EKS 集群的名称，替换*`kubernetes_namespace`*为为运行 Amazon EMR 工作负载而创建的 Kubernetes 命名空间。

> **Important**
>
> 必须使用上一步[设置 kubectl 和 eksctl](#) 下载最新的 eksctl 才能使用此功能。

执行手动步骤以启用 Amazon EMR on EKS 的集群访问

您还可以使用以下手动步骤来启用 Amazon EMR on EKS 的集群访问。

1. 在特定命名空间中创建 Kubernetes 角色

   Amazon EKS 1.22 - 1.29

   对于 Amazon EKS 1.22-1.29，运行以下命令以在特定命名空间中创建 Kubernetes 角色。此角色向 Amazon EMR on EKS 授予必要的 RBAC 权限。

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

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

Amazon EKS 1.21 and below

使用 Amazon EKS 1.21 及更低版本，运行以下命令以在特定命名空间中创建 Kubernetes 角色。此角色向 Amazon EMR on EKS 授予必要的 RBAC 权限。

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

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

2. 创建作用域为命名空间的 Kubernetes 角色绑定

运行以下命令以创建绑定在特定命名空间中的 Kubernetes 角色。此角色绑定将在上一步中创建的角色中定义的权限授予名为 `emr-containers` 的用户。此用户确定 Amazon EMR on EKS 的服务相关角色，因此允许 Amazon EMR on EKS 执行由您创建的角色所定义的操作。

```
namespace=my-namespace
```

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

3. 更新 Kubernetes **aws-auth** 配置映射

   您可以使用以下选项之一将 Amazon EMR on EKS 与服务关联的角色映射到 `emr-containers` 用户，并且该用户在上一步中绑定了 Kubernetes 角色。

   选项 1：使用 **eksctl**

   运行以下 `eksctl` 命令：将 Amazon EMR on EKS 服务相关角色映射到 `emr-containers` 用户。

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

   选项 2：不使用 eksctl

   1. 运行以下命令可在文本编辑器中打开 `aws-auth` 配置映射。

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

> **Note**
>
> 如果您收到错误信息`Error from server (NotFound): configmaps "aws-auth" not found`，请参阅 Amazon EKS [用户指南中添加用户角色](#)中的步骤来应用该股票 ConfigMap。

2. 在 `data` 下，将 Amazon EMR on EKS 服务相关角色详细信息添加到 `ConfigMap` 的 `mapRoles` 部分。如果此部分在文件中尚不存在，请添加它。已更新的 `mapRoles` 部分类似于以下示例。

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

3. 保存文件并退出文本编辑器。

为 EKS 集群启用 IAM 角色

以下主题详细介绍了启用 IAM 角色的选项。

主题

- [选项 1：在 EKS 集群上启用 EKS 容器身份](#)
- [选项 2：在 EKS 集群上为服务账户 (IRSA) 启用 IAM 角色](#)

选项 1：在 EKS 集群上启用 EKS 容器身份

Amazon EKS Pod 身份关联提供了管理应用程序凭证的功能，类似于亚马逊 EC2 实例配置文件向亚马逊实例提供凭证的方式。 EC2 Amazon EKS 容器组身份通过额外的 EKS Auth API，以及在每个节点上运行的代理容器组为您的工作负载提供凭证。

自 emr-7.3.0 发布提交模型以来，EKS 上的 Amazon EMR 开始支持 EKS 容器身份。 StartJobRun

有关 EKS 容器身份的更多信息，请参阅[了解 EKS Pod 身份的工作原理](#)。

为什么 EKS Pod 身份？

作为 EMR 设置的一部分，任务执行角色需要在 IAM 角色和特定命名空间（EMR 虚拟集群）中的服务账户之间建立信任边界。在 IRSA 中，这是通过更新 EMR Job 执行角色的信任策略来实现的。但是，由于 IAM 信任策略长度有 4096 个字符的硬限制，因此在最多十二 (12) 个 EKS 集群中共享单个 Job Execution IAM 角色存在限制。

由于 EMR 对 Pod 身份的支持，IAM 角色和服务账户之间的信任边界现在由 EKS 团队通过 EKS pod 身份的关联 APIs进行管理。

> **Note**
>
> EKS Pod 身份的安全边界仍处于服务帐户级别，而不是 Pod 级别。

Pod 身份注意事项

有关 Pod 身份限制的信息，请参阅 [EKS Pod 身份注意事项](#)。

在 EKS 集群中准备 EKS 容器身份

检查所需权限是否存在于 NodeInstanceRole

节点角色`NodeInstanceRole`需要代理权限才能在 EKS Auth API 中执行`AssumeRoleForPodIdentity`操作。您可以将以下内容添加到 [Amazon EKSWorker NodePolicy](#)（在 Amazon EKS 用户指南中定义），也可以使用自定义策略。

如果您的 EKS 集群是使用高于 0.181.0 的 eksctl 版本创建的，则 Amazon EKSWorkerNodePolicy（包括所需的`AssumeRoleForPodIdentity`权限）将自动附加到该节点角色。如果该权限不存在，请手动向 Amazon 添加以下权限 EKSWorkerNodePolicy ，允许担任 pod 身份的角色。EKS pod 身份代理需要此权限才能检索 pod 的凭证。

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

```
    ]
}
```

创建 EKS pod 身份代理附加组件

使用以下命令创建最新版本的 EKS Pod Identity Agent 附加组件：

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

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

使用以下步骤从 Amazon EKS 控制台创建 EKS Pod Identity Agent 附加组件：

1. 打开亚马逊 EKS 控制台：亚马逊 EKS 控制台。
2. 在左侧导航窗格中，选择集群，然后为您要配置 EKS 容器组身份代理插件的集群选择集群名称。
3. 选择附加组件选项卡。
4. 选择获取更多附加组件。
5. 选择 EKS 容器组身份代理插件框右上角的框，然后选择下一步。
6. 在 “配置选定的插件设置” 页面上，从 “版本” 下拉列表中选择任意版本。
7. （可选）展开可选配置设置以输入其他配置。例如，您可以提供备用容器映像位置和 `ImagePullSecrets`。带有已接受键的 JSON 架构显示在附加组件配置架构中。

   在配置值中输入配置键和值。
8. 选择下一步。
9. 通过 CLI 确认代理 pod 是否正在您的集群上运行。

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

输出示例如下：

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

这将在命名空间 DaemonSet 中设置一个新的`kube-system`命名空间。在每个 EKS 节点上运行的 Amazon EKS Pod 身份代理使用AssumeRoleForPodIdentity操作从 EKS 身份验证 API 中检索临时证书。然后，这些凭证可供您在容器内运行 Amazon SDKs 的凭证使用。

有关更多信息，请查看公共文档中的先决条件：设置 Amazon EKS Pod 身份代理。

创建 Job 执行角色

创建或更新允许 EKS Pod 身份的任务执行角色

要在 EKS 上使用 Amazon EMR 运行工作负载，您需要创建一个 IAM 角色。我们在本文档中将此角色称为任务执行角色。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。

此外，您必须创建一个 IAM 策略来指定任务执行角色的必要权限，然后将此策略附加到该角色以启用 EKS Pod Identity。

例如，您具有以下任务执行角色。有关更多信息，请参阅创建任务执行角色。

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

> **Important**
>
> EKS 上的 Amazon EMR 会根据你的任务执行角色名称自动创建 Kubernetes 服务账户。确保角色名称不要太长，因为如果、和的组合`service_account_name`超过长度限制`cluster_namepod_name`，您的任务可能会失败。

任务执行角色配置 — 确保使用以下 EKS Pod Identity 的信任权限创建任务执行角色。要更新现有的任务执行角色，请将其配置为信任以下 EKS 服务主体作为信任策略中的额外权限。此信任权限可以与现有的 IRSA 信任策略共存。

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

```
        }
    ]
}
EOF
```

用户权限：用户需要`iam:PassRole`权限才能执行 `StartJobRun` API 调用或提交作业。此权限使用户能够将任务执行角色传递给 EKS 上的 EMR。默认情况下，Job 管理员应拥有该权限。

以下是用户所需的权限：

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

要进一步限制用户对特定 EKS 集群的访问权限，请将 AssociatedResourceArn 属性筛选器添加到 IAM 策略中。它将角色的承担限制在经过授权的 EKS 集群中，从而增强了您的资源级安全控制。

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

设置 EKS 容器身份关联

先决条件

确保创建 pod 身份关联的 IAM 身份（例如 EKS 管理员用户）拥有`eks:CreatePodIdentityAssociation`和`iam:PassRole`。

```
    {
        "Effect": "Allow",
        "Action": [
            "eks:CreatePodIdentityAssociation",
```

```
        ],
        "Resource": "* or role-arn"
    },
    {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Action": "iam:PassRole",
                "Resource": "* or role-arn",
                "Condition": {
                    "StringEquals": {
                        "iam:PassedToService": "pods.eks.amazonaws.com"
                    }
                }
        }]
    }
```

为角色和 EMR 服务帐号创建关联

Create EMR role associations through the Amazon CLI

当您向 Kubernetes 命名空间提交任务时，管理员必须在任务执行角色和 EMR 托管服务帐号的身份之间创建关联。请注意，EMR 托管式服务账户将在任务提交时自动创建，作用域为提交任务的命名空间。

使用 Amazon CLI（2.24.0 以上版本），运行以下命令来创建具有 pod 标识的角色关联。

运行以下命令创建具有 pod 标识的角色关联：

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

注意：

- 每个集群最多可以有 1,000 个关联。每个任务执行角色——命名空间映射都需要任务提交者、驱动程序和执行者 pod 有 3 个关联。
- 您只能关联与集群属于同一 Amazon 账户的角色。您可以将其他账户访问权限委派给此账户中的角色，也即您配置的供 EKS 容器组身份使用的角色。有关委派访问权限的教程和`AssumeRole`，请参阅 IAM 教程：使用 IAM 角色跨 Amazon 账户委派访问权限。

Create EMR role associations through Amazon EKS

提交作业时，EMR 会创建具有特定命名模式的服务帐户。要进行手动关联或将此工作流程与 Amazon SDK 集成，请执行以下步骤：

构造服务帐户名称：

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

以下示例为示例 Job 执行角色创建角色关联 JobExecutionRoleIRSAv2。

角色关联示例：

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

CLI 命令示例：

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

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

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

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

完成 EKS Pod 身份所需的所有步骤后，您可以跳过 IRSA 设置的以下步骤：

- [在 EKS 集群上为服务账户 (IRSA) 启用 IAM 角色]()
- [创建任务执行角色]()
- [更新任务执行角色的信任策略]()

您可以直接跳到以下步骤：在 EKS 上[向用户授予访问 Amazon EMR]() 的权限

删除角色关联

每当您删除虚拟集群或任务执行角色并且不想再向其服务帐户授予对 EMR 的访问权限时，都应删除该角色的关联。这是因为 EKS 允许关联不存在的资源（命名空间和服务帐号）。EKS 上的 Amazon EMR 建议在命名空间被删除或角色不再使用时删除关联，以便为其他关联腾出空间。

> **Note**
>
> 如果您不将其删除，则挥之不去的关联可能会影响您的扩展能力，因为 EKS 对您可以创建的关联数量有限制（软限制：每个集群 1000 个关联）。你可以列出给定命名空间中的 pod 身份关联，以检查是否有任何需要清理的挥之不去的关联：

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

使用 Amazon CLI （版本 2.24.0 或更高版本），运行以下 emr-containers 命令以删除 EMR 的角色关联：

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

自动将现有 IRSA 迁移到 Pod 身份

您可以使用 eksctl 工具将现有的 IAM 服务账户角色 (IRSA) 迁移到 pod 身份关联：

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

运行不带`--approve`标志的命令只会输出反映迁移步骤的计划，不会进行实际迁移。

故障排除

我的任务因证书提供者出现 ClassNotFound 异常而失败，或者无法获取凭证提供商。NoClassDefinitionFound

EKS Pod Identity 使用容器凭证提供程序来检索必要的证书。如果您指定了自定义凭证提供程序，请确保其正常运行。或者，请确保您使用的是支持 EKS Pod 身份的正确 Amazon SDK 版本。有关更多信息，请参阅 [Amazon EKS 入门](#)。

Job 失败， eks-pod-identity-agent日志中显示 “由于 [x] 大小限制而无法检索凭证” 错误。

EKS 上的 EMR 根据任务执行角色名称创建 Kubernetes 服务账户。如果角色名称过长，EKS Auth 将无法检索证书，因为`cluster_namepod_name`、和的组合`service_account_name`超过了长度限制。确定哪个组件占用的空间最多，并相应地调整大小。

Job 失败， eks-pod-identity日志中显示 “无法检索凭证 xxx” 错误。

此问题的一个可能原因是 EKS 群集是在私有子网下配置的，而没有 PrivateLink 为该群集进行正确配置。检查您的集群是否位于私有网络中，然后进行配置 Amazon PrivateLink 以解决该问题。有关详细说明，请参阅 [Amazon EKS 入门](#)。

选项 2：在 EKS 集群上为服务账户 (IRSA) 启用 IAM 角色

服务账户的 IAM 角色功能适用于 Amazon EKS 版本 1.14 及更高版本，也适用于在 2019 年 9 月 3 日及之后更新为版本 1.13 及更高版本的 EKS 集群。要使用此功能，您可将现有 EKS 集群更新到版本 1.14 或更高版本。有关更多信息，请参阅[更新 Amazon EKS 集群 Kubernetes 版本](#)。

如果您的集群支持服务账户的 IAM 角色，则它会关联 [OpenID Connect](#) 发布者的 URL。您可以在 Amazon EKS 控制台中查看此 URL，也可以使用以下 Amazon CLI 命令来检索它。

> **Important**
>
> 必须使用最新版本的， Amazon CLI 才能从此命令获得正确的输出。

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

预期的输出如下所示。

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

要为集群中的服务账户使用 IAM 角色，您必须使用 eksctl 或Amazon Web Services Management Console创建 OIDC 身份提供商。

使用 **eksctl** 为集群创建 IAM OIDC 身份提供商

使用以下命令查看您的 `eksctl` 版本。此过程假定您已安装 `eksctl`，且您的 `eksctl` 版本最低为 0.32.0 或更高版本。

```
eksctl version
```

有关安装或升级 eksctl 的更多信息，请参阅安装或升级 eksctl。

使用以下命令为您的集群创建 OIDC 身份提供商。将 *cluster_name* 替换为您自己的值。

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

要使用您的集群创建 IAM OIDC 身份提供商 Amazon Web Services Management Console

从集群的 Amazon EKS 控制台描述中检索 OIDC 颁发者网址，或使用以下 Amazon CLI 命令。

使用以下命令检索 Amazon CLI中 OIDC 发布者的 URL。

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

使用以下步骤检索 Amazon EKS 控制台中 OIDC 发布者的 URL。

1. 使用 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 在导航窗格中，选择 Identity Providers (身份提供商)，然后选择 Create Provider (创建提供商)。
    1. 对于 Provider Type（提供商类型），选择 Choose a provider type（选择提供商类型），然后选择 OpenID Connect。
    2. 对于 Provider URL（提供商 URL），粘贴集群的 OIDC 发布者 URL。
    3. 对于受众，键入 sts.amazonaws.com 并选择 Next Step (下一步)。
3. 验证提供商信息正确，然后选择 Create（创建）以创建身份提供商。

创建任务执行角色

要在 Amazon EMR on EKS 上运行工作负载，您需要创建一个 IAM 角色。我们在本文档中将此角色称为任务执行角色。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。

您必须创建 IAM policy 来指定任务执行角色的权限，然后将 IAM policy 添加到任务执行角色。

以下任务执行角色策略允许访问资源目标、Amazon S3 和 CloudWatch。这些权限是监控任务和访问日志所必需的。要使用执行相同的流程 Amazon CLI，您还可以使用在 EKS 上的 Amazon EMR 研讨会的 “为任务执行创建 IAM 角色” 部分中的步骤来设置您的角色。

> **Note**
>
> 应适当限定访问范围，而不是授予对任务执行角色中所有 S3 对象的权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents",
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:*:*:*"
            ]
        }
    ]
}
```

有关更多信息，请参阅使用任务执行角色、将任务运行配置为使用 S3 日志和将任务运行配置为使用 CloudWatch 日志。

更新任务执行角色的信任策略

当您使用服务账户的 IAM 角色（IRSA）在 Kubernetes 命名空间上运行任务时，管理员必须在任务执行角色和 EMR 托管式服务账户身份之间创建信任关系。通过更新任务执行角色的信任策略来创建信任关系。请注意，EMR 托管式服务账户将在任务提交时自动创建，作用域为提交任务的命名空间。

要更新信任策略，请运行以下命令。

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

有关更多信息，请参阅 将任务执行角色与 Amazon EMR on EKS 结合使用。

> **Important**
>
> 运行上述命令的操作员必须具有以下权限：eks:DescribeCluster、iam:GetRole、iam:UpdateAssumeRolePolicy。

授予用户访问 Amazon EMR on EKS 的权限

对于您在 Amazon EMR on EKS 执行的任何操作，您需要对该操作具有相应的 IAM 权限。您必须创建 IAM policy，以便您执行 Amazon EMR on EKS 操作，并将该策略附加到您使用的 IAM 用户或角色。

本主题提供了创建新策略并将其附加到用户的步骤。IAM policy 还涵盖了在 EKS 环境上设置 Amazon EMR 所需的基本权限。我们建议您尽可能根据您的业务需求来优化特定资源的权限。

在 IAM 控制台中创建新 IAM policy 并将其附加到用户

创建新 IAM policy

1. 登录 Amazon Web Services Management Console 并打开 IAM 控制台，网址为https://console.aws.amazon.com/iam/。
2. 在 IAM 控制台的导航窗格中，选择 Policies (策略)。

3. 在策略页面上，选择 Create a policy (创建策略)。
4. 在 Create Policy (创建策略) 窗口中，导航到 Edit JSON (编辑 JSON) 选项卡。创建包含一个或多个 JSON 语句的策略文档，如该过程以下示例所示。接下来，选择 Review policy (查看策略)。
5. 在 Review Policy (查看策略) 屏幕上，输入您的 Policy Name (策略名称)，例如 `AmazonEMROnEKSPolicy`。输入可选描述，然后选择 Create policy (创建策略)。

将策略附加到用户或角色

1. 登录 Amazon Web Services Management Console 并打开 IAM 控制台，网址为 https://console.aws.amazon.com/iam/
2. 在导航窗格中，选择 Policies（策略）。
3. 在策略列表中，选中在前一部分中所创建策略旁边的复选框。您可以使用 Filter 菜单和搜索框来筛选策略列表。
4. 选择 Policy actions（策略操作），然后选择 Attach（附加）。
5. 选择要将此策略附加到的用户或角色。您可以使用 Filter（筛选条件）菜单和搜索框来筛选委托人实体列表。在选择要将策略附加到的用户或角色后，选择 Attach policy (附加策略)。

管理虚拟集群的权限

要管理您 Amazon 账户中的虚拟集群，请创建具有以下权限的 IAM 策略。这些权限允许您在 Amazon 账户中创建、列出、描述和删除虚拟集群。

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

```
            "Effect": "Allow",
            "Action": [
                "emr-containers:CreateVirtualCluster",
                "emr-containers:ListVirtualClusters",
                "emr-containers:DescribeVirtualCluster",
                "emr-containers:DeleteVirtualCluster"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon EMR 已与 Amazon EKS 集群访问管理（CAM）集成，因此您可以自动配置必要的 AuthN 和 AuthZ 策略，以便在 Amazon EKS 集群的命名空间中运行 Amazon EMR Spark 作业。要完成此操作，您必须具有以下权限：

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

有关更多信息，请参阅自动启用 Amazon EMR on EKS 集群访问。

首次从 Amazon 账户调用该`CreateVirtualCluster`操作时，您还需要拥有在 EKS 上为 Amazon EMR 创建服务相关角色的`CreateServiceLinkedRole`权限。有关更多信息，请参阅 对 Amazon EMR on EKS 使用服务相关角色。

提交任务的权限

要在您 Amazon 账户中的虚拟集群上提交任务，请创建具有以下权限的 IAM 策略。这些权限允许您启动、列出、描述和取消账户中所有虚拟集群的任务运行。您应该考虑添加列出或描述虚拟集群的权限，以便在提交任务之前检查虚拟集群的状态。

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

用于调试和监控的权限

要访问推送到 Amazon S3 的日志 CloudWatch，或者要在 Amazon EMR 控制台中查看应用程序事件日志，请创建具有以下权限的 IAM 策略。我们建议您尽可能根据您的业务需求来优化特定资源的权限。

> **Important**
>
> 如果您尚未创建 Amazon S3 存储桶，您需要添加 `s3:CreateBucket` 权限以访问策略声明。如果您尚未创建日志组，则需要添加 `logs:CreateLogGroup` 到策略声明。

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

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

有关如何配置任务运行以将日志推送到 Amazon S3 的更多信息 CloudWatch，请参阅将任务运行配置为使用 S3 日志和将任务运行配置为使用 CloudWatch 日志。

通过 Amazon EMR 注册 Amazon EKS 集群

注册集群是设置 Amazon EMR on EKS 以运行工作负载所需的最后一步。

使用以下命令创建一个虚拟集群，其名称为您在前面的步骤中为 Amazon EKS 集群和命名空间所选的名称。

> **Note**
>
> 每个虚拟集群都必须拥有一个在所有 EKS 集群中唯一的名称。如果两个虚拟集群共用相同的名称，则即使这两个虚拟集群属于不同的 EKS 集群，部署过程也将失败。

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

或者，您可以创建包含虚拟集群所需参数的 JSON 文件，然后使用 JSON 文件的路径运行 `create-virtual-cluster` 命令。有关更多信息，请参阅 管理虚拟集群。

> **Note**
>
> 要验证虚拟集群是否成功创建，请使用 `list-virtual-clusters` 操作或转到 Amazon EMR 控制台中的 Virtual Clusters (虚拟集群) 页面，以查看虚拟集群状态。

使用 StartJobRun 提交任务运行

使用具有指定参数的 JSON 文件来提交任务运行

1. 创建 `start-job-run-request.json` 文件并指定任务运行所需的参数，如下面的 JSON 文件所示。有关这些参数的更多信息，请参阅配置任务运行的选项。

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

```
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.driver.memory":"2G"
         }
      }
    ],
    "monitoringConfiguration": {
      "persistentAppUI": "ENABLED",
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "my_log_group",
        "logStreamNamePrefix": "log_stream_prefix"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://my_s3_log_location"
      }
    }
  }
}
```

2. 使用 `start-job-run` 命令和存储在本地的 `start-job-run-request.json` 文件路径。

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

使用 **`start-job-run`** 命令来开启任务运行。

1. 使用 `StartJobRun` 命令提供所有指定的参数，如以下示例所示。

```
aws emr-containers start-job-run \
--virtual-cluster-id 123456 \
--name myjob \
--execution-role-arn execution-role-arn \
--release-label emr-6.2.0-latest \
--job-driver '{"sparkSubmitJobDriver": {"entryPoint": "entryPoint_location",
 "entryPointArguments": ["argument1", "argument2", ...], "sparkSubmitParameters":
 "--class <main_class> --conf spark.executor.instances=2 --conf
 spark.executor.memory=2G --conf spark.executor.cores=2 --conf
 spark.driver.cores=1"}}' \
```

```
--configuration-overrides '{"applicationConfiguration": [{"classification":
 "spark-defaults", "properties": {"spark.driver.memory": "2G"}}],
 "monitoringConfiguration": {"cloudWatchMonitoringConfiguration":
 {"logGroupName": "log_group_name", "logStreamNamePrefix": "log_stream_prefix"},
 "persistentAppUI":"ENABLED",  "s3MonitoringConfiguration": {"logUri":
 "s3://my_s3_log_location" }}}'
```

2. 对于 Spark SQL，使用 `StartJobRun` 命令提供所有指定的参数，如以下示例所示。

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

使用任务提交者分类

概览

Amazon EMR on EKS `StartJobRun` 请求会创建一个 job-submitter Pod（也称为 job-runner Pod）来生成 Spark 驱动程序。您可以使用`emr-job-submitter`分类为作业提交者窗格配置节点选择器，也可以为作业提交者窗格的日志容器设置映像、CPU 和内存。

该`emr-job-submitter`分类下有以下设置可用：

`jobsubmitter.node.selector.[`*`labelKey`*`]`

添加到任务提交者 Pod 的节点选择器中，以键 *`labelKey`* 和值作为配置的配置值。例如，您可以将 `jobsubmitter.node.selector.identifier` 设置为 `myIdentifier`，任务提交者 Pod 会有一个键标识符值为 `myIdentifier` 的节点选择器。这可以用来指定任务提交者窗格可以放在哪些节点上。要添加多个节点选择器键，可使用此前缀设置多个配置。

jobsubmitter.logging.image

设置用于作业提交者窗格上的日志容器的自定义镜像。

jobsubmitter.logging.request.cores

为作业提交者窗格上的日志容器的 CPUs数量设置自定义值（以 CPU 为单位）。默认情况下，此值设置为 100 米。

jobsubmitter.logging.request.memory

为作业提交者窗格上的日志容器设置一个自定义的内存量（以字节为单位）。默认情况下，此值设置为 200Mi。兆字节是一种类似于兆字节的度量单位。

我们建议将任务提交者窗格放在按需实例上。如果运行作业提交者容器的实例受到竞价型实例中断，则在竞价型实例上放置作业提交者容器可能会导致任务失败。您也可以将任务提交者 Pod 放在单个可用区中，也可以使用应用于节点的任何 Kubernetes 标签。

任务提交者分类示例

本节内容

- StartJobRun 为任务提交者 Pod 请求按需型节点放置
- StartJobRun 为任务提交者 Pod 请求单可用区型节点放置
- StartJobRun使用单可用区和 Amazon EC2 实例类型放置来请求任务提交者 pod
- StartJobRun使用自定义日志容器镜像、CPU 和内存进行请求

StartJobRun 为任务提交者 Pod 请求按需型节点放置

```
cat >spark-python-in-s3-nodeselector-job-submitter.json << EOF
{
  "name": "spark-python-in-s3-nodeselector",
  "virtualClusterId": "virtual-cluster-id",
  "executionRoleArn": "execution-role-arn",
  "releaseLabel": "emr-6.11.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "s3://S3-prefix/trip-count.py",
       "sparkSubmitParameters": "--conf spark.driver.cores=5  --conf
 spark.executor.memory=20G --conf spark.driver.memory=15G --conf
 spark.executor.cores=6"
```

```
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.dynamicAllocation.enabled":"false"
         }
      },
      {
        "classification": "emr-job-submitter",
        "properties": {
            "jobsubmitter.node.selector.eks.amazonaws.com/capacityType": "ON_DEMAND"
        }
      }
    ],
    "monitoringConfiguration": {
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "/emr-containers/jobs",
        "logStreamNamePrefix": "demo"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://joblogs"
      }
    }
  }
}
EOF
aws emr-containers start-job-run --cli-input-json file:///spark-python-in-s3-
nodeselector-job-submitter.json
```

StartJobRun 为任务提交者 Pod 请求单可用区型节点放置

```
cat >spark-python-in-s3-nodeselector-job-submitter-az.json << EOF
{
  "name": "spark-python-in-s3-nodeselector",
  "virtualClusterId": "virtual-cluster-id",
  "executionRoleArn": "execution-role-arn",
  "releaseLabel": "emr-6.11.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "s3://S3-prefix/trip-count.py",
```

```
      "sparkSubmitParameters": "--conf spark.driver.cores=5  --conf
 spark.executor.memory=20G --conf spark.driver.memory=15G --conf
 spark.executor.cores=6"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.dynamicAllocation.enabled":"false"
         }
      },
      {
        "classification": "emr-job-submitter",
        "properties": {
            "jobsubmitter.node.selector.topology.kubernetes.io/zone": "Availability
 Zone"
        }
      }
    ],
    "monitoringConfiguration": {
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "/emr-containers/jobs",
        "logStreamNamePrefix": "demo"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://joblogs"
      }
    }
  }
}
EOF
aws emr-containers start-job-run --cli-input-json file:///spark-python-in-s3-
nodeselector-job-submitter-az.json
```

StartJobRun使用单可用区和 Amazon EC2 实例类型放置来请求任务提交者 pod

```
{
  "name": "spark-python-in-s3-nodeselector",
  "virtualClusterId": "virtual-cluster-id",
  "executionRoleArn": "execution-role-arn",
  "releaseLabel": "emr-6.11.0-latest",
```

```
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "s3://S3-prefix/trip-count.py",
       "sparkSubmitParameters": "--conf spark.driver.cores=5  --conf
 spark.kubernetes.pyspark.pythonVersion=3 --conf spark.executor.memory=20G
 --conf spark.driver.memory=15G --conf spark.executor.cores=6 --conf
 spark.sql.shuffle.partitions=1000"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.dynamicAllocation.enabled":"false",
         }
      },
      {
        "classification": "emr-job-submitter",
        "properties": {
            "jobsubmitter.node.selector.topology.kubernetes.io/zone": "Availability
 Zone",
            "jobsubmitter.node.selector.node.kubernetes.io/instance-type":"m5.4xlarge"
        }
      }
    ],
    "monitoringConfiguration": {
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "/emr-containers/jobs",
        "logStreamNamePrefix": "demo"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://joblogs"
      }
    }
  }
}
```

`StartJobRun`使用自定义日志容器镜像、CPU 和内存进行请求

```
{
  "name": "spark-python",
  "virtualClusterId": "virtual-cluster-id",
```

```
  "executionRoleArn": "execution-role-arn",
  "releaseLabel": "emr-6.11.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "s3://S3-prefix/trip-count.py"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "emr-job-submitter",
        "properties": {
            "jobsubmitter.logging.image": "YOUR_ECR_IMAGE_URL",
            "jobsubmitter.logging.request.memory": "200Mi",
            "jobsubmitter.logging.request.cores": "0.5"
        }
      }
    ],
    "monitoringConfiguration": {
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "/emr-containers/jobs",
        "logStreamNamePrefix": "demo"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://joblogs"
      }
    }
  }
}
```

使用亚马逊 EMR 容器默认分类

概览

该emr-containers-defaults分类下有以下设置可用：

job-start-timeout

默认情况下，如果作业无法启动，则会超时，并在该SUBMITTED状态下等待 15 分钟。此配置更改了在任务超时之前等待的秒数。

logging.image

设置用于驱动程序和执行器容器上的日志容器的自定义镜像。

logging.request.cores

为驱动程序和执行器容器上的日志容器的 CPUs数量设置自定义值（以 CPU 为单位）。默认情况下，未设置此选项。

logging.request.memory

为驱动程序和执行器容器上的日志容器设置一个自定义的内存量（以字节为单位）。默认情况下，此值设置为 512Mi。兆字节是一种类似于兆字节的度量单位。

任务提交者分类示例

本节内容

- StartJobRun带有自定义任务超时的请求
- StartJobRun使用自定义日志容器镜像、CPU 和内存进行请求

StartJobRun带有自定义任务超时的请求

```
{
  "name": "spark-python",
  "virtualClusterId": "virtual-cluster-id",
  "executionRoleArn": "execution-role-arn",
  "releaseLabel": "emr-6.11.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "s3://S3-prefix/trip-count.py"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "emr-containers-defaults",
        "properties": {
            "job-start-timeout": "1800"
        }
      }
    ],
    "monitoringConfiguration": {
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "/emr-containers/jobs",
        "logStreamNamePrefix": "demo"
```

```
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://joblogs"
      }
    }
  }
}
```

StartJobRun使用自定义日志容器镜像、CPU 和内存进行请求

```
{
  "name": "spark-python",
  "virtualClusterId": "virtual-cluster-id",
  "executionRoleArn": "execution-role-arn",
  "releaseLabel": "emr-6.11.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "s3://S3-prefix/trip-count.py"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "emr-containers-defaults",
        "properties": {
            "logging.image": "YOUR_ECR_IMAGE_URL",
            "logging.request.memory": "200Mi",
            "logging.request.cores": "0.5"
        }
      }
    ],
    "monitoringConfiguration": {
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "/emr-containers/jobs",
        "logStreamNamePrefix": "demo"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://joblogs"
      }
    }
  }
}
```

使用 Spark Operator 运行 Spark 任务

Amazon EMR 6.10.0 及更高版本都支持 Apache Spark 版 Kubernetes Operator（又称 Spark Operator），作为 Amazon EMR on EKS 的任务提交模型。使用 Spark Operator，您可以在自己的 Amazon EKS 集群上使用 Amazon EMR 发行版运行时系统部署并管理 Spark 应用程序。在 Amazon EKS 集群中部署 Spark Operator 后，您可以直接使用 Operator 提交 Spark 应用程序。Operator 会管理 Spark 应用程序的生命周期。

> **Note**
>
> Amazon EMR on EMR 会根据 vCPU 和内存消耗计算 Amazon EKS 的定价。该计算适用于驱动程序和执行程序容器组（pod）。该计算从您下载 Amazon EMR 应用程序映像开始，直至 Amazon EKS 容器组（pod）终止，并四舍五入到最接近的秒。

主题

- 设置 Amazon EMR on EKS 的 Spark Operator
- Amazon EMR on EKS 的 Spark Operator 入门
- 对 Amazon EMR on EKS 的 Spark Operator 使用垂直自动扩缩功能
- 卸载 Amazon EMR on EKS 的 Spark Operator
- 使用监控配置监控 Spark Kubernetes Operator 和 Spark 作业
- Amazon EMR on EKS 上的安全要求和 Spark Operator

设置 Amazon EMR on EKS 的 Spark Operator

在 Amazon EKS上安装 Spark Operator 之前，请执行下述任务来完成设置。如果已注册 Amazon Web Services（Amazon）并且一直在使用 Amazon EKS，您基本上就准备好使用 Amazon EMR on EKS 了。完成以下任务，在 Amazon EKS 上设置好 Spark Operator。跳过已完成的先决条件，转到下一个先决条件。

- 安装或更新到最新版本的 Amazon CLI — 如果您已经安装了 Amazon CLI，请确认您安装的是最新版本。
- 设置 kubectl 和 eksctl：eksctl 是用来与 Amazon EKS 通信的命令行工具。
- Install Helm – Kubernetes 的 Helm 包管理器可帮助您在 Kubernetes 集群上安装和管理应用程序。

- 开始使用 Amazon EKS - eksctl：按照以下步骤在 Amazon EKS 中创建带有节点的新 Kubernetes 集群。
- 选择 Amazon EMR 基础映像 URI（6.10.0 或更高版本）– Amazon EMR 6.10.0 及更高版本都支持 Spark Operator。

Amazon EMR on EKS 的 Spark Operator 入门

本主题旨在通过部署 Spark 应用程序和 Schedule Spark 应用程序，帮助您开始在 Amazon EKS 上使用 Spark Operator。

安装 Spark Operator

按照以下步骤安装 Apache Spark 版 Kubernetes Operator。

1. 如果尚未执行此操作，请完成 设置 Amazon EMR on EKS 的 Spark Operator 中的步骤。
2. 向 Amazon ECR 注册表验证您的 Helm 客户端。在以下命令中，将这些*region-id*值替换为您的首选*ECR-registry-account*值 Amazon Web Services 区域，以及按区域划分的 Amazon ECR 注册账户页面中相应区域的值。

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

3. 使用以下命令安装 Spark Operator。

   对于 Helm 图表 `--version` 参数，请使用删除了 `emr-` 前缀和日期后缀的 Amazon EMR 发行版标签。例如，若为 `emr-6.12.0-java17-latest` 发行版，则指定 `6.12.0-java17`。以下命令中的示例使用 `emr-7.8.0-latest` 发行版，因此为 Helm 图表 `--version` 指定了 `7.8.0`。

```
helm install spark-operator-demo \
  oci://895885662937.dkr.ecr.region-id.amazonaws.com/spark-operator \
  --set emrContainers.awsRegion=region-id \
  --version 7.8.0 \
  --namespace spark-operator \
  --create-namespace
```

默认情况下，该命令会为 Spark Operator 创建服务账户 emr-containers-sa-spark-operator。若要使用其他服务账户，请提供参数 serviceAccounts.sparkoperator.name。例如：

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

要在 Spark Operator 中使用垂直自动扩展功能，请在安装命令中添加以下行，从而允许 Operator 的 Webhook：

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

4. 使用 `helm list` 命令验证是否安装了 Helm 图表：

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

`helm list` 命令应返回最新部署的 Helm 图表的版本信息：

```
app_version: v1beta2-1.3.8-3.1.1
chart: spark-operator-7.8.0
name: spark-operator-demo
namespace: spark-operator
revision: "1"
status: deployed
updated: 2023-03-14 18:20:02.721638196 +0000 UTC
```

5. 使用所需的任何其他选项完成安装。有关更多信息，请参阅中的`spark-on-k8s-operator`文档。 GitHub

运行 Spark 应用程序

Amazon EMR 6.10.0 或更高版本都支持 Spark Operator。安装 Spark Operator 时，默认会创建服务账户 emr-containers-sa-spark 来运行 Spark 应用程序。按照下述步骤在 Amazon EMR on EKS 6.10.0 或更高版本上使用 Spark Operator 来运行 Spark 应用程序。

1. 在使用 Spark Operator 运行 Spark 应用程序之前，请先完成 设置 Amazon EMR on EKS 的 Spark Operator 和 安装 Spark Operator 中的步骤。
2. 使用以下示例内容创建 SparkApplication 定义文件 spark-pi.yaml：

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: spark-pi
  namespace: spark-operator
spec:
  type: Scala
  mode: cluster
  image: "895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.10.0:latest"
  imagePullPolicy: Always
  mainClass: org.apache.spark.examples.SparkPi
  mainApplicationFile: "local:///usr/lib/spark/examples/jars/spark-examples.jar"
  sparkVersion: "3.3.1"
  restartPolicy:
    type: Never
  volumes:
    - name: "test-volume"
      hostPath:
        path: "/tmp"
        type: Directory
  driver:
    cores: 1
    coreLimit: "1200m"
    memory: "512m"
    labels:
      version: 3.3.1
    serviceAccount: emr-containers-sa-spark
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
  executor:
    cores: 1
    instances: 1
    memory: "512m"
    labels:
      version: 3.3.1
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
```

3. 现在，使用以下命令提交 Spark 应用程序。此操作还会创建一个名为 `spark-pi` 的 `SparkApplication` 对象：

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

4. 使用以下命令检查 SparkApplication 对象的事件：

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

有关通过 Spark O perator 向 Spark 提交应用程序的更多信息，请在spark-on-k8s-operator文档SparkApplication中的 Using a GitHub。

使用 Amazon S3 进行存储

要将 Amazon S3 用作文件存储选项，请在 YAML 文件中添加以下配置。

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

```
 spark.executor.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/lib/
native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/native
```

如果您使用的是 Amazon EMR 7.2.0 及更高版本，则默认包含这些配置。在这种情况下，可以在 Spark 应用程序 YAML 文件中将文件路径设置为 `s3://<bucket_name>/<file_path>`，而不是 `local://<file_path>`。

然后正常提交 Spark 应用程序。

对 Amazon EMR on EKS 的 Spark Operator 使用垂直自动扩缩功能

从 Amazon EMR 7.0 开始，您可以使用 Amazon EMR on EKS 垂直自动扩缩功能以简化资源管理。该功能会自动调整内存和 CPU 资源，从而适应您为 Amazon EMR Spark 应用程序提供的工作负载需求。有关更多信息，请参阅 使用垂直自动扩展功能处理 Amazon EMR Spark 任务。

本节将介绍如何将 Spark Operator 配置为使用垂直自动扩展功能。

先决条件

在配置监视之前，请确保完成以下设置任务：

- 完成设置 Amazon EMR on EKS 的 Spark Operator中的步骤。
- （可选）如果之前安装的是旧版 Spark Operator，请删除 SparkApplication/ScheduledSparkApplication CRD。

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

- 完成安装 Spark Operator中的步骤。在第 3 步中，在安装命令中添加以下行，以允许运算符的 Webhook：

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

- 完成设置 Amazon EMR on EKS 的垂直自动扩展中的步骤。
- 允许访问 Amazon S3 位置中的文件：

  1. 使用具有 S3 权限的 `JobExecutionRole` 为驱动程序和 Operator 服务账户添加注释。

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

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

2. 更新该命名空间中作业执行角色的信任策略。

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

3. 编辑作业执行角色的 IAM 角色信任策略，并将 `serviceaccount` 从 `emr-containers-sa-spark-*-*-xxxx` 更新为 `emr-containers-sa-*`。

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

4. 如果使用 Amazon S3 作为文件存储空间，请在 yaml 文件中添加以下默认值。

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

 mapreduce.fileoutputcommitter.algorithm.version.emr_internal_use_only.EmrFileSystem:
 "2"
  mapreduce.fileoutputcommitter.cleanup-
failures.ignored.emr_internal_use_only.EmrFileSystem: "true"
sparkConf:
 # Required for EMR Runtime
```

```
 spark.driver.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/hadoop-aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/share/aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/hadoop/extrajars/*
 spark.driver.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/native
 spark.executor.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/hadoop-aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/share/aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/hadoop/extrajars/*
 spark.executor.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/native
```

在 Spark Operator 上使用垂直自动扩展功能运行作业

您必须首先完成 先决条件 中的步骤，然后才能使用 Spark Operator 运行 Spark 应用程序。

要在 Spark Operator 中使用垂直自动扩缩功能，请在 Spark 应用程序规范的驱动程序中添加以下配置，以打开垂直自动扩缩功能：

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

此配置支持垂直自动扩缩，也是必需的签名配置，允许您为作业选择签名。

有关配置和参数值的更多信息，请参阅配置 Amazon EMR on EKS 的自动扩缩。默认情况下，任务在垂直自动扩展的仅限监控关闭模式下提交。这种监控状态可让您在不执行自动扩展的情况下计算并查看资源建议。有关更多信息，请参阅垂直自动扩缩模式。

下面是一个名为 `spark-pi.yaml` 的 `SparkApplication` 定义文件示例，其中包含使用垂直自动扩缩功能所需的配置。

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
```

```
kind: SparkApplication
metadata:
  name: spark-pi
  namespace: spark-operator
spec:
  type: Scala
  mode: cluster
  image: "895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-7.8.0:latest"
  imagePullPolicy: Always
  mainClass: org.apache.spark.examples.SparkPi
  mainApplicationFile: "local:///usr/lib/spark/examples/jars/spark-examples.jar"
  sparkVersion: "3.4.1"
  dynamicSizing:
    mode: Off
    signature: "my-signature"
  restartPolicy:
    type: Never
  volumes:
    - name: "test-volume"
      hostPath:
        path: "/tmp"
        type: Directory
  driver:
    cores: 1
    coreLimit: "1200m"
    memory: "512m"
    labels:
      version: 3.4.1
    serviceAccount: emr-containers-sa-spark
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
  executor:
    cores: 1
    instances: 1
    memory: "512m"
    labels:
      version: 3.4.1
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
```

现在，使用以下命令提交 Spark 应用程序。此操作还会创建一个名为 spark-pi 的 SparkApplication 对象：

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

有关通过 Spark O perator 向 Spark 提交应用程序的更多信息，请在spark-on-k8s-operator文档SparkApplication中的 Using a GitHub。

验证垂直自动扩展功能

要验证已提交任务的垂直自动扩展功能是否正常工作，请使用 kubectl 获取 verticalpodautoscaler 自定义资源并查看扩展建议。

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

此查询的输出应类似以下内容：

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

如果输出内容与此不相似或包含错误代码，请参阅 对 Amazon EMR on EKS 垂直自动扩展进行问题排查，了解有助于解决问题的操作步骤。

要删除容器组（pod）和应用程序，请运行以下命令：

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

卸载 Amazon EMR on EKS 的 Spark Operator

按照以下步骤卸载 Spark Operator。

1. 使用正确的命名空间删除 Spark Operator。在本示例中，命名空间为 spark-operator-demo。

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

2. 删除 Spark Operator 服务账户：

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

3. 删除 Spark 运算符 `CustomResourceDefinitions` (CRDs)：

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

使用监控配置监控 Spark Kubernetes Operator 和 Spark 作业

监视配置支持您轻松地将 Spark 应用程序和 Operator 日志的日志存档设置为 Amazon S3 或 Amazon S3 Amazon CloudWatch。您可以选择其中一项，也可以都选择。这样可以将 log Agent Sidecar 添加到 Spark Operator Pod、驱动程序和 execator Pod，然后将这些组件的日志转发到配置好的接收器。

先决条件

在配置监视之前，请确保完成以下设置任务：

1. （可选）如果之前安装的是旧版 Spark Operator，请删除 SparkApplication/ScheduledSparkApplicationCRD。

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

2. 如果您还没有操作员/任务执行角色，请在 IAM 中创建该角色。
3. 要更新您刚刚创建的Operator /任务执行角色的信任策略，请运行以下命令：

```
aws emr-containers update-role-trust-policy \
--cluster-name cluster \
--namespace namespace \
--role-name iam_role_name_for_operator/job_execution_role
```

4. 将您的操作员/任务执行角色的 IAM 角色信任策略编辑为以下内容：

```
{
    "Effect": "Allow",
    "Principal": {
        "Federated": "${OIDC-provider}"
    },
    "Action": "sts:AssumeRoleWithWebIdentity",
```

```
    "Condition": {
        "StringLike": {
            "OIDC_PROVIDER:sub": "system:serviceaccount:${Namespace}:emr-
containers-sa-*"
        }
    }
}
```

5. 在 IAM 中创建具有以下权限的监控配置策略：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams",
                "logs:CreateLogStream",
                "logs:CreateLogGroup",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:region:account_id:log-group:log_group_name",
                "arn:aws:logs:region:account_id:log-group:log_group_name:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "logs:DescribeLogGroups",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::bucket_name",
                "arn:aws:s3:::bucket_name/*"
            ]
        }
    ]
```

```
}
```

6. 将上述政策附加到您的操作员/任务执行角色中。

SparkOr 日志

您可以通过以下方式来定义监视配置`helm install`：

```
helm install spark-operator spark-operator \
--namespace namespace \
--set emrContainers.awsRegion=aws_region \
--set emrContainers.monitoringConfiguration.image=log_agent_image_url \
--set
 emrContainers.monitoringConfiguration.s3MonitoringConfiguration.logUri=S3_bucket_uri \
--set
 emrContainers.monitoringConfiguration.cloudWatchMonitoringConfiguration.logGroupName=log_group
 \
--set
 emrContainers.monitoringConfiguration.cloudWatchMonitoringConfiguration.logStreamNamePrefix=lo
 \
--set emrContainers.monitoringConfiguration.sideCarResources.limits.cpuLimit=500m \
--set emrContainers.monitoringConfiguration.sideCarResources.limits.memoryLimit=512Mi \
--set
 emrContainers.monitoringConfiguration.containerLogRotationConfiguration.rotationSize=2GB
 \
--set
 emrContainers.monitoringConfiguration.containerLogRotationConfiguration.maxFilesToKeep=10
 \
--set webhook.enable=true \
--set emrContainers.operatorExecutionRoleArn=operator_execution_role_arn
```

监控配置

m onitorin gConfiguration 下的可用配置选项如下。

- 图片（可选）-日志代理图像网址。 emrReleaseLabel 如果未提供，将通过获取。
- s3 MonitoringConfiguration — 设置此选项以存档到 Amazon S3。
  - logUri —（必需）— 用来存储日志的 Amazon S3 存储桶路径。
  - 以下是上传日志后 Amazon S3 存储桶路径的示例格式。第一个示例显示未启用日志轮换。

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

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

默认日志轮换处于启用状态。您既可以看到带有递增索引的旋转文件，也可以看到与前一个示例相同的当前文件。

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

- cloudWatchMonitoring配置-用于设置转发的配置密钥 Amazon CloudWatch。
  - logGroupName（必需）— 要向其发送 Amazon CloudWatch 日志的日志组的名称。如果日志组不存在，则会自动创建。
  - logStreamName前缀（可选）— 要向其发送日志的日志流的名称。默认值是空字符串。中的格式 Amazon CloudWatch 如下：

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

- sideCarResources（可选）— 用于在启动的 Fluentd Sidecar 容器上设置资源限制的配置密钥。
  - memoryLimit（可选）— 内存限制。根据自身需求进行调整。默认值为 512Mi。
  - CPULimit（可选）— CPU 限制。根据自身需求进行调整。默认值为 500 米。
- containerLogRotation配置（可选）— 控制容器日志的轮换行为。该功能默认已启用。
  - rotationSiz e（必需）— 指定日志轮换的文件大小。可行值的范围从 2KB 到 2GB 不等。rotationSize 参数的数字单位部分以整数形式传递。由于不支持十进制值，您可以指定 1.5GB 的轮换大小，例如值 1500MB。默认值为 2GB。
  - maxFilesToKee@@ p（必需）— 指定轮换后要在容器中保留的最大文件数。最小值为 1，最大值为 50。默认值为 10。

配置 MonitoringC onfiguration 后，你应该能够在 Amazon S3 存储桶 Amazon CloudWatch 或两者上查看 spark 操作员容器日志。对于 Amazon S3 存储桶，您需要等待 2 分钟才能刷新第一个日志文件。

要查找日志 Amazon CloudWatch，您可以导航到以下位置：CloudWatch> 日志组 > > /operator *Pod name*/st **Log group name**derr

或者你可以导航到：CloudWatch> 日志组 > > /operator *Pod name*/st **Log group name**dout

Spark 应用程序日志

您可以通过下列方式来定义此配置。

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: spark-pi
  namespace: namespace
spec:
  type: Scala
  mode: cluster
  imagePullPolicy: Always
  mainClass: org.apache.spark.examples.SparkPi
  mainApplicationFile: "local:///usr/lib/spark/examples/jars/spark-examples.jar"
  sparkVersion: "3.3.1"
  emrReleaseLabel: emr_release_label
  executionRoleArn: job_execution_role_arn
  restartPolicy:
    type: Never
  volumes:
    - name: "test-volume"
      hostPath:
        path: "/tmp"
        type: Directory
  driver:
    cores: 1
    coreLimit: "1200m"
    memory: "512m"
    labels:
      version: 3.3.1
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
  executor:
    cores: 1
    instances: 1
    memory: "512m"
    labels:
      version: 3.3.1
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
  monitoringConfiguration:
    image: "log_agent_image"
    s3MonitoringConfiguration:
      logUri: "S3_bucket_uri"
```

```
    cloudWatchMonitoringConfiguration:
      logGroupName: "log_group_name"
      logStreamNamePrefix: "log_stream_prefix"
    sideCarResources:
      limits:
        cpuLimit: "500m"
        memoryLimit: "250Mi"
    containerLogRotationConfiguration:
      rotationSize: "2GB"
      maxFilesToKeep: "10"
```

m onitorin gConfiguration 下的可用配置选项如下。

- 图片（可选）-日志代理图像网址。 emrReleaseLabel 如果未提供，将通过获取。
- s3 MonitoringConfiguration — 设置此选项以存档到 Amazon S3。
  - logUri（必需）— 用来存储日志的 Amazon S3 存储桶路径。第一个示例显示未启用日志轮换：

    ```
    s3://${logUri}/${APPLICATION NAME}-${APPLICATION UID}/${POD NAME}/stdout.gz
    s3://${logUri}/${APPLICATION NAME}-${APPLICATION UID}/${POD NAME}/stderr.gz
    ```

    默认值为启用日志轮换。您可以同时使用轮换文件（带有递增索引的）和当前文件（不带日期戳的文件）。

    ```
    s3://${logUri}/${APPLICATION NAME}-${APPLICATION UID}/${POD NAME}/
    stdout_YYYYMMDD_index.gz
    s3://${logUri}/${APPLICATION NAME}-${APPLICATION UID}/${POD NAME}/
    stderr_YYYYMMDD_index.gz
    ```

- cloudWatchMonitoring配置-用于设置转发的配置密钥 Amazon CloudWatch。
  - logGroupName（必需）— 要向其发送日志的 CloudWatch 日志组的名称。如果日志不存在，则会自动创建。
  - logStreamNamePrefix（可选）— 要向其发送日志的日志流的名称。默认值是空字符串。中的格式 CloudWatch 如下：

    ```
    ${logStreamNamePrefix}/${APPLICATION NAME}-${APPLICATION UID}/${POD NAME}/stdout
    ${logStreamNamePrefix}/${APPLICATION NAME}-${APPLICATION UID}/${POD NAME}/stderr
    ```

- sideCarResources（可选）— 用于在启动的 Fluentd Sidecar 容器上设置资源限制的配置密钥。
  - memoryLimit（可选）— 内存限制。根据自身需求进行调整。默认值为 250Mi。

  - CPULimit — CPU 限制。根据自身需求进行调整。默认值为 500 米。
- containerLogRotation配置（可选）— 控制容器日志的轮换行为。该功能默认已启用。
  - rotationSiz e（必需）— 指定日志轮换的文件大小。可行值的范围从 2KB 到 2GB 不等。rotationSize 参数的数字单位部分以整数形式传递。由于不支持十进制值，您可以指定 1.5GB 的轮换大小，例如值 1500MB。默认值为 2GB。
  - maxFilesToKee@@ p（必需）— 指定轮换后要在容器中保留的最大文件数。最小值为 1。最大值为 50。默认值为 10。

配置 MonitoringConfiguration 后，您应该能够在 Amazon S3 存储桶或 CloudWatch 两者上查看您的 spark 应用程序驱动程序和执行器日志。对于 Amazon S3 存储桶，您需要等待 2 分钟才能刷新第一个日志文件。例如，在 Amazon S3 中，存储桶路径如下所示：

亚马逊 S3 > 存储桶 > ***Bucket name***> ***Spark application name - UUID*** > ***Pod Name*** > stderr.gz

或：

亚马逊 S3 > 存储桶 > ***Bucket name***> ***Spark application name - UUID*** > ***Pod Name*** > stdout.gz

在中 CloudWatch，路径如下所示：

CloudWatch> 日志组 > ***Log group name***> *Pod name*/***Spark application name - UUID***/stderr

或：

CloudWatch> 日志组 > ***Log group name***> *Pod name*/***Spark application name - UUID***/stdout

Amazon EMR on EKS 上的安全要求和 Spark Operator

使用 Spark Operator 时，有几种方法可以设置集群访问权限。第一种是使用基于角色的访问控制（RBAC），该方法根据个人在组织中的角色来限制访问。这是处理访问权限的主要方式。第二种访问方法是代入 Amazon Identity and Access Management 角色，通过分配特定权限来访问资源。

主题

- 使用基于角色的访问控制（RBAC）设置集群访问权限
- 使用服务账户的 IAM 角色（IRSA）设置集群访问权限

使用基于角色的访问控制（RBAC）设置集群访问权限

为了部署 Spark Operator，Amazon EMR on EKS 会为 Spark Operator 和 Spark 应用程序创建两个角色和服务账户。

主题

- Operator 服务账户和角色
- Spark 服务账户和角色

Operator 服务账户和角色

Amazon EMR on EKS 会创建 Operator 服务账户和角色来管理 Spark 任务的 SparkApplications 以及其他资源（例如服务）。

此服务账户的默认名称为 emr-containers-sa-spark-operator。

以下规则适用于此服务角色：

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

```
    - create
    - get
    - delete
  - apiGroups:
    - ""
    resources:
    - nodes
    verbs:
    - get
  - apiGroups:
    - ""
    resources:
    - events
    verbs:
    - create
    - update
    - patch
  - apiGroups:
    - ""
    resources:
    - resourcequotas
    verbs:
    - get
    - list
    - watch
  - apiGroups:
    - apiextensions.k8s.io
    resources:
    - customresourcedefinitions
    verbs:
    - create
    - get
    - update
    - delete
  - apiGroups:
    - admissionregistration.k8s.io
    resources:
    - mutatingwebhookconfigurations
    - validatingwebhookconfigurations
    verbs:
    - create
    - get
    - update
    - delete
```

```
- apiGroups:
  - sparkoperator.k8s.io
  resources:
  - sparkapplications
  - sparkapplications/status
  - scheduledsparkapplications
  - scheduledsparkapplications/status
  verbs:
  - "*"
  {{- if .Values.batchScheduler.enable }}
  # required for the `volcano` batch scheduler
- apiGroups:
  - scheduling.incubator.k8s.io
  - scheduling.sigs.dev
  - scheduling.volcano.sh
  resources:
  - podgroups
  verbs:
  - "*"
  {{- end }}
  {{ if .Values.webhook.enable }}
- apiGroups:
  - batch
  resources:
  - jobs
  verbs:
  - delete
  {{- end }}
```

Spark 服务账户和角色

Spark 驱动程序 Pod 需要一个与该 Pod 位于同一命名空间的 Kubernetes 服务账户。此服务账户需要权限才能创建、获取、列出、修补和删除执行程序 Pod，以及为驱动程序创建 Kubernetes 无头服务。除非 Pod 命名空间中的默认服务账户具有所需权限，否则驱动程序会失败并在没有服务账户的情况下退出。

此服务账户的默认名称为 `emr-containers-sa-spark`。

以下规则适用于此服务角色：

```
 rules:
- apiGroups:
  - ""
```

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

使用服务账户的 IAM 角色（IRSA）设置集群访问权限

本节使用示例演示如何配置 Kubernetes 服务账户来代入角色。 Amazon Identity and Access Management 然后，使用服务账户的 Pod 可以访问该角色有权访问的任何 Amazon 服务。

以下示例运行一个 Spark 应用程序来统计 Amazon S3 中某个文件的字数。为此，您可以设置服务账户的 IAM 角色（IRSA），对 Kubernetes 服务账户进行身份验证和授权。

> **Note**
>
> 此示例将“spark-operator”命名空间用于 Spark Operator 和提交 Spark 应用程序的命名空间。

先决条件

在尝试本页的示例之前，请先完成下述先决条件：

- 设置好 Spark Operator。
- 安装 Spark Operator.

- 创建 Amazon S3 存储桶。
- 将您最喜欢的诗歌保存在名为 poem.txt 的文本文件中，然后将该文件上传到 S3 存储桶。您在此页面上创建的 Spark 应用程序会读取该文本文件中的内容。有关如何将文件上传到 S3 的信息，请参阅《Amazon Simple Storage Service 用户指南》中的将对象上传到存储桶。

配置 Kubernetes 服务账户来代入 IAM 角色

按照以下步骤配置 Kubernetes 服务账户来代入 IAM 角色，Pod 可以使用该角色访问该角色有权访问的 Amazon 服务。

1. 完成后先决条件，使用创建文件，该example-policy.json文件允许 Amazon Command Line Interface 对您上传到 Amazon S3 的文件进行只读访问：

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

2. 然后，创建 IAM policy example-policy：

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

3. 接下来，创建一个 IAM 角色 example-role，将该角色与 Spark 驱动程序的 Kubernetes 服务账户关联起来：

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

4. 创建一个 yaml 文件，其中包含 Spark 驱动程序服务账户所需的集群角色绑定：

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

5. 应用集群角色绑定配置：

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

kubectl 命令应确认成功创建账户：

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

通过 Spark Operator 运行应用程序

配置 Kubernetes 服务账户后，您可以运行 Spark 应用程序来统计作为 先决条件 的一部分上传的文本文件中的字数。

1. 创建一个新文件 `word-count.yaml`，其中包含字数统计应用程序的 SparkApplication 定义。

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

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

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

2. 提交 Spark 应用程序。

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

kubectl 命令应返回您已成功创建名为 `word-count` 的 `SparkApplication` 对象的确认信息。

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

3. 运行以下命令检查 `SparkApplication` 对象的事件：

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

kubectl 命令应返回 `SparkApplication` 的描述和事件：

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

应用程序现在正在统计 S3 文件中的字数。要查找统计的字数，请参阅驱动程序的日志文件：

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

kubectl 命令应返回日志文件的内容以及字数统计应用程序的结果。

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

有关如何通过 Spark Operator 向 Spark 提交应用程序的更多信息，请在 Kubernetes Operator Apache Spark上参阅 Kubernetes Operator (spark-on-k8s-spark) 文档 SparkApplication中的 Using a。 GitHub

使用 Spark-submit 运行 Spark 任务

Amazon EMR 6.10.0 及更高版本都支持 `spark-submit` 作为命令行工具，而您可以使用该工具向 Amazon EMR on EKS 集群提交和执行 Flink 应用程序。

> **Note**
>
> Amazon EMR on EMR 会根据 vCPU 和内存消耗计算 Amazon EKS 的定价。该计算适用于驱动程序和执行程序容器组（pod）。该计算从您下载 Amazon EMR 应用程序映像开始，直至 Amazon EKS 容器组（pod）终止，并四舍五入到最接近的秒。

主题

- 设置 Amazon EMR on EKS 的 spark-submit
- Amazon EMR on EKS 的 spark-submit 入门
- 确认 spark-submit 的 Spark 驱动程序服务账户安全要求

设置 Amazon EMR on EKS 的 spark-submit

先执行以下任务来完成设置，才能在 Amazon EMR on EKS 上使用 spark-submit 运行应用程序。如果已注册 Amazon Web Services（Amazon）并且一直在使用 Amazon EKS，您基本上就准备好使用 Amazon EMR on EKS 了。跳过已完成的先决条件，转到下一个先决条件。

- 安装或更新到最新版本的 Amazon CLI — 如果您已经安装了 Amazon CLI，请确认您安装的是最新版本。
- 设置 kubectl 和 eksctl：eksctl 是用来与 Amazon EKS 通信的命令行工具。
- 开始使用 Amazon EKS - eksctl：按照以下步骤在 Amazon EKS 中创建带有节点的新 Kubernetes 集群。
- 选择 Amazon EMR 基础映像 URI（6.10.0 或更高版本）– Amazon EMR 6.10.0 及更高版本都支持 `spark-submit` 命令。
- 确认驱动程序服务账户有创建并监视执行程序 Pod 的相应权限。有关更多信息，请参阅 确认 spark-submit 的 Spark 驱动程序服务账户安全要求。
- 设置本地 Amazon 凭证配置文件。
- 在 Amazon EKS 控制台中，选择您的 EKS 集群，然后在概述、详细信息、API 服务器端点下找到 EKS 集群端点。

Amazon EMR on EKS 的 spark-submit 入门

Amazon EMR 6.10.0 及更高版本都支持 spark-submit 在 Amazon EKS 集群上运行 Spark 应用程序。以下部分展示了如何提交 Spark 应用程序命令。

运行 Spark 应用程序

要运行 Spark 应用程序，请按照下述步骤操作：

1. 在使用 `spark-submit` 命令运行 Spark 应用程序之前，请先完成[设置 Amazon EMR on EKS 的 spark-submit]中的步骤。
2. 使用 Amazon EMR on EKS 基础映像运行容器。有关更多信息，请参阅[如何选择基础映像 URI]。

```
kubectl run -it containerName --image=EMRonEKSImage --command -n namespace /bin/
bash
```

3. 设置以下环境变量的值：

```
export SPARK_HOME=spark-home
export MASTER_URL=k8s://Amazon EKS-cluster-endpoint
```

4. 现在，使用以下命令提交 Spark 应用程序：

```
$SPARK_HOME/bin/spark-submit \
 --class org.apache.spark.examples.SparkPi \
 --master $MASTER_URL \
 --conf spark.kubernetes.container.image=895885662937.dkr.ecr.us-
west-2.amazonaws.com/spark/emr-6.10.0:latest \
 --conf spark.kubernetes.authenticate.driver.serviceAccountName=spark \
 --deploy-mode cluster \
 --conf spark.kubernetes.namespace=spark-operator \
 local:///usr/lib/spark/examples/jars/spark-examples.jar 20
```

有关向 Spark 提交应用程序的更多信息，请参阅 Apache Spark 文档中的 [Submitting applications]。

> **Important**
>
> `spark-submit` 仅支持集群模式作为提交机制。

确认 spark-submit 的 Spark 驱动程序服务账户安全要求

Spark 驱动程序 Pod 使用 Kubernetes 服务账户访问 Kubernetes API 服务器来创建并监视执行程序 Pod。驱动程序服务账户必须具有相应权限，才能在集群中列出、创建、编辑、修补和删除 Pod。您可以通过运行以下命令验证自己是否可以列出这些资源：

```
kubectl auth can-i list|create|edit|delete|patch pods
```

运行每个命令，确认您是否拥有必要的权限。

```
kubectl auth can-i list pods
kubectl auth can-i create pods
kubectl auth can-i edit pods
kubectl auth can-i delete pods
kubectl auth can-i patch pods
```

以下规则适用于此服务角色：

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

```
  verbs:
  - "*"
```

设置 spark-submit 的服务账户（IRSA）的 IAM 角色

以下部分介绍了如何设置服务账户的 IAM 角色（IRSA），对 Kubernetes 服务账户进行身份验证和授权，从而运行存储在 Amazon S3 中的 Spark 应用程序。

先决条件

在尝试本文档中的任何示例之前，请确保已满足以下先决条件：

- 完成设置 spark-submit
- 已创建 S3 存储桶并上传 Spark 应用程序 jar 文件

配置 Kubernetes 服务账户以代入 IAM 角色

以下步骤介绍如何配置 Kubernetes 服务账户以代入 Amazon Identity and Access Management（IAM）角色。将这些 pod 配置为使用服务帐号后，他们就可以访问 Amazon Web Services 服务 该角色有权访问的任何内容。

1. 创建一个策略文件，允许对上传的 Amazon S3 对象进行只读访问：

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

2. 创建 IAM 策略。

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

3. 创建一个 IAM 角色，将该角色与 Spark 驱动程序的 Kubernetes 服务账户关联：

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

4. 创建一个 YAML 文件，其中包含 Spark 驱动程序服务账户所需的权限：

```
cat >spark-rbac.yaml <<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: emr-containers-role-spark
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - services
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - configmaps
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - persistentvolumeclaims
```

```
  verbs:
  - "*"
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: spark-role-binding
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: emr-containers-role-spark
subjects:
- kind: ServiceAccount
  name: emr-containers-sa-spark
  namespace: default
EOF
```

5. 应用集群角色绑定配置。

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

6. `kubectl` 命令将返回已创建账户的确认信息。

```
serviceaccount/emr-containers-sa-spark created
clusterrolebinding.rbac.authorization.k8s.io/emr-containers-role-spark configured
```

运行 Spark 应用程序

Amazon EMR 6.10.0 及更高版本都支持 spark-submit 在 Amazon EKS 集群上运行 Spark 应用程序。要运行 Spark 应用程序，请按照下述步骤操作：

1. 确保已完成设置 Amazon EMR on EKS 的 spark-submit 中的步骤。
2. 设置以下环境变量的值：

```
export SPARK_HOME=spark-home
export MASTER_URL=k8s://Amazon EKS-cluster-endpoint
```

3. 现在，使用以下命令提交 Spark 应用程序：

```
$SPARK_HOME/bin/spark-submit \
```

```
 --class org.apache.spark.examples.SparkPi \
 --master $MASTER_URL \
 --conf spark.kubernetes.container.image=895885662937.dkr.ecr.us-
west-2.amazonaws.com/spark/emr-6.15.0:latest \
 --conf spark.kubernetes.authenticate.driver.serviceAccountName=emr-containers-sa-
spark \
 --deploy-mode cluster \
 --conf spark.kubernetes.namespace=default \
 --conf "spark.driver.extraClassPath=/usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/
hadoop-aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/
share/aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/
security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-
glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-
serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/
hadoop/extrajars/*" \
 --conf "spark.driver.extraLibraryPath=/usr/lib/hadoop/lib/native:/usr/lib/hadoop-
lzo/lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/
native" \
 --conf "spark.executor.extraClassPath=/usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/
hadoop-aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/
share/aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/
security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-
glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-
serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/
hadoop/extrajars/*" \
 --conf "spark.executor.extraLibraryPath=/usr/lib/hadoop/lib/native:/usr/lib/
hadoop-lzo/lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/
lib/native" \
 --conf
 spark.hadoop.fs.s3.customAWSCredentialsProvider=com.amazonaws.auth.WebIdentityTokenCredent
 \
 --conf spark.hadoop.fs.s3.impl=com.amazon.ws.emr.hadoop.fs.EmrFileSystem \
 --conf
 spark.hadoop.fs.AbstractFileSystem.s3.impl=org.apache.hadoop.fs.s3.EMRFSDelegate \
 --conf spark.hadoop.fs.s3.buffer.dir=/mnt/s3 \
 --conf spark.hadoop.fs.s3.getObject.initialSocketTimeoutMilliseconds="2000" \
 --conf
 spark.hadoop.mapreduce.fileoutputcommitter.algorithm.version.emr_internal_use_only.EmrFile
 \
 --conf spark.hadoop.mapreduce.fileoutputcommitter.cleanup-
failures.ignored.emr_internal_use_only.EmrFileSystem="true" \
 s3://my-pod-bucket/spark-examples.jar 20
```

4. Spark 驱动程序完成 Spark 作业后，您应该会在提交结束时看到一个日志行，指示 Spark 作业已完成。

```
23/11/24 17:02:14 INFO LoggingPodStatusWatcherImpl: Application
 org.apache.spark.examples.SparkPi with submission ID default:org-apache-spark-
examples-sparkpi-4980808c03ff3115-driver finished
23/11/24 17:02:14 INFO ShutdownHookManager: Shutdown hook called
```

清理

运行完应用程序后，可使用以下命令执行清理。

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

将 Apache Livy 与 Amazon EMR on EKS 结合使用

对于 Amazon EMR 7.1.0 及更高版本，可使用 Apache Livy 在 Amazon EMR on EKS 上提交作业。通过 Apache Livy，您可以设置自己的 Apache Livy REST 端点，在 Amazon EKS 集群上部署和管理 Spark 应用程序。在 Amazon EKS 集群中安装 Livy 后，就可以使用 Livy 端点将 Spark 应用程序提交到 Livy 服务器。服务器会管理 Spark 应用程序的生命周期。

> **Note**
>
> Amazon EMR on EMR 会根据 vCPU 和内存消耗计算 Amazon EKS 的定价。该计算适用于驱动程序和执行程序容器组（pod）。该计算从您下载 Amazon EMR 应用程序映像开始，直至 Amazon EKS 容器组（pod）终止，并四舍五入到最接近的秒。

主题

- 设置适用于 Amazon EMR on EKS 的 Apache Livy
- 开始使用适用于 Amazon EMR on EKS 的 Apache Livy
- 使用适用于 Amazon EMR on EKS 的 Apache Livy 运行 Spark 应用程序
- 使用 Amazon EMR on EKS 卸载 Apache Livy
- 使用 Amazon EMR on EKS 配置 Apache Livy 的安全性
- Amazon EMR on EKS 发行版上的 Apache Livy 安装属性
- 排查常见的环境变量格式错误

设置适用于 Amazon EMR on EKS 的 Apache Livy

在 Amazon EKS 集群上安装 Apache Livy 之前，必须先安装并配置一组必备工具。包括用于处理 Amazon 资源的基本命令行工具 Amazon CLI、用于使用 Amazon EKS 的命令行工具以及在本用例中使集群应用程序可用于互联网并路由网络流量的控制器。

- 安装或更新到最新版本的 Amazon CLI — 如果您已经安装了 Amazon CLI，请确认您安装的是最新版本。
- 设置 kubectl 和 eksctl：eksctl 是用来与 Amazon EKS 通信的命令行工具。
- Install Helm – Kubernetes 的 Helm 包管理器可帮助您在 Kubernetes 集群上安装和管理应用程序。
- 开始使用 Amazon EKS - eksctl：按照以下步骤在 Amazon EKS 中创建带有节点的新 Kubernetes 集群。
- 选择 Amazon EMR 发行版标签：Amazon EMR 7.1.0 及更高版本支持 Apache Livy。
- 安装 ALB 控制器：ALB 控制器负责管理 Kubernetes 集群的 Amazon Elastic Load Balancing。当您在设置 Apache Livy 过程中创建 Kubernetes Ingress 时，将会创建 Amazon 网络负载均衡器（NLB）。

开始使用适用于 Amazon EMR on EKS 的 Apache Livy

完成以下步骤安装 Apache Livy。包括配置软件包管理器、为运行 Spark 工作负载创建命名空间、安装 Livy、设置负载均衡和验证步骤。要使用 Spark 运行批处理作业，必须完成以下步骤。

1. 设置适用于 Amazon EMR on EKS 的 Apache Livy。
2. 向 Amazon ECR 注册表验证您的 Helm 客户端。您可以 Amazon Web Services 区域 从按区域划分的 Amazon ECR 注册表账户中找到的相应 ECR-account `ECR-registry-account` 值。

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

3. 设置 Livy 会为 Livy 服务器创建一个服务账户，为 Spark 应用程序创建另一个账户。要为服务账户设置 IRSA，请参阅使用服务账户的 IAM 角色（IRSA）设置访问权限。
4. 创建一个命名空间来运行 Spark 工作负载。

```
kubectl create ns <spark-ns>
```

5. 使用以下命令安装 Livy。

此 Livy 端点仅在 EKS 集群的 VPC 内可用。要启用 VPC 之外的访问，请在 Helm 安装命令中设置 `—-set loadbalancer.internal=false`。

> **Note**
>
> 默认情况下，该 Livy 端点未启用 SSL，且该端点仅在 EKS 集群的 VPC 内可见。如果您设置了 `loadbalancer.internal=false` 和 `ssl.enabled=false`，就会将不安全的端点在 VPC 外部公开。要设置安全的 Livy 端点，请参阅[使用 TLS/SSL 配置安全的 Apache Livy 端点](#)。

```
helm install livy-demo \
  oci://895885662937.dkr.ecr.region-id.amazonaws.com/livy \
  --version 7.9.0 \
  --namespace livy-ns \
  --set image=ECR-registry-account.dkr.ecr.region-id.amazonaws.com/livy/
emr-7.9.0:latest \
  --set sparkNamespace=<spark-ns> \
  --create-namespace
```

您应当看到如下输出。

```
NAME: livy-demo
LAST DEPLOYED: Mon Mar 18 09:23:23 2024
NAMESPACE: livy-ns
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The Livy server has been installed.
Check installation status:
1. Check Livy Server pod is running
  kubectl --namespace livy-ns get pods -l "app.kubernetes.io/instance=livy-demo"
2. Verify created NLB is in Active state and it's target groups are healthy (if
 loadbalancer.enabled is true)

Access LIVY APIs:
    # Ensure your NLB is active and healthy
    # Get the Livy endpoint using command:
```

```
    LIVY_ENDPOINT=$(kubectl get svc -n livy-ns -l app.kubernetes.io/
instance=livy-demo,emr-containers.amazonaws.com/type=loadbalancer -o
 jsonpath='{.items[0].status.loadBalancer.ingress[0].hostname}' |  awk '{printf
 "%s:8998\n", $0}')
    # Access Livy APIs using http://$LIVY_ENDPOINT or https://$LIVY_ENDPOINT (if
 SSL is enabled)
    # Note: While uninstalling Livy, makes sure the ingress and NLB are deleted
 after running the helm command to avoid dangling resources
```

Livy 服务器和 Spark 会话的默认服务账户名称为 `emr-containers-sa-livy` 和 `emr-containers-sa-spark-livy`。要使用自定义名称，请使用 `serviceAccounts.name` 和 `sparkServiceAccount.name` 参数。

```
--set serviceAccounts.name=my-service-account-for-livy
--set sparkServiceAccount.name=my-service-account-for-spark
```

6. 验证是否已安装 Helm 图表。

```
helm list -n livy-ns -o yaml
```

`helm list` 命令应返回有关新 Helm 图表的信息。

```
app_version: 0.7.1-incubating
chart: livy-emr-7.9.0
name: livy-demo
namespace: livy-ns
revision: "1"
status: deployed
updated: 2024-02-08 22:39:53.539243 -0800 PST
```

7. 验证网络负载均衡器是否处于活动状态。

```
LIVY_NAMESPACE=<livy-ns>
LIVY_APP_NAME=<livy-app-name>
AWS_REGION=<AWS_REGION>

# Get the NLB Endpoint URL
NLB_ENDPOINT=$(kubectl --namespace $LIVY_NAMESPACE get svc -l "app.kubernetes.io/
instance=$LIVY_APP_NAME,emr-containers.amazonaws.com/type=loadbalancer" -o
 jsonpath='{.items[0].status.loadBalancer.ingress[0].hostname}')
```

```
# Get all the load balancers in the account's region
ELB_LIST=$(aws elbv2 describe-load-balancers --region $AWS_REGION)

# Get the status of the NLB that matching the endpoint from the Kubernetes service
NLB_STATUS=$(echo $ELB_LIST | grep -A 8 "\"DNSName\": \"$NLB_ENDPOINT\"" | awk '/
Code/{print $2}/}/' | tr -d '"},\n')
echo $NLB_STATUS
```

8. 现在，验证网络负载均衡器中的目标组是否运行正常。

```
LIVY_NAMESPACE=<livy-ns>
LIVY_APP_NAME=<livy-app-name>
AWS_REGION=<AWS_REGION>

# Get the NLB endpoint
NLB_ENDPOINT=$(kubectl --namespace $LIVY_NAMESPACE get svc -l "app.kubernetes.io/
instance=$LIVY_APP_NAME,emr-containers.amazonaws.com/type=loadbalancer" -o
 jsonpath='{.items[0].status.loadBalancer.ingress[0].hostname}')

# Get all the load balancers in the account's region
ELB_LIST=$(aws elbv2 describe-load-balancers --region $AWS_REGION)

# Get the NLB ARN from the NLB endpoint
NLB_ARN=$(echo $ELB_LIST | grep -B 1 "\"DNSName\": \"$NLB_ENDPOINT\"" | awk
 '/"LoadBalancerArn":/,/"/'| awk '/:/{print $2}' | tr -d \",)

# Get the target group from the NLB. Livy setup only deploys 1 target group
TARGET_GROUP_ARN=$(aws elbv2 describe-target-groups --load-balancer-arn $NLB_ARN
 --region $AWS_REGION | awk '/"TargetGroupArn":/,/"/'| awk '/:/{print $2}' | tr -d
 \",)

# Get health of target group
aws elbv2 describe-target-health --target-group-arn $TARGET_GROUP_ARN
```

以下是显示目标组状态的示例输出：

```
{
    "TargetHealthDescriptions": [
        {
            "Target": {
                "Id": "<target IP>",
                "Port": 8998,
                "AvailabilityZone": "us-west-2d"
```

```
            },
            "HealthCheckPort": "8998",
            "TargetHealth": {
                "State": "healthy"
            }
        }
    ]
}
```

一旦 NLB 的状态变为 `active`，且目标群体变为 `healthy`，就可以继续。该过程可能需要几分钟的时间。

9. 从 Helm 安装中检索 Livy 端点。Livy 端点是否安全取决于是否启用了 SSL。

```
LIVY_NAMESPACE=<livy-ns>
 LIVY_APP_NAME=livy-app-name
 LIVY_ENDPOINT=$(kubectl get svc -n livy-ns -l app.kubernetes.io/
instance=livy-app-name,emr-containers.amazonaws.com/type=loadbalancer -o
 jsonpath='{.items[0].status.loadBalancer.ingress[0].hostname}' |  awk '{printf
 "%s:8998\n", $0}')
 echo "$LIVY_ENDPOINT"
```

10. 从 Helm 安装中检索 Spark 服务账户

```
SPARK_NAMESPACE=spark-ns
LIVY_APP_NAME=<livy-app-name>
SPARK_SERVICE_ACCOUNT=$(kubectl --namespace $SPARK_NAMESPACE
 get sa -l "app.kubernetes.io/instance=$LIVY_APP_NAME" -o
 jsonpath='{.items[0].metadata.name}')
echo "$SPARK_SERVICE_ACCOUNT"
```

您应该可以看到类似于如下输出的内容：

```
emr-containers-sa-spark-livy
```

11. 如果`internalALB=true`将此参数设置为允许从 VPC 外部访问，请创建一个 Amazon EC2 EC2 实例，并确保网络负载均衡器允许来自 EC2 EC2 实例的网络流量。必须这样做，实例才能访问 Livy 端点。有关在 VPC 外部安全公开端点的更多信息，请参阅使用 TLS/SSL 设置安全的 Apache Livy 端点。

12. 安装 Livy 会创建服务账户 `emr-containers-sa-spark` 来运行 Spark 应用程序。如果 Spark 服务账户和 Spark 应用程序的驱动程序和执行程序无权访问，则必须将具有必要权限的 IAM 角色设置为变量。 Amazon Amazon 有关更多信息，请参阅使用服务账户的 IAM 角色（IRSA）设置访问权限。

Apache Livy 支持其他配置，您可以在安装 Livy 时使用这些配置。有关更多信息，请参阅 Amazon EMR on EKS 发行版上的 Apache Livy 安装属性。

使用适用于 Amazon EMR on EKS 的 Apache Livy 运行 Spark 应用程序

在使用 Apache Livy 运行 Spark 应用程序之前，请确保已完成设置适用于 Amazon EMR on EKS 的 Apache Livy 和开始使用适用于 Amazon EMR on EKS 的 Apache Livy 中的步骤。

您可以使用 Apache Livy 运行两种类型的应用程序：

- 批处理会话：一种 Livy 工作负载，用于提交 Spark 批处理作业。
- 交互式会话：一种 Livy 工作负载，提供了运行 Spark 查询的编程和可视化界面。

> **Note**
>
> 来自不同会话的驱动程序和执行程序容器组（pod）可以相互通信。命名空间不能保证容器组（pod）之间的任何安全性。Kubernetes 不允许对给定命名空间内的容器组（pod）子集进行选择性许可。

运行批处理会话

要提交批处理作业，请运行以下命令。

```
curl -s -k -H 'Content-Type: application/json' -X POST \
      -d '{
            "name": "my-session",
            "file": "entryPoint_location (S3 or local)",
            "args": ["argument1", "argument2", ...],
            "conf": {
                "spark.kubernetes.namespace": "<spark-namespace>",
                "spark.kubernetes.container.image": "public.ecr.aws/emr-on-eks/spark/
emr-7.9.0:latest",
```

```
                "spark.kubernetes.authenticate.driver.serviceAccountName": "<spark-
service-account>"
            }
          }' <livy-endpoint>/batches
```

要监控批处理作业，请运行以下命令。

```
curl -s -k -H 'Content-Type: application/json' -X GET <livy-endpoint>/batches/my-
session
```

运行交互式会话

要使用 Apache Livy 运行交互式会话，请参阅以下步骤。

1. 确保您可以访问自托管或托管的 Jupyter Notebook SageMaker 。您的 Jupyter Notebook 必须安装了 [sparkmagic](https://github.com/jupyter-incubator/sparkmagic)。
2. 为 Spark 配置 `spark.kubernetes.file.upload.path` 创建一个存储桶。确保 Spark 服务账户拥有对存储桶的读写权限。有关如何配置 Spark 服务账户的更多详细信息，请参阅“使用服务账户的 IAM 角色（IRSA）设置访问权限”
3. 使用 `%load_ext sparkmagic.magics` 命令在 Jupyter Notebook 中加载 sparkmagic。
4. 运行 `%manage_spark` 命令以使用 Jupyter Notebook 设置 Livy 端点。选择添加端点选项卡，选择配置的身份验证类型，将 Livy 端点添加到笔记本，然后选择添加端点。
5. 再次运行 `%manage_spark` 以创建 Spark 上下文，然后转到创建会话。选择 Livy 端点，指定唯一的会话名称，选择语言，然后添加以下属性。

```
{
  "conf": {
    "spark.kubernetes.namespace": "livy-namespace",
    "spark.kubernetes.container.image": "public.ecr.aws/emr-on-eks/spark/
emr-7.9.0:latest",
    "spark.kubernetes.authenticate.driver.serviceAccountName": "<spark-service-
account>",
    "spark.kubernetes.file.upload.path": "<URI_TO_S3_LOCATION_>"
  }
}
```

6. 提交应用程序，等待其创建 Spark 上下文。
7. 要监控交互式会话的状态，请运行以下命令。

```
curl -s -k -H 'Content-Type: application/json' -X GET livy-endpoint/sessions/my-interactive-session
```

监控 Spark 应用程序

要使用 Livy UI 监控 Spark 应用程序的进度，请使用链接 `http://<livy-endpoint>/ui`。

使用 Amazon EMR on EKS 卸载 Apache Livy

按照以下步骤卸载 Apache Livy。

1. 使用命名空间名称和应用程序名称删除 Livy 设置。在此示例中，应用程序名称为 `livy-demo`，命名空间为 `livy-ns`。

   ```
   helm uninstall livy-demo -n livy-ns
   ```

2. 默认情况下，Livy 服务器和 Spark 应用程序的驱动程序和执行程序以及在本用例中使集群应用程序可用于互联网并路由由网络负载均衡器和您在安装过程中创建的目标组。 Amazon 删除资源可能需要几分钟时间。再次在命名空间上安装 Livy 之前，确保已删除资源。

3. 删除 Spark 命名空间。

   ```
   kubectl delete namespace spark-ns
   ```

使用 Amazon EMR on EKS 配置 Apache Livy 的安全性

有关使用 Amazon EMR on EKS 配置 Apache Livy 安全性的更多信息，请参阅以下主题。这些选项包括使用传输层安全性、基于角色的访问控制（基于个人在组织中的角色进行访问）以及使用 IAM 角色（根据授予的权限提供对资源的访问）。

主题

- 使用 TLS/SSL 设置安全的 Apache Livy 端点
- 使用基于角色的访问控制（RBAC）设置 Apache Livy 和 Spark 应用程序权限
- 使用服务账户的 IAM 角色（IRSA）设置访问权限

使用 TLS/SSL 设置安全的 Apache Livy 端点

请参阅以下部分，了解有关使用端到端 TL/TL/TL/TL/TL/TL/TLS 的 Apache end-to-end Livy 的更多信息。

设置 TLS 和 SSL 加密

要在 Apache Livy 端点上设置 SSL 加密，请按照以下步骤操作。

- 安装 Secrets Store CSI Driver 和 S Amazon ecrets and Configuration Provider（ASCP）：Secrets Store CSI Driver 和 ASCP 可安全存储 Livy 服务器容器组（pod）启用 SSL 所需的 Livy JKS 证书和密码 您也可以只安装 Secrets Store CSI Driver，然后使用任何其他受支持的密钥提供程序。
- 创建 ACM 证书：该证书是确保客户端与 ALB 端点之间安全连接所必需的。
- 为设置 JKS 证书、密钥和密钥库密码：这是确保 ALB 端点与 Livy 服务器之间安全连接所必需的。Amazon Secrets Manager
- 向 Livy 服务账户添加权限以从中 Amazon Secrets Manager 检索密钥：Livy 服务器需要这些权限才能从 ASCP 检索密钥，并添加 Livy 服务器的访问。要向服务账户添加 IAM 权限，请参阅“使用服务账户的 IAM 角色（IRSA）设置访问权限”。

使用的密钥和密钥库密码设置 JKS 证书 Amazon Secrets Manager

按照以下步骤使用密钥和密钥库密码设置 JKS 证书

1. 为 Livy 服务器生成密钥库文件。

```
keytool -genkey -alias <host> -keyalg RSA -keysize 2048 –dname
 CN=<host>,OU=hw,O=hw,L=<your_location>,ST=<state>,C=<country> –
keypass <keyPassword> -keystore <keystore_file> -storepass <storePassword> --
validity 3650
```

2. 创建证书。

```
keytool -export -alias <host> -keystore mykeystore.jks -rfc -
file mycertificate.cert -storepass <storePassword>
```

3. 创建信任存储库文件。

```
keytool -import -noprompt -alias <host>-file <cert_file> -
keystore <truststore_file> -storepass <truststorePassword>
```

4. 将 JKS 证书保存在中。 Amazon Secrets Manager将 `livy-jks-secret` 替换为您的密钥，将 `fileb://mykeystore.jks` 替换为密钥库 JKS 证书的路径。

    ```
    aws secretsmanager create-secret \
    --name livy-jks-secret \
    --description "My Livy keystore JKS secret" \
    --secret-binary fileb://mykeystore.jks
    ```

5. 将密钥库和密钥密码保存在 Secrets Manager 中。确保使用您自己的参数。

    ```
    aws secretsmanager create-secret \
    --name livy-jks-secret \
    --description "My Livy key and keystore password secret" \
    --secret-string "{\"keyPassword\":\"<test-key-password>\",\"keyStorePassword\":
    \"<test-key-store-password>\"}"
    ```

6. 使用以下命令创建 Livy 服务器命名空间。

    ```
    kubectl create ns <livy-ns>
    ```

7. 为具有 JKS 证书和密码的 Livy 服务器创建 `ServiceProviderClass` 对象。

    ```
    cat >livy-secret-provider-class.yaml << EOF
    apiVersion: secrets-store.csi.x-k8s.io/v1
    kind: SecretProviderClass
    metadata:
      name: aws-secrets
    spec:
      provider: aws
      parameters:
        objects: |
            - objectName: "livy-jks-secret"
              objectType: "secretsmanager"
            - objectName: "livy-passwords"
              objectType: "secretsmanager"

    EOF
    kubectl apply -f livy-secret-provider-class.yaml -n <livy-ns>
    ```

开始使用启用 SSL 的 Apache Livy

在 Livy 服务器上启用 SSL 后，必须设置 `serviceAccount` 才能访问 Amazon Secrets Manager上的 `keyStore` 和 `keyPasswords` 密钥。

1. 创建 Livy 服务器命名空间。

```
kubectl create namespace <livy-ns>
```

2. 设置 Livy 服务账户以访问 Secrets Manager 中的密钥。有关设置 IRSA 的更多信息，请参阅在安装 Apache Livy 过程中设置 IRSA。

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

3. 安装 Livy。对于 Helm 图表 --version 参数，请使用 Amazon EMR 发行版标签，例如：`7.1.0`您还必须将 Amazon ECR 注册表账户 ID 和区域 ID 替换为自己 IDs的 ID。您可以 Amazon Web Services 区域 从按区域划分的 Amazon ECR 注册表账户中找到的相应 ECR-account `ECR-registry-account` 值。

```
helm install <livy-app-name> \
  oci://895885662937.dkr.ecr.region-id.amazonaws.com/livy \
  --version 7.9.0 \
  --namespace livy-namespace-name \
  --set image=<ECR-registry-account.dkr.ecr>.<region>.amazonaws.com/livy/
emr-7.9.0:latest \
  --set sparkNamespace=spark-namespace \
  --set ssl.enabled=true
  --set ssl.CertificateArn=livy-acm-certificate-arn
  --set ssl.secretProviderClassName=aws-secrets
  --set ssl.keyStoreObjectName=livy-jks-secret
  --set ssl.keyPasswordsObjectName=livy-passwords
  --create-namespace
```

4. 继续执行在 Amazon EMR on EKS 上安装 Apache Livy 的第 5 步。

使用基于角色的访问控制（RBAC）设置 Apache Livy 和 Spark 应用程序权限

为了部署 Livy，Amazon EMR on EKS 会创建一个服务器服务账户和角色，以及一个 Spark 服务账户和角色。这些角色必须具有必要的 RBAC 权限才能完成设置和运行 Spark 应用程序。

服务器服务账户和角色的 RBAC 权限

Amazon EMR on EKS 会创建 Livy 服务器服务账户和角色，来管理 Spark 作业的 Livy 会话，以及路由进出入口和其他资源的流量。

此服务账户的默认名称为 `emr-containers-sa-livy`。必须拥有以下权限。

```
rules:
- apiGroups:
  - ""
  resources:
  - "namespaces"
  verbs:
  - "get"
- apiGroups:
  - ""
  resources:
  - "serviceaccounts"
    "services"
    "configmaps"
    "events"
    "pods"
    "pods/log"
  verbs:
  - "get"
    "list"
    "watch"
    "describe"
    "create"
    "edit"
    "delete"
    "deletecollection"
    "annotate"
    "patch"
    "label"
 - apiGroups:
   - ""
   resources:
   - "secrets"
   verbs:
   - "create"
     "patch"
     "delete"
     "watch"
```

```
 - apiGroups:
   - ""
   resources:
   - "persistentvolumeclaims"
   verbs:
   - "get"
     "list"
     "watch"
     "describe"
     "create"
     "edit"
     "delete"
     "annotate"
     "patch"
     "label"
```

Spark 服务账户和角色的 RBAC 权限

Spark 驱动程序 Pod 需要一个与该 Pod 位于同一命名空间的 Kubernetes 服务账户。该服务账户需要拥有权限才能管理执行程序容器组（pod）和驱动程序容器组（pod）所需的任何资源。除非命名空间中的默认服务账户具有所需的权限，否则驱动程序会失败并退出。需要以下 RBAC 权限。

```
rules:
- apiGroups:
  - ""
    "batch"
    "extensions"
    "apps"
  resources:
  - "configmaps"
    "serviceaccounts"
    "events"
    "pods"
    "pods/exec"
    "pods/log"
    "pods/portforward"
    "secrets"
    "services"
    "persistentvolumeclaims"
    "statefulsets"
  verbs:
  - "create"
    "delete"
```

```
    "get"
    "list"
    "patch"
    "update"
    "watch"
    "describe"
    "edit"
    "deletecollection"
    "patch"
    "label"
```

使用服务账户的 IAM 角色（IRSA）设置访问权限

默认情况下，Livy 服务器和 Spark 应用程序的驱动程序和执行程序无权 Amazon 访问资源。服务器服务账户和 Spark 服务账户控制对 Livy 服务器和 Spark 应用程序容器组（pod） Amazon 资源的访问。要授予访问权限，您需要将服务账户映射到具有必要 Amazon 权限的 IAM 角色。

您可以在安装 Apache Livy 之前、安装过程中或安装完成后设置 IRSA 映射。

在安装 Apache Livy 过程中设置 IRSA（适用于服务器服务账户）

> **Note**
>
> 只有服务器服务账户支持此映射。

1. 确保已完成设置适用于 Amazon EMR on EKS 的 Apache Livy，并正在使用 Amazon EMR on EKS 安装 Apache Livy。
2. 为 Livy 服务器创建 Kubernetes 命名空间。在此示例中，应用程序名称为 `livy-ns`。
3. 创建一个 IAM 策略，其中包含您希望容 Amazon Web Services 服务 器组（pod）访问的权限。以下示例创建了一个 IAM 策略，用于获取 Spark 入口点的 Amazon S3 资源。

```
cat >my-policy.json <<EOF{
"Version": "2012-10-17",
    "Statement": [
        {
"Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::my-spark-entrypoint-bucket"
        }
```

```
    ]
}
EOF

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

4. 使用以下命令将 Amazon Web Services 账户 ID 设置为变量。

```
account_id=$(aws sts get-caller-identity --query "Account" --output text)
```

5. 将集群的 OpenID Connect（OIDC）身份提供程序设置为环境变量。

```
oidc_provider=$(aws eks describe-cluster --name my-cluster --region $AWS_REGION --
query "cluster.identity.oidc.issuer" --output text | sed -e "s/^https:\/\///")
```

6. 为服务账户的命名空间和名称设置变量。务必使用您自己的值。

```
export namespace=default
export service_account=my-service-account
```

7. 使用以下命令创建信任策略文件。如果要向命名空间内的所有服务账户授予角色访问权限，请复制以下命令，将 `StringEquals` 替换为 `StringLike`，将 `$service_account` 替换为 `*`。

```
cat >trust-relationship.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::$account_id:oidc-provider/$oidc_provider"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "$oidc_provider:aud": "sts.amazonaws.com",
          "$oidc_provider:sub": "system:serviceaccount:$namespace:$service_account"
        }
      }
    }
  ]
}
```

```
EOF
```

8. 创建角色。

```
aws iam create-role --role-name my-role --assume-role-policy-document file://trust-
relationship.json --description "my-role-description"
```

9. 使用以下 Helm install 命令设置 `serviceAccount.executionRoleArn` 以映射 IRSA。以下是 Helm install 命令的示例。您可以 Amazon Web Services 区域 从按区域划分的 Amazon ECR 注册表账户中找到的相应 ECR-account `ECR-registry-account` 值。

```
helm install livy-demo \
  oci://895885662937.dkr.ecr.us-west-2.amazonaws.com/livy \
  --version 7.9.0 \
  --namespace livy-ns \
  --set image=ECR-registry-account.dkr.ecr.region-id.amazonaws.com/livy/
emr-7.9.0:latest \
  --set sparkNamespace=spark-ns \
  --set serviceAccount.executionRoleArn=arn:aws:iam::123456789012:role/my-role
```

将 IRSA 映射到 Spark 服务账户

将 IRSA 映射到 Spark 服务账户之前，请确保已完成以下项目：

- 确保已完成设置适用于 Amazon EMR on EKS 的 Apache Livy，并正在使用 Amazon EMR on EKS 安装 Apache Livy。
- 您的集群必须具有现有的 IAM OpenID Connect（OIDC）提供程序。要查看您是否拥有提供程序或如何创建，请参阅为集群创建 IAM OIDC 提供程序。
- 确保您已安装 0.171.0 或更高版本的 `eksctl` CLI 或 Amazon CloudShell。要安装或更新 `eksctl`，请参阅 `eksctl` 文档中的安装。

按照以下步骤将 IRSA 映射到 Spark 服务账户：

1. 使用以下命令获取 Spark 服务账户。

```
SPARK_NAMESPACE=<spark-ns>
LIVY_APP_NAME=<livy-app-name>
kubectl --namespace $SPARK_NAMESPACE describe sa -l "app.kubernetes.io/instance=
$LIVY_APP_NAME" | awk '/^Name:/ {print $2}'
```

2. 为服务账户的命名空间和名称设置变量。

```
export namespace=default
export service_account=my-service-account
```

3. 使用以下命令为 IAM 角色创建信任策略文件。下面的示例允许命名空间内的所有服务账户使用该角色。为此，请将 `StringEquals` 替换为 `StringLike`，将 `$service_account` 替换为 *。

```
cat >trust-relationship.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::$account_id:oidc-provider/$oidc_provider"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "$oidc_provider:aud": "sts.amazonaws.com",
          "$oidc_provider:sub": "system:serviceaccount:$namespace:$service_account"
        }
      }
    }
  ]
}
EOF
```

4. 创建角色。

```
aws iam create-role --role-name my-role --assume-role-policy-document file://trust-
relationship.json --description "my-role-description"
```

5. 使用以下 `eksctl` 命令映射服务器或 Spark 服务账户。务必使用您自己的值。

```
 eksctl create iamserviceaccount --name spark-sa \
 --namespace spark-namespace --cluster livy-eks-cluster \
 --attach-role-arn arn:aws:iam::0123456789012:role/my-role \
 --approve --override-existing-serviceaccounts
```

Amazon EMR on EKS 发行版上的 Apache Livy 安装属性

Apache Livy 安装允许您选择 Livy Helm 图表的版本。Helm 图表提供了各种属性来自定义您的安装和设置体验。Amazon EMR on EKS 7.1.0 及更高版本支持这些属性。

主题

- Amazon EMR 7.1.0 安装属性

Amazon EMR 7.1.0 安装属性

下表列出了所有支持的 Livy 属性。安装 Apache Livy 时，可以选择 Livy Helm 图表版本。要在安装过程中设置属性，请使用命令 `--set <property>=<value>`。

属性	描述	默认
image	Livy 服务器的 Amazon EMR 发行版 URI。这是必要的配置。	""
sparkNamespace	运行 Livy Spark 会话的命名空间。例如，指定 "livy"。这是必要的配置。	""
nameOverride	提供名称而不是 `livy`。该名称设置为所有 Livy 资源的标签	"livy"
fullnameOverride	提供要使用的名称，而不是资源的全名。	""
ssl.enabled	启用从 end-to-end Livy 端点到 Livy 服务器的端到端 SSL	FALSE
ssl.certificateArn	如果启用了 SSL，该属性为服务创建的 NLB 的 ACM 证书 ARN。	""

属性	描述	默认
SSL secretProviderClassSSL	如果启用了 SSL，该属性为使用 SSL 保护 Livy 服务器连接的 NLB 的密钥提供程序类名。	""
SSL keyStoreObjectSSL	如果启用了 SSL，该属性为密钥提供程序类中密钥库证书的对象名称。	""
SSL keyPasswordsObjectSSL	如果启用了 SSL，该属性为包含密钥库和密钥密码的密钥对象名称。	""
rbac.create	如果为 true，则创建 RBAC 资源。	FALSE
serviceAccount.create	如果为 true，则创建 Livy 服务账户。	TRUE
serviceAccount.name	用于 Livy 的服务账户名称。如果未设置此属性并创建了服务账户，则 Amazon EMR on EKS 会自动使用 `fullname` 覆盖属性生成名称。	"emr-containers-sa-livy"
服务账户。 executionRoleArn	Livy 服务账户的执行角色 ARN。	""
sparkServiceAccount.create	如果为 true，则在 `.Release.Namespace` 中创建 Spark 服务账户	TRUE

属性	描述	默认
sparkServiceAccount.name	用于 Spark 的服务账户名称。如果在未设置此属性的情况下创建了 Spark 服务账户，则 Amazon EMR on EKS 会自动生成一个名称，其属性为 `fullnameOverride` 后缀为 `-spark-livy` 。	“emr-containers-sa-spark-livy”
service.name	Livy 服务名称	`"emr-containers-livy"`
service.annotations	Livy 服务注释	`{}`
loadbalancer.enabled	是否为 Livy 服务创建一个负载均衡器，用于将 Livy 端点在 Amazon EKS 集群外部公开。	`FALSE`
loadbalancer.internal	是否将 Livy 端点配置为 VPC 内部端点或外部端点。 将此属性设置为 `FALSE`，则会将端点向 VPC 外部的源公开。我们建议使用 TLS/SSL 保护端点。有关更多信息，请参阅设置 TLS 和 SSL 加密。	`FALSE`
imagePullSecrets	用于从私有存储库中提取 Livy 映像的 `imagePullSecret` 名称列表。	[]
resources	Livy 容器的资源请求和限制。	{}
nodeSelector	要为其调度 Livy 容器组（pod）的节点。	{}
tolerations	包含要定义的 Livy 容器组（pod）容差的列表。	[]

属性	描述	默认
affinity	Livy 容器组（pod）亲和力规则。	{}
persistence.enabled	如果为 true，则为会话目录启用持久性。	FALSE
persistence.subPath	要挂载到会话目录的 PVC 子路径。	""
persistence.existingClaim	要使用的 PVC，而不是创建新的 PVC。	{}
persistence.storageClass	要使用的存储类。要定义此参数，请使用格式 `storageClassName: <storageClass>`。将该参数设置为 `"-"` 将禁用动态配置。如果将此参数设置为 null 或未指定任何内容，则 Amazon EMR on EMR on EMR on EKS 不会设置 storageClassName，storageClassName 而是使用默认置备程序。	""
persistence.accessMode	PVC 访问模式。	`ReadWriteOnce`
persistence.size	PVC 大小。	20Gi
persistence.annotations	PVC 的附加注释。	{}
env.*	要设置为 Livy 容器的其他环境。有关更多信息，请参阅在安装 Livy 时输入您自己的 Livy 和 Spark 配置。	{}
envFrom.*	要从 Kubernetes 配置映射或密钥设置为 Livy 的其他环境。	[]

属性	描述	默认
livyConf.*	要从挂载的 Kubernetes 配置映射或密钥设置的其他 livy.conf 条目。	{}
sparkDefaultsConf.*	要从挂载的 Kubernetes 配置映射或密钥设置的其他 `spark-defaults.conf` 条目。	{}

排查常见的环境变量格式错误

在输入 Livy 和 Spark 配置时，某些环境变量格式不受支持，可能会导致错误。该过程将引导您完成一系列步骤，以确保使用正确的格式。

在安装 Livy 时输入您自己的 Livy 和 Spark 配置

您可以使用 `env.*` Helm 属性配置任何 Apache Livy 或 Apache Spark 环境变量。按照以下步骤将示例配置 `example.config.with-dash.withUppercase` 转换为支持的环境变量格式。

1. 将大写字母替换为 1 和小写字母。例如，`example.config.with-dash.withUppercase` 改为 `example.config.with-dash.with1uppercase`。
2. 将破折号（-）替换为 0。例如，`example.config.with-dash.with1uppercase` 改为 `example.config.with0dash.with1uppercase`。
3. 将点（.）替换为下划线（_）。例如，`example.config.with0dash.with1uppercase` 改为 `example_config_with0dash_with1uppercase`。
4. 将所有小写字母替换为大写字母。
5. 为变量名添加前缀 `LIVY_`。
6. 使用格式--set env. *`YOUR_VARIABLE_NAME`*.value= *`yourvalue`*

例如，要设置 Livy 和 Spark 配置 `livy.server.recovery.state-store = filesystem` 和 `spark.kubernetes.executor.podNamePrefix = my-prefix`，请使用以下 Helm 属性：

```
—set env.LIVY_LIVY_SERVER_RECOVERY_STATE0STORE.value=filesystem
—set env.LIVY_SPARK_KUBERNETES_EXECUTOR_POD0NAME0PREFIX.value=myprefix
```

管理 Amazon EMR on EKS 任务运行

以下各节旨在介绍如何帮助您管理 Amazon EMR on EKS 任务运行。其中包括在使用时配置作业运行参数 Amazon CLI、配置日志数据的存储方式、运行 Spark SQL 脚本来运行查询、了解作业运行状态以及了解如何监控作业。如果要设置和完成作业运行以处理数据，通常可以按顺序浏览这些主题。

主题

- 使用管理作业运行 Amazon CLI
- 通过 StartJobRun API 运行 Spark SQL 脚本
- 任务运行状态
- 查看 Amazon EMR 控制台中的任务
- 运行任务时的常见错误

使用管理作业运行 Amazon CLI

本主题介绍如何使用 Amazon Command Line Interface (Amazon CLI) 管理作业运行。其中详细介绍了安全参数、驱动程序和各种覆盖设置等属性。还包括一些子课题，涵盖了配置日志记录的各种方法。

主题

- 配置任务运行的选项
- 配置任务运行以使用 Amazon S3 日志
- 将任务运行配置为使用 Amazon CloudWatch 日志
- 列出任务运行
- 描述任务运行
- 取消任务运行

配置任务运行的选项

使用以下选项配置任务运行参数：

- `--execution-role-arn`：您必须提供用于运行任务的 IAM 角色。有关更多信息，请参阅 将任务执行角色与 Amazon EMR on EKS 结合使用。
- `--release-label`：您可以使用 Amazon EMR 5.32.0 和 6.2.0 及更高版本部署 Amazon EMR on EKS。以前的 Amazon EMR 版本不支持 Amazon EMR on EKS。有关更多信息，请参阅 Amazon EMR on EKS 版本。

- `--job-driver`：任务驱动程序用于为主要任务提供输入。这是一个联合类型字段，您只能传递要运行的任务类型值之一。支持的任务类型包括：
  - Spark 提交任务 - 用于通过 Spark 提交运行命令。你可以使用这种作业类型通过 Spark Submit 运行 Scala PySpark、、SparkR、SparkSQL 和任何其他支持的作业。此任务类型具有以下参数：
    - Entrypoint - 这是 HCFS（兼容 Hadoop 的文件系统）对要运行的主 jar/py 文件的引用。
    - EntryPointArguments -这是您要传递给主 jar/py 文件的参数数组。您应该使用 Entrypoint 代码来处理读取这些参数。数组中的每个参数都应用逗号分隔。 EntryPointArguments 不能包含方括号或圆括号，例如 ()、{} 或 []。
    - SparkSubmitParameters -这些是您要发送给任务的其他 spark 参数。使用此参数可覆盖默认的 Spark 属性，例如驱动程序内存、—conf 或 —class 等执行程序的数量。有关更多信息，请参阅使用 spark-submit 启动应用程序。
  - Spark SQL 任务 - 用于通过 Spark SQL 运行 SQL 查询文件。您可以使用此任务类型运行 SparkSQL 任务。此任务类型具有以下参数：
    - Entrypoint - 这是 HCFS（兼容 Hadoop 的文件系统）对要运行的 SQL 查询文件的引用。

    有关可用于 Spark SQL 任务的其他 Spark 参数的列表，请参阅 通过 StartJobRun API 运行 Spark SQL 脚本。
- `--configuration-overrides`：您可以为应用程序提供配置对象以覆盖默认配置。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。可用的配置分类因 Amazon EMR 发行版而异。有关 Amazon EMR 每个发行版可用的配置分类列表，请参阅Amazon EMR on EKS 版本。

  如果在应用程序覆盖和 Spark 提交参数中传递相同的配置，则 Spark 提交参数优先。完整的配置优先级列表如下，按最高优先级到最低优先级的顺序排列。
  - 创建 `SparkSession` 时提供的配置。
  - 使用 —conf 作为 `sparkSubmitParameters` 的一部分提供的配置。
  - 作为应用程序覆盖的一部分提供的配置。
  - Amazon EMR 为发行版选择的优化配置。
  - 应用程序的默认开源配置。

  要使用 Amazon CloudWatch 或 Amazon S3 监控任务运行，您必须提供配置的详细信息 CloudWatch。有关更多信息，请参阅配置任务运行以使用 Amazon S3 日志 和将任务运行配置为使用 Amazon CloudWatch 日志。如果 S3 存储桶或 CloudWatch 日志组不存在，则 Amazon EMR 会在将日志上传到存储桶之前创建该存储桶或日志组。

- 有关 Kubernetes 配置选项的其它列表，请参阅 Kubernetes 中的 Spark 属性。

  不支持以下 Spark 配置。

  - `spark.kubernetes.authenticate.driver.serviceAccountName`
  - `spark.kubernetes.authenticate.executor.serviceAccountName`
  - `spark.kubernetes.namespace`
  - `spark.kubernetes.driver.pod.name`
  - `spark.kubernetes.container.image.pullPolicy`
  - `spark.kubernetes.container.image`

> **Note**
>
> 您可以将 `spark.kubernetes.container.image` 用于自定义 Docker 镜像。有关更多信息，请参阅 为 Amazon EMR on EKS 自定义 Docker 镜像。

配置任务运行以使用 Amazon S3 日志

为了能够监控任务进度并排除故障，您必须将任务配置为向 Amazon S3、Amazon Logs 或两者发送 CloudWatch 日志信息。本主题可提供入门知识，帮助您将通过 Amazon EMR on EKS 启动的任务的应用程序日志发布到 Amazon S3。

S3 日志 IAM policy

任务执行角色的权限策略中必须包含以下权限，然后才能将日志数据发送到 Amazon S3。将 *amzn-s3-demo-logging-bucket* 替换为日志记录存储桶的名称。

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

```
                    "arn:aws:s3:::amzn-s3-demo-logging-bucket/*",
                ]
            }
        ]
}
```

> **Note**
>
> Amazon EMR on EKS 也可以创建 Amazon S3 存储桶。如果 Amazon S3 存储桶不可用，则 IAM policy 应包含 “`s3:CreateBucket`” 权限。

在您授予执行角色将日志发送到 Amazon S3 的合适权限后，只要 `s3MonitoringConfiguration` 通过 `monitoringConfiguration` 请求的 `start-job-run` 部分，您的日志数据将发送到以下 Amazon S3 位置，如 [使用管理作业运行 Amazon CLI](#) 所示。

- 提交者日志-*virtual-cluster-id* ///jobs/ /c *job-id* ontainers/*logUri*/(stderr.gz/ *pod-name* stdout.gz)
- 驱动程序日志-*virtual-cluster-id* ///jobs/ /containers*logUri*//spark--driver/ (st *job-id* derr.gz/stdout.gz*spark-application-id*) *job-id*
- 执行者日志-*virtual-cluster-id* ///jobs/ /containers*logUri*///(st *job-id* derr.gz/stdout. *spark-application-id* gz) *executor-pod-name*

将任务运行配置为使用 Amazon CloudWatch 日志

要监控任务进度并排除故障，您必须将任务配置为向 Amazon S3、Amazon Logs 或两者发送 CloudWatch 日志信息。本主题可帮助您开始在 EKS 上使用 Amazon EMR 启动的任务上使用 CloudWatch 日志。有关 CloudWatch 日志的更多信息，请参阅 Amazon CloudWatch 用户指南中的[监控日志文件](#)。

CloudWatch 日志 IAM 策略

要将日志数据发送到 CloudWatch 日志，任务执行角色的权限策略中必须包含以下权限。将*my_log_group_name*和*my_log_stream_prefix*，分别替换为 CloudWatch 日志组的名称和日志流名称。如果日志组和日志流式传输不存在，只要执行角色 ARN 具有适当的权限，则 Amazon EMR on EKS 将创建这些日志组和日志流式传输。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:*:*:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:*:*:log-group:my_log_group_name:log-
stream:my_log_stream_prefix/*"
            ]
        }
    ]
}
```

> **Note**
>
> Amazon EMR on EKS 还可以创建日志流。如果日志流不存在，IAM policy 应包含 `"logs:CreateLogGroup"` 权限。

在您为执行角色授予适当的权限后，您的应用程序会在`start-job-run`请求`monitoringConfiguration`部分中传递其 CloudWatch 日志数据时`cloudWatchMonitoringConfiguration`将其日志数据发送到 Logs，如所示[使用管理作业运行 Amazon CLI](#)。

在 `StartJobRun` API 中，*`log_group_name`* 是的日志组名称 CloudWatch，*`log_stream_prefix`*是的日志流名称前缀 CloudWatch。您可以在 Amazon Web Services Management Console中查看和搜索这些日志。

- 提交者日志-*virtual-cluster-id* ///jobs*logGroup*//c *job-id* ontainers*logStreamPrefix*// (stderr/ *pod-name* stdout)
- 驱动程序日志-*virtual-cluster-id* ///jobs*logGroup*//containers*logStreamPrefix*//spark--driv *job-id* er/ (stderrstdout) *spark-application-id* *job-id*
- 执行者日志-*virtual-cluster-id* ///jobs*logGroup*//containers*logStreamPrefix*///(st *job-id* derr/stdout*spark-application-id*) *executor-pod-name*

列出任务运行

您可以运行 `list-job-run` 以显示任务运行的状态，如以下示例所示。

```
aws emr-containers list-job-runs --virtual-cluster-id <cluster-id>
```

描述任务运行

您可以运行 `describe-job-run` 来获取有关任务的更多详细信息，如任务状态、状态详细信息和任务名称，如以下示例所示。

```
aws emr-containers describe-job-run --virtual-cluster-id cluster-id --id job-run-id
```

取消任务运行

您可以运行 `cancel-job-run` 以取消正在运行的任务，如以下示例所示。

```
aws emr-containers cancel-job-run --virtual-cluster-id cluster-id --id job-run-id
```

通过 StartJobRun API 运行 Spark SQL 脚本

Amazon EMR on EKS 6.7.0 及更高版本包含一个 Spark SQL 任务驱动程序，可让您通过 `StartJobRun` API 运行 Spark SQL 脚本。您可以提供 SQL 入口点文件，以使用 `StartJobRun` API 在 Amazon EMR on EKS 上直接运行 Spark SQL 查询，而无需对现有 Spark SQL 脚本进行任何修改。下表列出了通过 StartJobRun API 支持 Spark SQL 作业的 Spark 参数。

您可以选择以下 Spark 参数发送到 Spark SQL 任务。使用这些参数覆盖默认 Spark 属性。

选项	描述
--name NAME	应用程序名称

选项	描述
-jars JARS	以逗号分隔的 jar 列表，这些元素包含在驱动程序和执行程序的类路径中。
--packages	以逗号分隔的 jar 的 maven 坐标列表，这些坐标包含在驱动程序和执行程序的类路径中。
--exclude-packages	以逗号分隔的 groupId:artifactId 列表，在解析 –packages 中提供的依赖项时要排除这些元素以避免依赖项冲突。
--repositories	以逗号分隔的其他远程存储库列表，用于搜索 –packages 给出的 maven 坐标。
--files FILES	以逗号分隔的文件列表，这些文件放置在每个执行器的工作目录中。
--conf PROP=VALUE	Spark 配置属性。
--properties-file FILE	从中加载额外属性的文件的路径。
--driver-memory MEM	驱动程序的内存。默认值为 1024MB。
--driver-java-options	传递给驱动程序的额外 Java 选项。
--driver-library-path	传递给驱动程序的额外库路径条目。
--driver-class-path	传递给驱动程序的额外类路径条目。
--executor-memory MEM	每个执行器的内存。默认为 1GB。
--driver-cores NUM	驱动程序使用的内核数。
--total-executor-cores NUM	所有执行器的内核总数。
--executor-cores NUM	每个执行程序使用的内核数。
--num-executors NUM	要启动的执行程序数。

选项	描述
-hivevar <key=value>	适用于 Hive 命令的变量替换，例如 `-hivevar A=B`
-hiveconf <property=value>	用于给定属性的值。

对于 Spark SQL 作业，请创建一个 start-job-run-request .json 文件并指定作业运行所需的参数，如以下示例所示：

```
{
  "name": "myjob",
  "virtualClusterId": "123456",
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "emr-6.7.0-latest",
  "jobDriver": {
    "sparkSqlJobDriver": {
      "entryPoint": "entryPoint_location",
       "sparkSqlParameters": "--conf spark.executor.instances=2 --conf
 spark.executor.memory=2G --conf spark.executor.cores=2 --conf spark.driver.cores=1"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.driver.memory":"2G"
         }
      }
    ],
    "monitoringConfiguration": {
      "persistentAppUI": "ENABLED",
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "my_log_group",
        "logStreamNamePrefix": "log_stream_prefix"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://my_s3_log_location"
      }
    }
```

```
    }
}
```

任务运行状态

当您将一个任务运行提交到 Amazon EMR on EKS 的任务队列时，该任务运行将进入 `PENDING` 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）。

任务运行可具有以下状态：

- `PENDING` - 将任务运行提交到 Amazon EMR on EKS 时的初始任务状态。任务正在等待提交到虚拟集群，并且 Amazon EMR on EKS 正在提交此任务。
- `SUBMITTED` - 已成功提交到虚拟集群的任务运行。然后，集群调度器尝试在集群上运行此任务。
- `RUNNING` - 任务运行正在虚拟集群中运行。在 Spark 应用程序中，这意味着 Spark 驱动程序进程处于 `running` 状态。
- `FAILED` - 任务运行未能提交到虚拟集群或未成功完成提交。查看 StateDetails 并 FailureReason 查找有关此任务失败的更多信息。
- `COMPLETED` - 任务运行已成功完成。
- `CANCEL_PENDING` - 任务运行已请求取消。Amazon EMR on EKS 正在尝试取消虚拟集群上的任务。
- `CANCELLED` - 任务运行已成功取消。

查看 Amazon EMR 控制台中的任务

您可以查看作业运行数据，从而监控每个作业的运行状态。要在 Amazon EMR 控制台中查看任务，请执行以下步骤。

1. 在 Amazon EMR 控制台左侧菜单中，从 Amazon EMR on EKS 下，选择 Virtual clusters（虚拟集群）。
2. 从虚拟集群列表中，选择您要查看任务的虚拟集群。
3. 在任务运行表中，选择 View logs (查看日志) 以查看任务运行的详细信息。

> **Note**
> 默认情况下，系统会启用一键单击体验支持。在任务提交过程中，可通过将 `monitoringConfiguration` 中的设置从 `persistentAppUI` 改为 `DISABLED`，关闭提交。有关更多信息，请参阅查看持久性应用程序用户界面。

运行任务时的常见错误

当您运行 `StartJobRun` API 时，可能会发生以下错误。表中列出了每个错误，提供了缓解措施，以便您快速解决问题。

错误消息	错误条件	向您建议的后续步骤
错误：参数--*argument* 为必填项	缺少必需参数。	将缺少的实参添加到 API 请求中。
调用 StartJobRun操作时出错 (AccessDeniedException)：用户：无权执行：emr-con *ARN* tainers：StartJobRun	缺少执行角色。	请参阅使用 将任务执行角色与 Amazon EMR on EKS 结合使用。
调用 StartJobRun操作时出错 (AccessDeniedException)：用户：无权执行：emr-con *ARN* tainers：StartJobRun	调用方没有权限通过条件键访问执行角色 [[有效/无效格式]。	请参阅 将任务执行角色与 Amazon EMR on EKS 结合使用。
调用 StartJobRun操作时出错 (AccessDeniedException)：用户：无权执行：emr-con *ARN* tainers：StartJobRun	任务提交者和执行角色 ARN 来自不同的账户。	确保任务提交者和执行角色 ARN 来自同一 Amazon 账户。
检测到 1 个验证错误：“ex ecutionRoleArn” *Role* 处的值未能满足 ARN 正则表达式模式：^arn :(aws [a-za-z0-9-] *): iam:: (\ d {12}) ？：(角色 (\ u002f) | (\ u002F [\ u0021-\	调用方拥有通过条件键执行角色的权限，但该角色不满足 ARN 格式的约束条件。	提供遵循 ARN 格式的执行角色。请参阅 将任务执行角色与 Amazon EMR on EKS 结合使用。

错误消息	错误条件	向您建议的后续步骤
u007F] +\ u002F)) [\ w+=, .@-] +) [\ w+=, .@-] +)		
调用 StartJobRun操作时出错 (ResourceNotFoundException)：虚拟群集*Virtual Cluster ID*不存在。	找不到虚拟集群 ID。	请提供在 Amazon EMR on EKS 上注册的虚拟集群 ID。
调用 StartJobRun操作时出现错误 (ValidationException)：虚拟群集*state*状态对创建资源无效 JobRun。	虚拟集群尚未准备好执行任务。	请参阅 虚拟集群状态。
调用 StartJobRun操作时出错 (ResourceNotFoundException)：版本*RELEASE*不存在。	任务提交中指定的版本不正确。	请参阅 Amazon EMR on EKS 版本。
调用 StartJobRun操作时出现错误 (AccessDeniedException)：用户：无权在资源StartJobRun 上执行：*ARN*emr-containers：*ARN*并明确拒绝。 调用 StartJobRun操作时出错 (AccessDeniedException)：用户：无权在资源StartJobRun 上执行：*ARN*emr-containers：*ARN*	用户无权拨打电话 StartJobRun。	请参阅 将任务执行角色与 Amazon EMR on EKS 结合使用。
调用 StartJobRun操作时出错 (ValidationException)：ConfigurationOverrides.monitoringConfiguration.s3 .log MonitoringConfiguration uri 无法满足约束条件：%s	S3 路径 URI 语法无效。	logUri 的格式应为 s3://...

当您在任务运行之前运行 `DescribeJobRun` API 时，可能会发生以下错误。

错误消息	错误条件	向您建议的后续步骤
状态详情： JobRun 提交失败。 *classification* 不支持分类。 failureReason：VALIDATION_ERROR state: FAILED。	中的 StartJobRun 参数无效。	请参阅 Amazon EMR on EKS 版本。
状态详情：集群*EKS Cluster ID*不存在。 failureReason: CLUSTER_UNAVAILABLE state: FAILED	EKS 集群不可用。	检查 EKS 集群是否存在并拥有适当的权限。有关更多信息，请参阅 设置 Amazon EMR on EKS。
状态详情：集群权限*EKS Cluster ID*不足。 failureReason: CLUSTER_UNAVAILABLE state: FAILED	Amazon EMR 无权访问 EKS 集群。	验证是否在已注册的命名空间上为 Amazon EMR 设置了权限。有关更多信息，请参阅 设置 Amazon EMR on EKS。
状态详情：集群*EKS Cluster ID*目前无法访问。 failureReason: CLUSTER_UNAVAILABLE state: FAILED	无法访问 EKS 集群。	检查 EKS 集群是否存在并拥有适当的权限。有关更多信息，请参阅 设置 Amazon EMR on EKS。
状态详情：由于内部错误，JobRun 提交失败。	EKS 集群内部出现错误。	不适用

错误消息	错误条件	向您建议的后续步骤
failureReason: INTERNAL_ERROR state: FAILED		
状态详情：集群*EKS Cluster ID*没有足够的资源。 failureReason: USER_ERROR state: FAILED	EKS 集群中没有足够的资源来运行任务。	向 EKS 节点组添加更多容量或设置 EKS Autoscaler。有关更多信息，请参阅 Cluster Autoscaler。

当您在任务运行后运行 `DescribeJobRun` API 时，会出现以下错误。

错误消息	错误条件	向您建议的后续步骤
StateDetails: 无法监控您的 JobRun. 集群*EKS Cluster ID*不存在。 failureReason: CLUSTER_UNAVAILABLE state: FAILED	该 EKS 集群不存在。	检查 EKS 集群是否存在并拥有适当的权限。有关更多信息，请参阅 设置 Amazon EMR on EKS。
StateDetails: 无法监控您的 JobRun. 集群权限*EKS Cluster ID*不足。 failureReason: CLUSTER_UNAVAILABLE state: FAILED	Amazon EMR 无权访问 EKS 集群。	验证是否在已注册的命名空间上为 Amazon EMR 设置了权限。有关更多信息，请参阅 设置 Amazon EMR on EKS。

错误消息	错误条件	向您建议的后续步骤
StateDetails: 无法监控您的 JobRun. *EKS Cluster ID*目前无法访问集群。 failureReason: CLUSTER_UNAVAILABLE state: FAILED	无法访问 EKS 集群。	检查 EKS 集群是否存在并拥有适当的权限。有关更多信息，请参阅 [设置 Amazon EMR on EKS](#)。
StateDetails： JobRun 由于内部错误，无法监控您的状态 failureReason: INTERNAL_ERROR state: FAILED	发生了内部错误，无法进行 JobRun 监控。	不适用

当作业无法启动且作业在“已提交”状态下等待 15 分钟时，可能会发生以下错误。这可能是由于集群资源缺失导致的。

错误消息	错误条件	向您建议的后续步骤
集群超时	作业处于“已提交”状态已有 15 分钟或更长时间。	您可以使用下面所示的配置覆盖来覆盖此参数的默认设置 15 分钟。

使用以下配置，将集群超时设置更改为 30 分钟。请注意，您提供的新 `job-start-timeout` 值应以秒为单位：

```
{
"configurationOverrides": {
  "applicationConfiguration": [{
      "classification": "emr-containers-defaults",
      "properties": {
```

```
            "job-start-timeout":"1800"
        }
    }]
}
```

使用任务模板

在启动任务运行时，任务模板中存储的值可以在 `StartJobRun` API 调用之间共享。它支持两种使用场景：

- 防止重复出现周期性的 `StartJobRun` API 请求值。
- 强制实施必须通过 `StartJobRun` API 请求提供某些值的规则。

借助任务模板，您可以定义可重复使用的任务运行模板，以应用额外的自定义，例如：

- 配置执行程序和驱动程序的计算容量
- 设置安全性和监管属性，例如 IAM 角色
- 自定义 Docker 映像以跨多个应用程序和数据管道使用

以下主题提供了有关使用模板的详细信息，包括如何使用模板启动作业运行以及如何更改模板参数。

主题

- 创建并使用作业模板启动作业运行
- 定义任务模板参数
- 控制对任务模板的访问权限

创建并使用作业模板启动作业运行

本节介绍如何创建作业模板以及使用模板通过 Amazon Command Line Interface (Amazon CLI) 开始作业运行。

创建任务模板

1. 创建一个 `create-job-template-request.json` 文件并指定任务模板所需的参数，如下面的示例 JSON 文件所示。有关所有可用参数的信息，请参阅 CreateJobTemplateAPI。

StartJobRun API 所需的大多数值也是 jobTemplateData 所必需的。如果您想为任何参数使用占位符并在 StartJobRun 使用作业模板调用时提供值，请参阅下一节有关作业模板参数的内容。

```
{
   "name": "mytemplate",
   "jobTemplateData": {
        "executionRoleArn": "iam_role_arn_for_job_execution",
        "releaseLabel": "emr-6.7.0-latest",
        "jobDriver": {
            "sparkSubmitJobDriver": {
                "entryPoint": "entryPoint_location",
                "entryPointArguments": [ "argument1","argument2",...],
                "sparkSubmitParameters": "--class <main_class> --conf
 spark.executor.instances=2 --conf spark.executor.memory=2G --conf
 spark.executor.cores=2 --conf spark.driver.cores=1"
            }
        },
        "configurationOverrides": {
            "applicationConfiguration": [
                {
                    "classification": "spark-defaults",
                    "properties": {
                         "spark.driver.memory":"2G"
                    }
                }
            ],
            "monitoringConfiguration": {
                "persistentAppUI": "ENABLED",
                "cloudWatchMonitoringConfiguration": {
                    "logGroupName": "my_log_group",
                    "logStreamNamePrefix": "log_stream_prefix"
                },
                "s3MonitoringConfiguration": {
                    "logUri": "s3://my_s3_log_location/"
                }
            }
        }
     }
}
```

2. 使用 create-job-template 命令和存储在本地的 create-job-template-request.json 文件路径。

```
aws emr-containers create-job-template \
--cli-input-json file://./create-job-template-request.json
```

使用任务模板启动任务运行

在 StartJobRun 命令中提供虚拟集群 ID、任务模板 ID 和任务名称，如以下示例所示。

```
aws emr-containers start-job-run \
--virtual-cluster-id 123456 \
--name myjob \
--job-template-id 1234abcd
```

定义任务模板参数

借助任务模板参数，您可以在任务模板中指定变量。使用该任务模板启动任务运行时，您需要指定这些参数变量的值。任务模板参数以 ${parameterName} 格式指定。您可以选择将 jobTemplateData 字段中的任何值指定为任务模板参数。指定每个任务模板参数变量的数据类型（STRING 或 NUMBER），此外还可以选择指定一个默认值。以下示例演示了如何为入口点位置、主类和 S3 日志位置值指定任务模板参数。

将入口点位置、主类和 Amazon S3 日志位置指定为任务模板参数

1. 创建一个 create-job-template-request.json 文件并指定任务模板所需的参数，如下面的示例 JSON 文件所示。有关参数的更多信息，请参阅 CreateJobTemplateAPI。

```
{
   "name": "mytemplate",
   "jobTemplateData": {
        "executionRoleArn": "iam_role_arn_for_job_execution",
        "releaseLabel": "emr-6.7.0-latest",
        "jobDriver": {
            "sparkSubmitJobDriver": {
                "entryPoint": "${EntryPointLocation}",
                "entryPointArguments": [ "argument1","argument2",...],
                "sparkSubmitParameters": "--class ${MainClass} --conf
 spark.executor.instances=2 --conf spark.executor.memory=2G --conf
 spark.executor.cores=2 --conf spark.driver.cores=1"
            }
        },
        "configurationOverrides": {
```

```
            "applicationConfiguration": [
                {
                    "classification": "spark-defaults",
                    "properties": {
                         "spark.driver.memory":"2G"
                    }
                }
            ],
            "monitoringConfiguration": {
                "persistentAppUI": "ENABLED",
                "cloudWatchMonitoringConfiguration": {
                    "logGroupName": "my_log_group",
                    "logStreamNamePrefix": "log_stream_prefix"
                },
                "s3MonitoringConfiguration": {
                    "logUri": "${LogS3BucketUri}"
                }
            }
        },
        "parameterConfiguration": {
            "EntryPointLocation": {
                "type": "STRING"
            },
            "MainClass": {
                "type": "STRING",
                "defaultValue":"Main"
            },
            "LogS3BucketUri": {
                "type": "STRING",
                "defaultValue":"s3://my_s3_log_location/"
            }
        }
    }
}
```

2. 使用 `create-job-template` 命令和存储在本地或 Amazon S3 上的 `create-job-template-request.json` 文件路径。

```
aws emr-containers create-job-template \
--cli-input-json file://./create-job-template-request.json
```

使用具有任务模板参数的任务模板启动任务运行

要使用包含任务模板参数的任务模板启动运行任务，请在 StartJobRun API 请求中指定任务模板 ID 以及任务模板参数的值，如下例所示。

```
aws emr-containers start-job-run \
--virtual-cluster-id 123456 \
--name myjob \
--job-template-id 1234abcd \
--job-template-parameters '{"EntryPointLocation": "entry_point_location","MainClass":
 "ExampleMainClass","LogS3BucketUri": "s3://example_s3_bucket/"}'
```

控制对任务模板的访问权限

借助 StartJobRun 策略，您可以强制要求某个用户或角色只能使用您指定的任务模板运行任务，如果不使用指定的任务模板，则无法运行 StartJobRun 操作。要实现此目的，请首先确保向该用户或角色授予对指定任务模板的读取权限，如下例所示。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "emr-containers:DescribeJobTemplate",
            "Resource": [
                "job_template_1_arn",
                "job_template_2_arn",
                ...
            ]
        }
    ]
}
```

要强制某个用户或角色只能在使用指定任务模板时调用 StartJobRun 操作，您可以将以下 StartJobRun 策略权限分配给该用户或角色。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "emr-containers:StartJobRun",
            "Resource": [
                "virtual_cluster_arn",
```

```
            ],
            "Condition": [
                "StringEquals": {
                    "emr-containers:JobTemplateArn": [
                        "job_template_1_arn",
                        "job_template_2_arn",
                        ...
                    ]
                 }
                ]
            }
        }
    ]
}
```

如果任务模板在执行角色 ARN 字段中指定了任务模板参数，则用户将能够提供该参数的值，从而使用任意执行角色来调用 `StartJobRun`。要限制用户可以提供的执行角色，请参阅 将任务执行角色与 Amazon EMR on EKS 结合使用 中的控制对执行角色的访问权限。

如果在上述 `StartJobRun` 操作策略中没有为给定用户或角色指定任何条件，则表示允许该用户或角色使用其具有读取权限的任意任务模板或使用任意执行角色在指定的虚拟集群上调用 `StartJobRun` 操作。

使用 Pod 模板

从 Amazon EMR 版本 5.33.0 或 6.3.0 开始，Amazon EMR on EKS 支持 Spark 的 Pod 模板功能。Pod 是包含一个或多个容器的容器组，包含共享存储和网络资源，以及如何运行容器的规范。Pod 模板是决定如何运行每个容器的规范。您可以使用 Pod 模板文件定义 Spark 配置不支持的驱动程序或执行程序 Pod 的配置。有关 Spark 的 Pod 模板功能的更多信息，请参阅 Pod 模板。

> **Note**
>
> Pod 模板功能仅适用于驱动程序和执行程序 Pod。无法使用容器组（pod）模板来配置作业提交者容器组（pod）。

常见场景

您可以通过将 Pod 模板与 Amazon EMR on EKS 结合使用，来定义如何在共享 EKS 集群上运行 Spark 任务，从而节省成本并提高资源利用率和性能。

- 为了降低成本，您可以将 Spark 驱动程序任务安排为在亚马逊 EC2 按需实例上运行，同时安排 Spark 执行器任务在亚马逊 EC2 竞价型实例上运行。
- 为了提高资源利用率，您可以支持多个团队在同一 EKS 集群上运行其工作负载。每个团队都将获得一个指定的 Amazon EC2 节点组来运行其工作负载。您可以使用 Pod 模板对其工作负载应用相应的容忍度。
- 为了改进监控，您可以运行单独的日志记录容器，将日志转发到现有的监控应用程序。

例如，以下 Pod 模板文件演示了常见的使用场景。

```
apiVersion: v1
kind: Pod
spec:
  volumes:
    - name: source-data-volume
      emptyDir: {}
    - name: metrics-files-volume
      emptyDir: {}
  nodeSelector:
    eks.amazonaws.com/nodegroup: emr-containers-nodegroup
  containers:
  - name: spark-kubernetes-driver # This will be interpreted as driver Spark main
 container
    env:
      - name: RANDOM
        value: "random"
    volumeMounts:
      - name: shared-volume
        mountPath: /var/data
      - name: metrics-files-volume
        mountPath: /var/metrics/data
  - name: custom-side-car-container # Sidecar container
    image: <side_car_container_image>
    env:
      - name: RANDOM_SIDECAR
        value: random
    volumeMounts:
      - name: metrics-files-volume
        mountPath: /var/metrics/data
    command:
      - /bin/sh
      - '-c'
```

```
      -  <command-to-upload-metrics-files>
  initContainers:
  - name: spark-init-container-driver # Init container
    image: <spark-pre-step-image>
    volumeMounts:
      - name: source-data-volume # Use EMR predefined volumes
        mountPath: /var/data
    command:
      - /bin/sh
      - '-c'
      -  <command-to-download-dependency-jars>
```

Pod 模板完成以下任务：

- 添加一个新的 init 容器（初始化容器），并在 Spark 主容器启动之前执行。初始化容器`source-data-volume`与 Spark 主容器共享调用的EmptyDir卷。您可让您的 init 容器运行初始化步骤，例如下载依赖项或生成输入数据。然后 Spark 主容器使用数据。
- 添加其它 Sidecar 容器（边车容器），与 Spark 主容器一起执行。这两个容器共享另一个称为 `metrics-files-volume` 的 `EmptyDir` 卷。您的 Spark 任务可以生成指标，例如 Prometheus 指标。然后，Spark 任务可以将指标放入文件中，并让边车容器将文件上载到您自己的 BI 系统，以供将来分析。
- 将新的环境变量添加到 Spark 主容器中。您可以让您的任务使用环境变量。
- 定义节点选择器，以便该 Pod 仅安排在 `emr-containers-nodegroup` 节点组。这有助于各任务和团队隔离计算资源。

利用 Amazon EMR on EKS 启用 Pod 模板

要使用 Amazon EMR on EKS 启用 Pod 模板功能，请配置 Spark 属性 `spark.kubernetes.driver.podTemplateFile` 和 `spark.kubernetes.executor.podTemplateFile`，以指向 Amazon S3 中的 Pod 模板文件。然后，Spark 下载 Pod 模板文件并使用它来构建驱动程序和执行程序 Pod。

> **Note**
>
> Spark 使用任务执行角色加载 Pod 模板，因此任务执行角色必须具有访问 Amazon S3 才能加载 Pod 模板的权限。有关更多信息，请参阅创建任务执行角色。

您可以使用 `SparkSubmitParameters` 来指定 Pod 模板的 Amazon S3 路径，如以下任务运行 JSON 文件所示。

```
{
  "name": "myjob",
  "virtualClusterId": "123456",
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "release_label",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "entryPoint_location",
      "entryPointArguments": ["argument1", "argument2", ...],
       "sparkSubmitParameters": "--class <main_class> \
         --conf
 spark.kubernetes.driver.podTemplateFile=s3://path_to_driver_pod_template \
         --conf
 spark.kubernetes.executor.podTemplateFile=s3://path_to_executor_pod_template \
         --conf spark.executor.instances=2 \
         --conf spark.executor.memory=2G \
         --conf spark.executor.cores=2 \
         --conf spark.driver.cores=1"
    }
  }
}
```

此外，您也可以使用 `configurationOverrides` 来指定 Pod 模板的 Amazon S3 路径，如以下任务运行 JSON 文件所示。

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

```
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.driver.memory":"2G",
          "spark.kubernetes.driver.podTemplateFile":"s3://path_to_driver_pod_template",

 "spark.kubernetes.executor.podTemplateFile":"s3://path_to_executor_pod_template"
        }
      }
    ]
  }
}
```

> **Note**
>
> 1. 将 Pod 模板功能与 Amazon EMR on EKS 结合使用时，您需要遵循安全指南，例如隔离不受信任的应用程序代码。有关更多信息，请参阅Amazon EMR on EKS 安全最佳实践。
> 2. 您无法通过使用 `spark.kubernetes.driver.podTemplateContainerName` 和 `spark.kubernetes.executor.podTemplateContainerName` 来更改 Spark 主容器名称，因为这些名称已被硬编码为 `spark-kubernetes-driver` 和 `spark-kubernetes-executors`。如果要自定义 Spark 主容器，则必须使用这些硬编码名称在 Pod 模板中指定容器。

Pod 模板字段

使用 Amazon EMR on EKS 配置 Pod 模板时，请考虑以下字段限制。

- Amazon EMR on EKS 仅允许 Pod 模板中的以下字段启用合适的任务调度。

  以下是所允许的 Pod 级别字段：

  - `apiVersion`
  - `kind`
  - `metadata`
  - `spec.activeDeadlineSeconds`

- `spec.affinity`
- `spec.containers`
- `spec.enableServiceLinks`
- `spec.ephemeralContainers`
- `spec.hostAliases`
- `spec.hostname`
- `spec.imagePullSecrets`
- `spec.initContainers`
- `spec.nodeName`
- `spec.nodeSelector`
- `spec.overhead`
- `spec.preemptionPolicy`
- `spec.priority`
- `spec.priorityClassName`
- `spec.readinessGates`
- `spec.runtimeClassName`
- `spec.schedulerName`
- `spec.subdomain`
- `spec.terminationGracePeriodSeconds`
- `spec.tolerations`
- `spec.topologySpreadConstraints`
- `spec.volumes`

这些是所允许的 Spark 主容器级别字段：

- `env`
- `envFrom`
- `name`
- `lifecycle`
- `livenessProbe`
- `readinessProbe`
- `resources`

  - `startupProbe`
  - `stdin`
  - `stdinOnce`
  - `terminationMessagePath`
  - `terminationMessagePolicy`
  - `tty`
  - `volumeDevices`
  - `volumeMounts`
  - `workingDir`

  当您在 Pod 模板中使用任何不允许的字段时，Spark 将引发异常，并且任务将失败。以下示例显示，由于不允许字段而导致 Spark 控制器日志中出现错误消息。

```
Executor pod template validation failed.
Field container.command in Spark main container not allowed but specified.
```

- Amazon EMR on EKS 预定义了 Pod 模板中的以下参数。您在 Pod 模板中指定的字段不得与这些字段重叠。

  这些是预定义的卷名称：

  - `emr-container-communicate`
  - `config-volume`
  - `emr-container-application-log-dir`
  - `emr-container-event-log-dir`
  - `temp-data-dir`
  - `mnt-dir`
  - `home-dir`
  - `emr-container-s3`

  这些是仅适用于 Spark 主容器的预定义卷挂载：

  - 名称:`emr-container-communicate`; MountPath: `/var/log/fluentd`
  - 名称:`emr-container-application-log-dir`; MountPath: `/var/log/spark/user`
  - 名称:`emr-container-event-log-dir`; MountPath: `/var/log/spark/apps`

  - 名称:`mnt-dir`; MountPath: `/mnt`

- 名称:`temp-data-dir`; MountPath: `/tmp`
- 名称:`home-dir`; MountPath: `/home/hadoop`

这些是仅适用于 Spark 主容器的预定义环境变量：

- `SPARK_CONTAINER_ID`
- `K8S_SPARK_LOG_URL_STDERR`
- `K8S_SPARK_LOG_URL_STDOUT`
- `SIDECAR_SIGNAL_FILE`

> **Note**
>
> 您仍然可以使用这些预定义卷，并将它们挂载到其它边车容器中。例如，您可以使用 `emr-container-application-log-dir`，并将其挂载到您在 Pod 模板中定义的边车容器上。

如果指定的字段与 Pod 模板中的任何预定义字段冲突，Spark 将引发异常，并且任务将失败。以下示例显示，由于与预定义字段冲突，Spark 应用程序日志出现了错误消息。

```
Defined volume mount path on main container must not overlap with reserved mount
 paths: [<reserved-paths>]
```

边车容器注意事项

Amazon EMR 控制由 Amazon EMR on EKS 预置的 Pod 生命周期。边车容器应遵循与 Spark 主容器相同的生命周期。如果您在 Pod 中注入额外的边车容器，我们建议您与 Amazon EMR 定义的 Pod 生命周期管理集成，以便在 Spark 主容器退出时，边车容器可以自行停止。

为了降低成本，我们建议您实施一个流程，以防止存在边车容器的驱动程序 Pod 在任务完成后继续运行。完成执行程序后，Spark 驱动程序会删除执行程序 Pod。但是，当驱动程序完成后，其它边车容器将继续运行。在 Amazon EMR on EKS 清除 Pod 驱动程序之前，Pod 都会产生费用，通常在驱动程序 Spark 主容器完成后不到一分钟内产生费用。为了降低成本，您可以将更多边车容器与生命周期管理机制集成，以便 Amazon EMR on EKS 为驱动程序和执行程序 Pod 定义生命周期管理机制，如下节所述。

在驱动程序和执行程序 Pod 中的 Spark 主容器每两秒钟将 `heartbeat` 发送到文件 `/var/log/fluentd/main-container-terminated`。通过向您的边车容器添加 Amazon EMR 预定义的 `emr-container-communicate` 卷挂载，您可以定义 Sidecar 容器的子进程，以定期跟踪此文件的上次修改时间。然后，如果子进程发现 Spark 主容器在较长持续时间内停止使用 `heartbeat`，则子进程会自行停止。

以下示例演示了跟踪检测信号文件并自行停止的子进程。*`your_volume_mount`*替换为挂载预定义卷的路径。脚本捆绑在边车容器使用的镜像内部。在 Pod 模板文件中，您可以使用以下命令 `sub_process_script.sh` 和 `main_command` 来指定边车容器。

```
MOUNT_PATH="your_volume_mount"
FILE_TO_WATCH="$MOUNT_PATH/main-container-terminated"
INITIAL_HEARTBEAT_TIMEOUT_THRESHOLD=60
HEARTBEAT_TIMEOUT_THRESHOLD=15
SLEEP_DURATION=10

function terminate_main_process() {
  # Stop main process
}

# Waiting for the first heartbeat sent by Spark main container
echo "Waiting for file $FILE_TO_WATCH to appear..."
start_wait=$(date +%s)
while ! [[ -f "$FILE_TO_WATCH" ]]; do
    elapsed_wait=$(expr $(date +%s) - $start_wait)
    if [ "$elapsed_wait" -gt "$INITIAL_HEARTBEAT_TIMEOUT_THRESHOLD" ]; then
        echo "File $FILE_TO_WATCH not found after $INITIAL_HEARTBEAT_TIMEOUT_THRESHOLD
 seconds; aborting"
        terminate_main_process
        exit 1
    fi
    sleep $SLEEP_DURATION;
done;
echo "Found file $FILE_TO_WATCH; watching for heartbeats..."

while [[ -f "$FILE_TO_WATCH" ]]; do
    LAST_HEARTBEAT=$(stat -c %Y $FILE_TO_WATCH)
    ELAPSED_TIME_SINCE_AFTER_HEARTBEAT=$(expr $(date +%s) - $LAST_HEARTBEAT)
    if [ "$ELAPSED_TIME_SINCE_AFTER_HEARTBEAT" -gt "$HEARTBEAT_TIMEOUT_THRESHOLD" ];
 then
        echo "Last heartbeat to file $FILE_TO_WATCH was more than
 $HEARTBEAT_TIMEOUT_THRESHOLD seconds ago at $LAST_HEARTBEAT; terminating"
```

```
        terminate_main_process
        exit 0
    fi
    sleep $SLEEP_DURATION;
done;
echo "Outside of loop, main-container-terminated file no longer exists"

# The file will be deleted once the fluentd container is terminated

echo "The file $FILE_TO_WATCH doesn't exist any more;"
terminate_main_process
exit 0
```

使用作业重试策略

在 Amazon EMR on EKS 6.9.0 版本及更高版本中，您可以为作业运行设置重试策略。重试策略会在作业驱动程序容器组（pod）失败或被删除时自动重新启动。这使得长时间运行的 Spark 流式传输作业在出现故障时更具弹性。

设置任务的重试策略

要配置重试策略，您需要使用 StartJobRunAPI 提供一个`RetryPolicyConfiguration`字段。此处显示了一个示例 `retryPolicyConfiguration`：

```
aws emr-containers start-job-run \
--virtual-cluster-id cluster_id \
--name sample-job-name \
--execution-role-arn execution-role-arn \
--release-label emr-6.9.0-latest \
--job-driver '{
  "sparkSubmitJobDriver": {
    "entryPoint": "local:///usr/lib/spark/examples/src/main/python/pi.py",
    "entryPointArguments": [ "2" ],
    "sparkSubmitParameters": "--conf spark.executor.instances=2 --conf
 spark.executor.memory=2G --conf spark.executor.cores=2 --conf spark.driver.cores=1"
  }
}' \
--retry-policy-configuration '{
    "maxAttempts": 5
  }' \
--configuration-overrides '{
```

```
  "monitoringConfiguration": {
    "cloudWatchMonitoringConfiguration": {
      "logGroupName": "my_log_group_name",
      "logStreamNamePrefix": "my_log_stream_prefix"
    },
    "s3MonitoringConfiguration": {
       "logUri": "s3://amzn-s3-demo-logging-bucket"
    }
  }
}'
```

> **Note**
>
> `retryPolicyConfiguration`仅从 Amazon CLI 1.27.68 版本开始可用。要将更新 Amazon CLI 到最新版本，请参阅安装或更新最新版本的 Amazon CLI

为此 `maxAttempts` 字段配置您希望在作业驱动程序容器组（pod）失败或被删除时重新启动的最大次数。两次作业驱动程序重试之间的执行间隔是指数重试间隔（10 秒、20 秒、40 秒...），上限为 6 分钟，如 Kubernetes 文档中所述。

> **Note**
>
> 每额外执行一次作业驱动程序都将作为另一次作业运行进行计费，并根据 Amazon EMR on EKS 定价而定。

重试策略配置值

- 作业的默认重试策略：`StartJobRun` 包括默认情况下最大尝试次数设置为 1 的重试策略。您可以根据需要配置重试策略。

  > **Note**
  >
  > 如果 `retryPolicyConfiguration` 的 `maxAttempts` 设置为 1，则表示在失败时不会重试启动驱动程序容器组（pod）。

- 禁用任务的重试策略：要禁用重试策略，请将中的最大尝试次数值设置 retryPolicyConfiguration 为 1。

```
"retryPolicyConfiguration": {
    "maxAttempts": 1
}
```

- 在有效范围内为作业设置 maxAttempts：如果 `maxAttempts` 值超出有效范围，则 `StartJobRun` 调用将失败。有效 `maxAttempts` 范围介于 1 到 2147483647（32 位整数）之间，该范围受 Kubernetes 的 `backOffLimit` 配置设置支持。有关更多信息，请参阅 Kubernetes 文档中的 [容器组（pod）回退失败策略]。如果该 `maxAttempts` 值无效，则系统会返回以下错误消息：

```
{
 "message": "Retry policy configuration's parameter value of maxAttempts is invalid"
}
```

检索任务重试策略的状态

您可以使用和查看任务的重试状态[`ListJobRuns`]。[`DescribeJobRun`] APIs请求启用了重试策略配置的任务后，`ListJobRun` 和 `DescribeJobRun` 响应将在 `RetryPolicyExecution` 字段中包含重试策略的状态。此外，`DescribeJobRun` 响应将包含在任务的 `StartJobRun` 请求中输入的 `RetryPolicyConfiguration`。

示例响应

ListJobRuns response

```
{
  "jobRuns": [
    ...
    ...
    "retryPolicyExecution" : {
      "currentAttemptCount": 2
    }
    ...
    ...
  ]
}
```

DescribeJobRun response

```
{
```

```
  ...
  ...
  "retryPolicyConfiguration": {
    "maxAttempts": 5
   },
   "retryPolicyExecution" : {
    "currentAttemptCount": 2
  },
  ...
  ...
}
```

在禁用作业中的重试策略后，这些字段将不可见，如以下 重试策略配置值 中所述。

使用重试策略监控作业

启用重试策略后，会为创建的每个任务驱动程序生成一个 CloudWatch 事件。要订阅这些事件，请使用以下命令设置 CloudWatch 事件规则：

```
aws events put-rule \
--name cwe-test \
--event-pattern '{"detail-type": ["EMR Job Run New Driver Attempt"]}'
```

该事件将返回有关作业驱动程序的 `newDriverPodName`、`newDriverCreatedAt` 时间戳、`previousDriverFailureMessage` 和 `currentAttemptCount` 的信息。如果禁用重试策略，则不会创建这些事件。

有关如何通过 CloudWatch 事件监控作业的更多信息，请参阅通过 Amazon Events CloudWatch 监控作业。

查找驱动程序和执行程序的日志

驱动程序 Pod 名称遵循格式 `spark-<job id>-driver-<random-suffix>`。相同的 `random-suffix` 会添加到驱动程序生成的执行程序 Pod 名称中。使用此 `random-suffix` 时，您可以找到驱动程序及其关联执行程序的日志。只有为任务启用了重试策略，`random-suffix` 才会存在；否则，`random-suffix` 不会存在。

有关如何使用监控配置来配置任务以用于日志记录的更多信息，请参阅 运行 Spark 应用程序。

使用 Spark 事件日志轮替

借助 Amazon EMR 6.3.0 及更高版本，您可以启用 Amazon EMR on EKS 的 Spark 事件日志轮换功能。此功能根据配置的时间间隔轮替文件，并删除最旧的事件日志文件，而不是生成单个事件日志文件。

轮替 Spark 事件日志，以助您避免因长时间运行或串流任务而生成的大型 Spark 事件日志文件出现潜在问题。例如，启动一个长时间运行的 Spark 任务，并且其启用了使用 `persistentAppUI` 参数的事件日志。Spark 驱动程序会生成一个事件日志文件。如果任务运行数小时或数天，并且 Kubernetes 节点上的磁盘空间有限，则事件日志文件可以占用所有可用磁盘空间。启用 Spark 事件日志轮替功能，可通过将日志文件拆分为多个文件，并删除最旧的文件来解决问题。

> **Note**
>
> 此功能只能在 Amazon EMR on EKS 上使用。在亚马逊上运行的亚马逊 EMR EC2 不支持 Spark 事件日志轮换。

要打开 Spark 事件日志轮替功能，请配置以下 Spark 参数：

- `spark.eventLog.rotation.enabled`：启用日志轮替功能。默认情况下，Spark 配置文件中会禁用它。将其设置为 true 以启用此功能。
- `spark.eventLog.rotation.interval`：指定日志轮替的时间间隔。最小值为 60 秒。默认值为 300 秒。
- `spark.eventLog.rotation.minFileSize`：指定轮替日志文件的最小文件大小。默认值最少为 1MB。
- `spark.eventLog.rotation.maxFilesToRetain`：指定清理期间要保留的轮替日志文件的数量。有效范围为 1 到 10。默认值是 2。

您可以在 StartJobRun API 的 `sparkSubmitParameters` 部分制定这些参数，如以下示例所示。

```
"sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi --conf
 spark.eventLog.rotation.enabled=true --conf spark.eventLog.rotation.interval=300 --
conf spark.eventLog.rotation.minFileSize=1m --conf
 spark.eventLog.rotation.maxFilesToRetain=2"
```

使用 Spark 容器日志轮换

借助 Amazon EMR 6.11.0 及更高版本，您可以启用 Amazon EMR on EKS 的 Spark 容器日志轮换功能。此功能根据配置的轮换大小轮替文件，删除最旧的容器日志文件，而不是生成单个 `stdout` 或 `stderr` 日志文件。

轮替 Spark 容器日志，可以助您避免因长时间运行或串流任务生成大型 Spark 日志文件出现的潜在问题。例如，在您启动一个长时间运行的 Spark 任务后，Spark 驱动程序会生成一个容器日志文件。如果任务运行数小时或数天，并且 Kubernetes 节点上的磁盘空间有限，则容器日志文件可以占用所有可用磁盘空间。启用 Spark 容器日志轮换功能后，可将日志文件拆分为多个文件，并删除最旧的文件。

要打开 Spark 容器日志轮换功能，请配置以下 Spark 参数：

`containerLogRotationConfiguration`

在 `monitoringConfiguration` 中加入此参数可开启日志轮换功能。默认情况下它是禁用的。除了 `s3MonitoringConfiguration`，还必须使用 `containerLogRotationConfiguration`。

`rotationSize`

`rotationSize` 参数指定日志轮换的文件大小。可行值的范围从 2KB 到 2GB 不等。`rotationSize` 参数的数字单位部分以整数形式传递。由于不支持十进制值，您可以指定 1.5GB 的轮换大小，例如值 `1500MB`。

`maxFilesToKeep`

`maxFilesToKeep` 参数指定轮换后要在容器中保留的最大文件数。最小值为 1，最大值为 50。

您可以在 `StartJobRun` API 的 `monitoringConfiguration` 部分制定这些参数，如以下示例所示。在此示例中，当 `rotationSize = "10 MB"` 且 `maxFilesToKeep = 3` 时，Amazon EMR on EKS 会在 10MB 时轮换日志，生成新的日志文件，然后在日志文件数量达到 3 个时清除最早的日志文件。

```
{
  "name": "my-long-running-job",
  "virtualClusterId": "123456",
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "emr-6.11.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "entryPoint_location",
```

```
      "entryPointArguments": ["argument1", "argument2", ...],
       "sparkSubmitParameters": "--class main_class --conf spark.executor.instances=2
 --conf spark.executor.memory=2G --conf spark.executor.cores=2 --conf
 spark.driver.cores=1"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.driver.memory":"2G"
         }
      }
    ],
    "monitoringConfiguration": {
      "persistentAppUI": "ENABLED",
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "my_log_group",
        "logStreamNamePrefix": "log_stream_prefix"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://my_s3_log_location"
      },
      "containerLogRotationConfiguration": {
        "rotationSize":"10MB",
        "maxFilesToKeep":"3"
      }
    }
  }
}
```

要启动使用 Spark 容器日志轮换运行的任务，请在 `StartJobRun` 命令中包含您使用这些参数配置的 json 文件的路径。

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

使用垂直自动扩展功能处理 Amazon EMR Spark 任务

Amazon EMR on EKS 会自动调整内存和 CPU 资源，从而适应您为 Amazon EMR Spark 应用程序工作负载提供的需求。此功能可简化资源管理工作。

为了跟踪 Amazon EMR Spark 应用程序的实时和历史资源利用率，垂直自动扩展功能会利用 Kubernetes Vertical Pod Autoscaler（VPA）。垂直自动扩展功能使用 VPA 收集的数据来自动调整分配给 Spark 应用程序的内存和 CPU 资源。这种简化流程既能提高可靠性，又能优化成本。

主题

- 设置 Amazon EMR on EKS 的垂直自动扩展
- Amazon EMR on EKS 的垂直自动扩展入门
- 配置 Amazon EMR on EKS 的垂直自动扩展
- 监控 Amazon EMR on EKS 的垂直自动扩展
- 卸载 Amazon EMR on EKS 垂直自动扩展 Operator

设置 Amazon EMR on EKS 的垂直自动扩展

本主题旨在帮助您设置好 Amazon EKS 集群，以便通过垂直自动扩展功能提交 Amazon EMR Spark 任务。设置过程要求您确认或完成以下各节中的任务：

主题

- 先决条件
- 在 Amazon EKS 集群上安装 Operator Lifecycle Manager（OLM）
- 安装 Amazon EMR on EKS 垂直自动扩展 Operator

先决条件

在集群上安装垂直自动扩展 Kubernetes Operator 之前，请完成以下任务。跳过已完成的先决条件，转到下一个先决条件。

- 安装或更新到最新版本的 Amazon CLI — 如果您已经安装了 Amazon CLI，请确认您安装的是最新版本。
- 安装 kubectl – kubectl 是一个命令行工具，用于与 Kubernetes API 服务器进行通信。在 Amazon EKS 集群上安装并监控与垂直自动扩展相关的构件，需要用到 kubectl。
- 安装 Operator SDK – Amazon EMR on EKS 使用 Operator SDK 作为您在集群上安装的垂直自动扩展 Operator 生命周期的包管理器。
- 安装 Docker – 您需要访问 Docker CLI，才能验证并获取要安装在 Amazon EKS 集群上的垂直自动扩展相关的 Docker 映像。

- 安装 Kubernetes Metrics Server：必须先安装 Metrics Server，垂直容器组（pod）Autoscaler 才能从 Kubernetes API 服务器获取指标。
- 开始使用 Amazon EKS - eksctl（1.24 或更高版本）：Amazon EKS 1.24 及更高版本支持垂直自动扩缩功能。创建集群后，请注册集群用于 Amazon EMR。
- 选择 Amazon EMR 基础映像 URI（6.10.0 或更高版本）– Amazon EMR 6.10.0 及更高版本都支持垂直自动扩展功能。

在 Amazon EKS 集群上安装 Operator Lifecycle Manager（OLM）

使用 Operator SDK CLI 在要设置垂直自动扩展功能的 Amazon EMR on EKS 集群上安装 Operator Lifecycle Manager（OLM），示例如下。设置完成后，您可以使用 OLM 来安装并管理 Amazon EMR 垂直自动扩展 Operator 的生命周期。

```
operator-sdk olm install
```

要验证安装情况，请运行 `olm status` 命令：

```
operator-sdk olm status
```

如果安装成功，验证命令返回类似如下示例输出的结果：

```
INFO[0007] Successfully got OLM status for version X.XX
```

如果安装失败，请参阅 对 Amazon EMR on EKS 垂直自动扩展进行问题排查。

安装 Amazon EMR on EKS 垂直自动扩展 Operator

按照以下步骤在 Amazon EKS 集群上安装垂直自动扩展 Operator：

1. 设置以下要用来完成安装的环境变量：
    - **`$REGION`** 指向集群的 Amazon Web Services 区域 。例如，`us-west-2`。
    - **`$ACCOUNT_ID`** 指向所在地区的 Amazon ECR 账户 ID。有关更多信息，请参阅 按区域划分的 Amazon ECR 注册账户。
    - **`$RELEASE`** 指向要用于集群的 Amazon EMR 版本。使用垂直自动扩展功能时，必须使用 Amazon EMR 6.10.0 或更高版本。
2. 接下来，为 Operator 获取 Amazon ECR 注册表的身份验证令牌。

```
aws ecr get-login-password \
 --region region-id | docker login \
 --username AWS \
 --password-stdin $ACCOUNT_ID.dkr.ecr.region-id.amazonaws.com
```

3. 使用以下命令在 Amazon EMR on EKS 上安装垂直自动扩展 Operator：

   ```
   ECR_URL=$ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com && \
   REPO_DEST=dynamic-sizing-k8s-operator-olm-bundle && \
   BUNDLE_IMG=emr-$RELEASE-dynamic-sizing-k8s-operator && \
   operator-sdk run bundle \
   $ECR_URL/$REPO_DEST/$BUNDLE_IMG\:latest
   ```

   这会在 Amazon EKS 集群的默认命名空间中创建发行版垂直自动扩展 Operator。使用此命令在不同命名空间中安装：

   ```
   operator-sdk run bundle \
   $ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/dynamic-sizing-k8s-operator-olm-bundle/
   emr-$RELEASE-dynamic-sizing-k8s-operator:latest \
   -n operator-namespace
   ```

   > **Note**
   >
   > 如果您指定的命名空间不存在，OLM 不会安装该 Operator。有关更多信息，请参阅 未找到 Kubernetes 命名空间。

4. 使用 kubectl Kubernetes 命令行工具验证是否成功安装了 Operator。

   ```
   kubectl get csv -n operator-namespace
   ```

   `kubectl` 命令应返回新部署的垂直 Autoscaler Operator，且阶段状态为成功。如在安装或设置时遇到问题，请参阅 对 Amazon EMR on EKS 垂直自动扩展进行问题排查。

Amazon EMR on EKS 的垂直自动扩展入门

如果您希望自动调整内存和 CPU 资源以适应 Amazon EMR Spark 应用程序工作负载，则对 Amazon EMR on EKS 使用垂直自动扩缩功能。有关更多信息，请参阅使用垂直自动扩缩功能处理 Amazon EMR Spark 作业。

使用垂直自动扩展功能提交 Spark 任务

当您通过 StartJobRunAPI 提交任务时，请将以下两个配置添加到驱动程序中，以便您的 Spark 作业开启垂直自动缩放：

```
"spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/
dynamic.sizing":"true",
"spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/
dynamic.sizing.signature":"YOUR_JOB_SIGNATURE"
```

在上面的代码中，第一行启用了垂直自动扩展功能。下一行是必需的签名配置，可让您为任务选择签名。

有关这些配置和可接受的参数值的更多信息，请参阅 配置 Amazon EMR on EKS 的垂直自动扩展。默认情况下，任务在垂直自动扩展的仅限监控关闭模式下提交。这种监控状态可让您在不执行自动扩展的情况下计算并查看资源建议。有关更多信息，请参阅 垂直自动扩展模式。

以下示例说明如何使用垂直自动扩展来完成 `start-job-run` 示例命令：

```
aws emr-containers start-job-run \
--virtual-cluster-id $VIRTUAL_CLUSTER_ID \
--name $JOB_NAME \
--execution-role-arn $EMR_ROLE_ARN \
--release-label emr-6.10.0-latest \
--job-driver '{
  "sparkSubmitJobDriver": {
     "entryPoint": "local:///usr/lib/spark/examples/src/main/python/pi.py"
   }
 }' \
--configuration-overrides '{
    "applicationConfiguration": [{
        "classification": "spark-defaults",
        "properties": {
          "spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/
dynamic.sizing": "true",
          "spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/
dynamic.sizing.signature": "test-signature"
        }
    }]
  }'
```

验证垂直自动扩展功能

要验证已提交任务的垂直自动扩展功能是否正常工作，请使用 kubectl 获取 `verticalpodautoscaler` 自定义资源并查看扩展建议。例如，以下命令会查询 使用垂直自动扩展功能提交 Spark 任务 部分中有关示例任务的建议：

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

此查询的输出应类似以下内容：

```
NAME                                                          MODE   CPU         MEM
 PROVIDED   AGE
ds-jceyefkxnhrvdzw6djum3naf2abm6o63a6dvjkkedqtkhlrf25eq-vpa   Off    3304504865  True
         87m
```

如果输出内容与此不相似或包含错误代码，请参阅 对 Amazon EMR on EKS 垂直自动扩展进行问题排查，了解有助于解决问题的操作步骤。

配置 Amazon EMR on EKS 的垂直自动扩展

当你通过 API 提交 Amazon EMR Spark 任务时，你可以配置垂直自动扩展。StartJobRun按照 使用垂直自动扩展功能提交 Spark 任务 中的示例所示，在 Spark 驱动程序 Pod 上设置与自动扩展相关的配置参数。

Amazon EMR on EKS 垂直自动扩展 Operator 会监听具有自动扩展功能的驱动程序 Pod，然后使用驱动程序 Pod 上的设置来设定与 Kubernetes Vertical Pod Autoscaler（VPA）的集成。这有助于对 Spark 执行程序 Pod 进行资源跟踪和自动扩展。

以下各节旨在介绍在为 Amazon EKS 集群配置垂直自动扩展时可以使用的参数。

> **Note**
>
> 将功能切换参数配置为标签，然后在 Spark 驱动程序 Pod 上将其余参数配置为注释。自动扩展参数属于 `emr-containers.amazonaws.com/` 域并且带有 `dynamic.sizing` 前缀。

必需参数

提交任务时，必须在 Spark 任务驱动程序中包含以下两个参数：

键	描述	接受的值	默认值	类型	Spark 参数[1]
`dynamic.sizing`	功能切换	`true`, `false`	未设置	label	`spark.kubernetes.driver.label.emr-containers.amazonaws.com/dynamic.sizing`
`dynamic.sizing.signature`	任务签名	string	未设置	注释	`spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.signature`

[1] 在 `StartJobRun` API 中将此参数用作 `SparkSubmitParameter` 或 `ConfigurationOverride`。

- **`dynamic.sizing`** – 您可以使用 `dynamic.sizing` 标签来打开或关闭垂直自动扩展功能。要开启垂直自动扩展功能，请在 Spark 驱动程序 Pod 上将 `dynamic.sizing` 设置为 `true`。如果省略此标签或将其设置为 `true` 之外的任何值，则垂直自动扩展功能将关闭。
- **`dynamic.sizing.signature`** – 在驱动程序 Pod 上使用 `dynamic.sizing.signature` 注释设置任务签名。垂直自动扩展功能可汇总不同的 Amazon EMR Spark 任务运行的资源使用数据，从中得出资源建议。您要提供唯一标识符将任务关联在一起。

> **Note**
>
> 如果任务以固定间隔（例如每天或每周）重复出现，任务的每个新实例的任务签名都应保持不变。这可确保垂直自动扩展功能可以计算并汇总不同任务运行中的建议。

[1] 在 `StartJobRun` API 中将此参数用作 `SparkSubmitParameter` 或 `ConfigurationOverride`。

可选参数

垂直自动扩展功能还支持以下可选参数。将这些参数设置为驱动程序 Pod 上的注释。

键	描述	接受的值	默认值	类型	Spark 参数[1]
`dynamic.sizing.mode`	垂直自动扩展模式	`Off`, `Initial`, `Auto`	`Off`	注释	`spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.mode`
`dynamic.sizing.scale.memory`	启用内存扩展	*`true`*, *`false`*	`true`	注释	`spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.sc`

键	描述	接受的值	默认值	类型	Spark 参数[1]
					ale.memory
dynamic.sizing.scale.cpu	开启或关闭 CPU 扩展	*true*, *false*	false	注释	spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.scale.cpu
dynamic.sizing.scale.memory.min	内存扩展的最小限制	字符串，K8s 资源数量，示例：1G	未设置	注释	spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.scale.memory.min

键	描述	接受的值	默认值	类型	Spark 参数[1]
dynamic.sizing.scale.memory.max	内存扩展的最大限制	字符串，K8s 资源数量，示例：4G	未设置	注释	spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.scale.memory.max
dynamic.sizing.scale.cpu.min	CPU 扩展的最小限制	字符串，K8s 资源数量，示例：1	未设置	注释	spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.scale.cpu.min

键	描述	接受的值	默认值	类型	Spark 参数[1]
dynamic.sizing.scale.cpu.max	CPU 扩展的最大限制	字符串，K8s 资源数量，示例：2	未设置	注释	spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.scale.cpu.max

垂直自动扩展模式

mode 参数映射到 VPA 支持的不同自动扩展模式。使用驱动程序 Pod 上的 dynamic.sizing.mode 注释来设置模式。此参数支持下列值：

- 关闭 – 一种试运行模式，支持在其中监控建议，但不支持执行自动扩展。这是垂直自动扩展功能的默认模式。在这种模式下，关联的垂直 Pod Autoscaler 资源会对建议进行计算，您可以通过 kubectl、Prometheus 和 Grafana 等工具监控建议。
- 初始 – 在此模式下，如果根据任务的历史运行情况（例如重复任务）提供建议，VPA 会在任务开始时自动扩展资源。
- 自动 – 在此模式下，VPA 会驱逐 Spark 执行程序 Pod，并在 Spark 驱动程序 Pod 重启时使用推荐的资源设置自动扩展这些 Pod。有时，VPA 会驱逐正在运行的 Spark 执行程序 Pod；因此，重试中断的执行程序可能会导致额外延迟。

资源扩展

在设置垂直自动扩展功能时，您可以选择是否扩展 CPU 和内存资源。将 dynamic.sizing.scale.cpu 和 dynamic.sizing.scale.memory 注释设置为 true 或 false。默认情况下，CPU 扩展设置为 false，内存扩展设置为 true。

资源最小值和最大值（边界）

您也可以选择设置 CPU 和内存资源的边界。启用自动扩展功能时，请使用 `dynamic.sizing.[memory/cpu].[min/max]` 注释为这些资源选择最小值和最大值。默认情况下，资源没有限制。将注释设置为表示 Kubernetes 资源数量的字符串值。例如，将 `dynamic.sizing.memory.max` 设置为 4G 来表示 4GB。

监控 Amazon EMR on EKS 的垂直自动扩展

您可以使用 kubectl Kubernetes 命令行工具列出集群上与垂直自动扩缩相关的有效建议。您还可以查看跟踪的任务签名，清除与签名关联的所有不需要的资源。

列出集群的垂直自动扩展建议

使用 kubectl 获取 `verticalpodautoscaler` 资源，查看当前状态和建议。以下示例查询返回 Amazon EKS 集群上的所有活跃资源。

```
kubectl get verticalpodautoscalers \
-o custom-columns="NAME:.metadata.name,"\
"SIGNATURE:.metadata.labels.emr-containers\.amazonaws\.com/dynamic\.sizing
\.signature,"\
"MODE:.spec.updatePolicy.updateMode,"\
"MEM:.status.recommendation.containerRecommendations[0].target.memory" \
--all-namespaces
```

此查询的输出类似以下内容：

```
NAME                  SIGNATURE                  MODE      MEM
ds-example-id-1-vpa   job-signature-1            Off       none
ds-example-id-2-vpa   job-signature-2            Initial   12936384283
```

查询并删除集群的垂直自动扩展建议

删除 Amazon EMR 垂直自动扩展任务运行资源时，也会自动删除跟踪和存储建议的关联 VPA 对象。

以下示例使用 kubectl 清除由签名标识的任务的建议：

```
kubectl delete jobrun -n emr -l=emr-containers\.amazonaws\.com/dynamic\.sizing
\.signature=integ-test
```

```
jobrun.dynamicsizing.emr.services.k8s.aws "ds-job-signature" deleted
```

如果不知道具体的任务签名，或者想要清除集群上的所有资源，则可以在命令中使用 `--all` 或 `--all-namespaces` 代替唯一的任务 ID，示例如下：

```
kubectl delete jobruns --all --all-namespaces
jobrun.dynamicsizing.emr.services.k8s.aws "ds-example-id" deleted
```

卸载 Amazon EMR on EKS 垂直自动扩展 Operator

如果您想从 Amazon EKS 集群中移除垂直自动扩展 Operator，请将 `cleanup` 命令与 Operator SDK CLI 配合使用，示例如下。这也会删除与 Operator 一起安装的上游依赖项，例如 Vertical Pod Autoscaler。

```
operator-sdk cleanup emr-dynamic-sizing
```

删除 Operator 后，如果集群上有正在运行的任务，这些任务会在不进行垂直自动扩展的情况下继续运行。如果在删除 Operator 后在集群上提交任务，Amazon EMR on EKS 将忽略您在配置期间可能定义的任何与垂直自动扩展相关的参数。

在 Amazon EMR on EKS 上运行交互式工作负载

交互式端点是将 Amazon EMR Studio 连接到 Amazon EMR on EKS 的网关，以便您可以运行交互式工作负载。您可以将交互式端点与 EMR Studio 结合使用，对 Amazon S3 和 Amazon DynamoDB 等数据存储中的数据集运行交互式分析。

使用案例

- 使用 EMR Studio IDE 体验创建 ETL 脚本。IDE 提取本地数据并在转换后将其存储在 Amazon S3 中，以供后续分析。
- 使用 Notebook 浏览数据集并训练机器学习模型来检测数据集中的异常。
- 创建用于为业务控制面板等分析应用程序生成每日报告的脚本。

主题

- 交互式端点概述
- 在 Amazon EMR on EKS 上创建交互式端点的先决条件
- 为虚拟集群创建交互式端点
- 配置交互式端点的设置
- 监控交互式端点
- 使用自托管式 Jupyter notebook
- 使用 CLI 命令获取有关交互式端点的信息

交互式端点概述

交互式端点向 Amazon EMR Studio 等交互式客户端提供连接到 Amazon EMR on EKS 集群以运行交互式工作负载的功能。交互式端点由 Jupyter Enterprise Gateway 提供支持，该网关提供交互式客户端所需的远程内核生命周期管理功能。内核是与特定于语言的进程，通过与基于 Jupyter 的 Amazon EMR Studio 客户端进行交互来运行交互式工作负载。

交互式端点支持以下内核：

- Python 3
- PySpark 在 Kubernetes 上
- Apache Spark with Scala

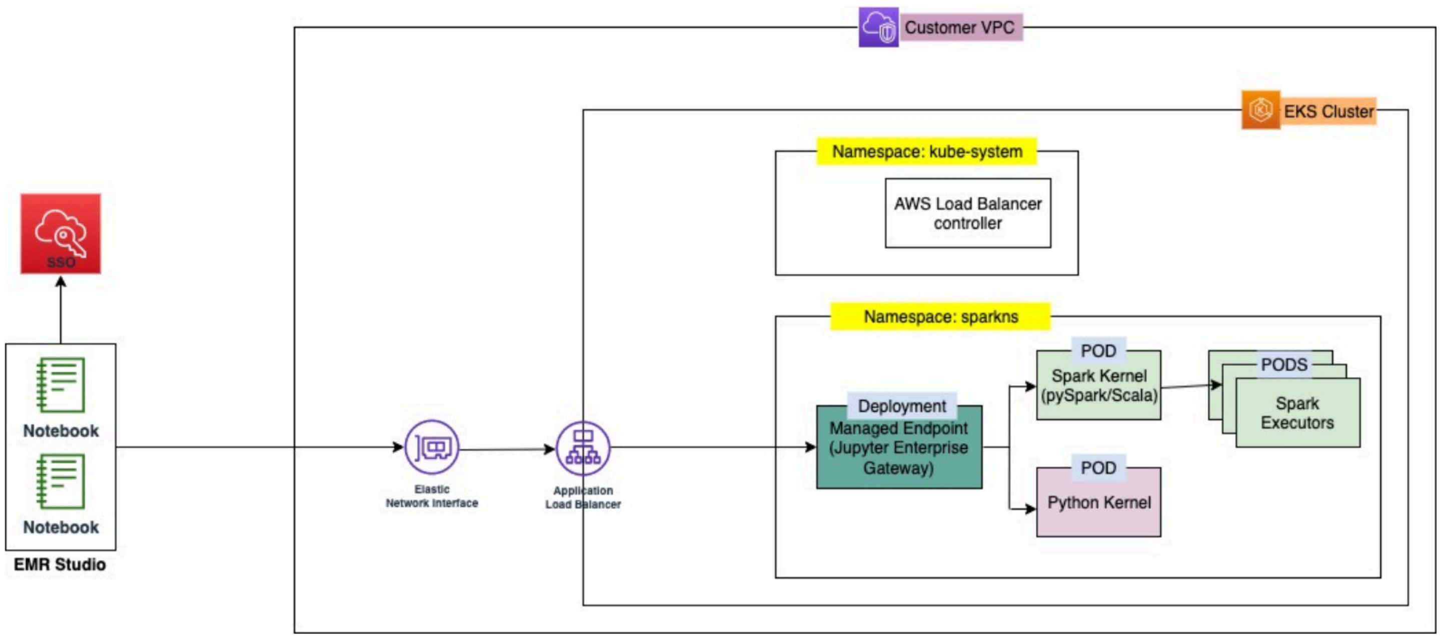
> **Note**
>
> Amazon EMR on EKS 定价适用于交互式端点和内核。有关更多信息，请参阅 Amazon EMR on EKS 定价页面。

EMR Studio 需要以下实体才能与 Amazon EMR on EKS 建立连接。

- Amazon EMR on EKS 虚拟集群 – 虚拟集群是注册 Amazon EMR 使用的 Kubernetes 命名空间。Amazon EMR 使用虚拟集群运行任务和主机终端节点。您可以使用同一个物理集群支持多个虚拟集群。不过，每个虚拟集群都映射到 Amazon EKS 集群上的命名空间。虚拟集群不会创建任何可增加您账单的活动资源，以及需要在服务之外进行生命周期管理的活动资源。
- Amazon EMR on EKS 交互式端点 – 交互式端点是 HTTPS 端点，EMR Studio 用户可以将 Workspace 连接到该端点。您只能通过 EMR Studio 访问 HTTPS 端点，并在 Amazon EKS 集群的 Amazon Virtual Private Cloud（Amazon VPC）的私有子网中创建。

  Python PySpark、和 Spark Scala 内核使用你的 Amazon EMR on EKS 任务执行角色中定义的权限来调用其他内核。 Amazon Web Services 服务连接到交互式端点的所有内核和用户都使用创建端点时指定的角色。我们建议您为不同的用户创建单独的终端节点，并且这些用户具有不同的 Amazon Identity and Access Management (IAM) 角色。
- Amazon A@@ pplication Load Balancer 控制器 — Amazon 应用程序负载均衡器控制器管理 Amazon EKS Kubernetes 集群的弹性负载平衡。在创建 Kubernetes 入口资源时，控制器会预置应用程序负载均衡器（ALB）。ALB 会在 Amazon EKS 集群之外但在同一 Amazon VPC 内公开 Kubernetes 服务，例如交互式端点。创建交互式端点时还会部署入口资源，该资源通过 ALB 公开交互式端点，以供交互式客户端连接。您只需要为每个 Amazon EKS 集群安装一个 App Amazon lication Load Balancer 控制器。

下图描述了 Amazon EMR on EKS 中的交互式端点架构。Amazon EKS 集群包括运行分析工作负载的计算和交互式端点。应用程序负载均衡器控制器在 `kube-system` 命名空间中运行；工作负载和交互式端点在创建虚拟集群时指定的命名空间中运行。创建交互式端点后，Amazon EMR on EKS 控制面板会在 Amazon EKS 集群中创建交互式端点部署。此外，应用程序负载均衡器入口的实例由负载均衡器控制器创建。 Amazon 应用程序负载均衡器为 EMR Studio 等客户端提供外部接口，以便连接到 Amazon EMR 集群并运行交互式工作负载。



在 Amazon EMR on EKS 上创建交互式端点的先决条件

此部分介绍设置交互式端点的先决条件，EMR Studio 将使用该端点连接到 Amazon EMR on EKS 集群并运行交互式工作负载。

Amazon CLI

按照安装或更新到最新版本中的 Amazon CLI步骤安装最新版本的 Amazon Command Line Interface (Amazon CLI)。

安装 eksctl

要安装最新版本的 eksctl，请按照安装 kubectl 中的步骤操作。如果在 Amazon EKS 集群中使用 Kubernetes 版本 1.22 或更高版本，则请使用大于 0.117.0 的 eksctl 版本。

Amazon EKS 集群

创建 Amazon EKS 集群。将集群注册为 Amazon EMR on EKS 的虚拟集群。以下是此集群的要求和注意事项：

- 集群必须与您的 EMR Studio 位于同一 Amazon Virtual Private Cloud（VPC）中。
- 集群必须至少有一个私有子网，才能激活交互式端点、链接基于 Git 的存储库以及在私有模式下启动应用程序负载均衡器。

- 您的 EMR Studio 和用于注册虚拟集群的 Amazon EKS 集群之间必须至少有一个私有子网。这可确保您的交互式端点作为选项显示在 Studio Workspaces 中，并激活从 Studio 到应用程序负载均衡器的连接。

  连接 Studio 和 Amazon EKS 集群，有两种方法可供选择：

  - 创建 Amazon EKS 集群并将其关联到属于 EMR Studio 的子网。
  - 或者，创建 EMR Studio 并指定 Amazon EKS 集群的私有子网。
- 亚马逊 EKS 优化 AMIs 的 ARM Amazon Linux 不支持 EKS 交互式终端节点上的亚马逊 EMR。
- 交互式端点可与运行 Kubernetes 1.30 或更低版本的 Amazon EKS 集群配合使用。
- 仅支持 Amazon EKS 托管节点组。

授予 Amazon EMR on EKS 对集群的访问权限

使用 Grant Cluster Access for Amazon EMR on EKS 中的步骤，授予 Amazon EMR on EKS 对集群中特定命名空间的访问权限。

在 Amazon EKS 集群上激活 IRSA

要在 Amazon EKS 集群上激活服务账户的 IAM 角色（IRSA），请按照启用服务账户的 IAM 角色（IRSA）中的步骤操作。

创建 IAM 任务执行角色

您必须创建一个 IAM 角色，才能在 Amazon EMR on EKS 交互式端点上运行工作负载。我们在本文档中将此 IAM 角色称为任务执行角色。此 IAM 角色既会分配给交互式端点容器，也会分配给您向 EMR Studio 提交任务时创建的实际执行容器。您将需要 Amazon EMR on EKS 任务执行角色的 Amazon 资源名称（ARN）。为此，请按以下两个步骤操作：

- 创建任务执行的 IAM 角色。
- 更新任务执行角色的信任策略。

授予用户访问 Amazon EMR on EKS 的权限

请求创建交互式终端节点的 IAM 实体（用户或角色）还必须具有以下 Amazon EC2 和`emr-containers`权限。按照 授予用户访问 Amazon EMR on EKS 的权限 中介绍的步骤授予这些权限，以允许 Amazon EMR on EKS 创建、管理和删除安全组，这些安全组会限制流向交互式端点负载均衡器的入站流量。

以下 `emr-containers` 权限允许用户执行基本的交互式端点操作：

```
"ec2:CreateSecurityGroup",
"ec2:DeleteSecurityGroup",
"ec2:AuthorizeSecurityGroupEgress",
"ec2:AuthorizeSecurityGroupIngress",
"ec2:RevokeSecurityGroupEgress",
"ec2:RevokeSecurityGroupIngress"

"emr-containers:CreateManagedEndpoint",
"emr-containers:ListManagedEndpoints",
"emr-containers:DescribeManagedEndpoint",
"emr-containers:DeleteManagedEndpoint"
```

通过 Amazon EMR 注册 Amazon EKS 集群

设置虚拟集群并将其映射到您要运行任务的 Amazon EKS 集群中的命名空间。对于 Amazon Fargate 仅限集群，请为 EKS 虚拟集群上的 Amazon EMR 和 Fargate 配置文件使用相同的命名空间。

有关设置 Amazon EMR on EKS 虚拟集群的信息，请参阅 通过 Amazon EMR 注册 Amazon EKS 集群。

将 Loa Amazon d Balancer 控制器部署到 Amazon EKS 集群

您的 Amazon EKS 集群需要 Amazon 应用程序负载均衡器。您只需为每个 Amazon EKS 集群设置一个应用程序负载均衡器控制器。有关设置 App Amazon lication Load Balancer 控制器的信息，请参阅 Amazon EKS 用户指南中的安装 Amazon 负载均衡器控制器插件。

为虚拟集群创建交互式端点

本主题介绍了使用 Amazon 命令行界面 (Amazon CLI) 创建交互式端点的几种方法，并包括有关可用配置参数的详细信息。

使用 `create-managed-endpoint` 命令创建交互式端点

指定 `create-managed-endpoint` 命令中的参数，如下所示。Amazon EMR on EKS 支持使用 Amazon EMR 版本 6.7.0 及更高版本创建交互式端点。

```
aws emr-containers create-managed-endpoint \
--type JUPYTER_ENTERPRISE_GATEWAY \
```

```
--virtual-cluster-id 1234567890abcdef0xxxxxxxx \
--name example-endpoint-name \
--execution-role-arn arn:aws:iam::444455556666:role/JobExecutionRole \
--release-label emr-6.9.0-latest \
--configuration-overrides '{
    "applicationConfiguration": [{
        "classification": "spark-defaults",
        "properties": {
            "spark.driver.memory": "2G"
        }
    }],
    "monitoringConfiguration": {
        "cloudWatchMonitoringConfiguration": {
            "logGroupName": "log_group_name",
            "logStreamNamePrefix": "log_stream_prefix"
        },
        "persistentAppUI": "ENABLED",
        "s3MonitoringConfiguration": {
            "logUri": "s3://my_s3_log_location"
        }
    }
}'
```

有关更多信息，请参阅 用于创建交互式端点的参数。

使用 JSON 文件中的指定参数创建交互式端点

1. 创建一个 `create-managed-endpoint-request.json` 文件并指定端点所需的参数，如下面的 JSON 文件所示：

```
{
    "name": "MY_TEST_ENDPOINT",
    "virtualClusterId": "MY_CLUSTER_ID",
    "type": "JUPYTER_ENTERPRISE_GATEWAY",
    "releaseLabel": "emr-6.9.0-latest",
    "executionRoleArn": "arn:aws:iam::444455556666:role/JobExecutionRole",
    "configurationOverrides":
    {
        "applicationConfiguration":
        [
            {
                "classification": "spark-defaults",
                "properties":
```

```
                {
                    "spark.driver.memory": "8G"
                }
            }
        ],
        "monitoringConfiguration":
        {
            "persistentAppUI": "ENABLED",
            "cloudWatchMonitoringConfiguration":
            {
                "logGroupName": "my_log_group",
                "logStreamNamePrefix": "log_stream_prefix"
            },
            "s3MonitoringConfiguration":
            {
                "logUri": "s3://my_s3_log_location"
            }
        }
    }
}
```

2. 使用 `create-managed-endpoint` 命令和存储在本地或 Amazon S3 上的 `create-managed-endpoint-request.json` 文件路径。

```
aws emr-containers create-managed-endpoint \
--cli-input-json  file://./create-managed-endpoint-request.json --region AWS-Region
```

创建交互式端点输出

您应该参阅终端中的以下输出。输出包括新的交互式端点的名称和标识符：

```
{
    "id": "1234567890abcdef0",
    "name": "example-endpoint-name",
    "arn": "arn:aws:emr-containers:us-west-2:111122223333:/
virtualclusters/444455556666/endpoints/444455556666",
    "virtualClusterId": "111122223333xxxxxxxx"
}
```

运行 `aws emr-containers create-managed-endpoint` 会创建自签名证书，该证书允许 EMR Studio 与交互式端点服务器之间进行 HTTP 通信。

如果您运行 `create-managed-endpoint` 但尚未完成先决条件，Amazon EMR 将返回一条错误消息，其中包含您必须执行才能继续的操作。

用于创建交互式端点的参数

主题

- 交互式端点的必需参数
- 交互式端点的可选参数

交互式端点的必需参数

创建交互式端点时，必须指定以下参数：

`--type`

使用 `JUPYTER_ENTERPRISE_GATEWAY`。这是唯一支持的类型。

`--virtual-cluster-id`

您向 Amazon EMR on EKS 注册的虚拟集群的标识符。

`--name`

交互式端点的描述性名称，有助于 EMR Studio 用户从下拉列表中选择交互式端点。

`--execution-role-arn`

Amazon EMR on EKS 的 IAM 任务执行角色的 Amazon 资源名称（ARN），其作为先决条件的一部分创建。

`--release-label`

用于端点的 Amazon EMR 版本的发行版标签。例如 `emr-6.9.0-latest`。Amazon EMR on EKS 支持使用 Amazon EMR 版本 6.7.0 及更高版本的交互式端点。

交互式端点的可选参数

或者，创建交互式端点时，还能指定以下参数：

`--configuration-overrides`

要覆盖应用程序的默认配置，请提供配置对象。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。可用的配置分类因 Amazon EMR on EKS 发行版而异。有关 Amazon EMR on EKS 每个发行版可用的配置分类列表，请参阅 Amazon EMR on EKS 版本。除了为每个发行版列出的配置分类外，交互式端点还引入了额外的分类 `jeg-config`。有关更多信息，请参阅 Jupyter Enterprise Gateway（JEG）配置选项。

配置交互式端点的设置

本节包含一系列主题，这些主题涵盖交互式端点和容器组（pod）设置的各种配置。它们使您能够监控和排除故障，将日志信息发送到 Amazon S3 或 Amazon CloudWatch Logs，或者创建交互式终端节点，在其中指定自定义 pod 模板。

主题

- 监控 Spark 任务
- 使用交互式端点指定自定义 Pod 模板
- 将 JEG Pod 部署到节点组
- Jupyter Enterprise Gateway（JEG）配置选项
- 修改 PySpark 会话参数
- 使用交互式端点的自定义内核映像

监控 Spark 任务

为了监控故障并排除故障，请配置您的交互式终端节点，以便通过终端节点启动的任务可以向 Amazon S3、Amazon Logs 或两者发送 CloudWatch 日志信息。以下各节介绍如何将您使用 Amazon EMR on EKS 交互式端点启动的 Spark 任务的 Spark 应用程序日志发送到 Amazon S3。

配置 Amazon S3 日志的 IAM policy

任务执行角色的权限策略中必须包含以下权限，内核才能将日志数据发送到 Amazon S3。将 *amzn-s3-demo-destination-bucket* 替换为日志记录存储桶的名称。

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

```
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-destination-bucket",
                "arn:aws:s3:::amzn-s3-demo-logging-bucket/*",
            ]
        }
    ]
}
```

> Note
>
> Amazon EMR on EKS 也可以创建 S3 存储桶。如果 S3 存储桶不可用，则 IAM policy 应包含 `s3:CreateBucket` 权限。

在授予执行角色将日志发送到 S3 存储桶的权限后，日志数据将发送到以下 Amazon S3 位置。当 `s3MonitoringConfiguration` 在 `create-managed-endpoint` 请求的 `monitoringConfiguration` 部分中传递时，就会发生这种情况。

- 驱动程序日志：`logUri/virtual-cluster-id/endpoints/endpoint-id/containers/spark-application-id/spark-application-id-driver/(stderr.gz/stdout.gz)`
- 执行程序日志：`logUri/virtual-cluster-id/endpoints/endpoint-id/containers/spark-application-id/executor-pod-name-exec-<Number>/(stderr.gz/stdout.gz)`

> Note
>
> Amazon EMR on EKS 不会将端点日志上传到您的 S3 存储桶。

使用交互式端点指定自定义 Pod 模板

您可以创建交互式端点，在其中为驱动程序和执行程序指定自定义 Pod。Pod 模板是决定如何运行每个 Pod 的规范。您可以使用 Pod 模板文件定义 Spark 配置不支持的驱动程序或执行程序 Pod 的配置。Amazon EMR 发行版 6.3.0 及更高版本目前支持 Pod 模板。

有关 Pod 模板的更多信息，请参阅《Amazon EMR on EKS 开发指南》中的使用 Pod 模板。

以下示例展示如何使用 Pod 模板创建交互式端点：

```
aws emr-containers create-managed-endpoint \
    --type JUPYTER_ENTERPRISE_GATEWAY \
    --virtual-cluster-id virtual-cluster-id \
    --name example-endpoint-name \
    --execution-role-arn arn:aws:iam::aws-account-id:role/EKSClusterRole \
    --release-label emr-6.9.0-latest \
    --configuration-overrides '{
        "applicationConfiguration": [
        {
            "classification": "spark-defaults",
            "properties": {
                "spark.kubernetes.driver.podTemplateFile": "path/to/driver/
template.yaml",
                "spark.kubernetes.executor.podTemplateFile": "path/to/executor/
template.yaml"
            }
        }]
    }'
```

将 JEG Pod 部署到节点组

JEG（Jupyter Enterprise Gateway）Pod 放置功能允许您在特定节点组上部署交互式端点。使用此功能，您可以为交互式端点配置 `instance type` 等设置。

将 JEG Pod 关联到托管式节点组

以下配置属性允许您指定 Amazon EKS 集群上的托管式节点组的名称，JEG Pod 将在此集群中部署。

```
//payload
--configuration-overrides '{
      "applicationConfiguration": [
            {
                "classification": "endpoint-configuration",
                "properties": {
                    "managed-nodegroup-name": NodeGroupName
                }
            }
        ]
```

```
    }'
```

节点组必须将 Kubernetes 标签 `for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName` 附加到属于该节点组的所有节点。要列出节点组中所有带有此标签的节点，请使用以下命令：

```
kubectl get nodes --show-labels | grep for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName
```

如果上述命令的输出未返回属于托管式节点组的节点，则该节点组中没有附加了 `for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName` Kubernetes 标签的节点。在这种情况下，请按照以下步骤将此标签附加到节点组中的节点。

1. 使用以下命令向托管式节点组 *NodeGroupName* 中的所有节点添加 `for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName` Kubernetes 标签：

   ```
   kubectl label nodes --selector eks:nodegroup-name=NodeGroupName for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName
   ```

2. 使用以下命令验证节点是否已正确标记：

   ```
   kubectl get nodes --show-labels | grep for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName
   ```

托管式节点组必须与 Amazon EKS 集群的安全组关联，如果您使用 `eksctl` 创建集群和托管式节点组，通常会出现这种情况。您可以使用以下步骤在 Amazon 控制台中对此进行验证。

1. 转到 Amazon EKS 控制台中的集群。
2. 转到集群的“网络”选项卡，记下集群安全组。
3. 转到集群的“计算”选项卡，然后单击托管式节点组名称。
4. 在托管式节点组的详细信息选项卡下，验证您之前记下的集群安全组是否已列在安全组下。

如果托管式节点组未附加到 Amazon EKS 集群安全组，则需要将 `for-use-with-emr-containers-managed-endpoint-sg=ClusterName/NodeGroupName` 标签附加到节点组安全组。使用以下步骤附加此标签。

1. 前往 Amazon EC2 控制台，点击左侧导航窗格中的安全组。

2. 单击复选框，选择托管式节点组的安全组。
3. 在标签选项卡下，使用管理标签按钮添加标签 `for-use-with-emr-containers-managed-endpoint-sg=`*ClusterName*`/`*NodeGroupName*。

将 JEG Pod 关联到自托管式节点组

以下配置属性允许您指定 Amazon EKS 集群上的自托管式或非托管式节点组的名称，JEG Pod 将在此集群中部署。

```
//payload
--configuration-overrides '{
      "applicationConfiguration": [
            {
                "classification": "endpoint-configuration",
                "properties": {
                    "self-managed-nodegroup-name": NodeGroupName
                }
            }
        ]
    }'
```

节点组必须将 `for-use-with-emr-containers-managed-endpoint-ng=`*NodeGroupName* Kubernetes 标签附加到属于该节点组的所有节点。要列出节点组中所有带有此标签的节点，请使用以下命令：

```
kubectl get nodes --show-labels | grep for-use-with-emr-containers-managed-endpoint-
ng=NodeGroupName
```

如果上述命令的输出未返回属于自托管式节点组的节点，则该节点组中没有附加了 `for-use-with-emr-containers-managed-endpoint-ng=`*NodeGroupName* Kubernetes 标签的节点。在这种情况下，请按照以下步骤将此标签附加到节点组中的节点。

1. 如果您使用 `eksctl` 创建自托管式节点组，请使用以下命令将 `for-use-with-emr-containers-managed-endpoint-ng=`*NodeGroupName* Kubernetes 标签一次性添加到自托管式节点组 *NodeGroupName* 中的所有节点。

   ```
   kubectl label nodes --selector alpha.eksctl.io/nodegroup-name=NodeGroupName for-use-
   with-emr-containers-managed-endpoint-ng=NodeGroupName
   ```

如果您没有使用 `eksctl` 创建自托管式节点组，则需要将上述命令中的选择器替换为附加到该节点组的所有节点的其他 Kubernetes 标签。

2. 使用以下命令验证节点是否已正确标记：

```
kubectl get nodes --show-labels | grep for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName
```

自托管式节点组的安全组必须附加 `for-use-with-emr-containers-managed-endpoint-sg=ClusterName/NodeGroupName` 标签。使用以下步骤将标签附加到 Amazon Web Services Management Console的安全组。

1. 导航至 Amazon EC2 控制台。在左侧导航窗格中，选择安全组。
2. 选中自托管式节点组的安全组旁边的复选框。
3. 在标签选项卡下，使用管理标签按钮添加标签 `for-use-with-emr-containers-managed-endpoint-sg=ClusterName/NodeGroupName`。用适当的值替换 *ClusterName* 和 *NodeGroupName*。

将 JEG Pod 关联到具有按需型实例的托管式节点组

您还可以定义其他标签（称为 Kubernetes 标签选择器），以指定在给定节点或节点组上运行交互式端点的其他约束或限制。以下示例展示了如何使用按需 Amazon EC2 实例来创建 JEG 容器。

```
--configuration-overrides '{
      "applicationConfiguration": [
            {
                "classification": "endpoint-configuration",
                "properties": {
                    "managed-nodegroup-name": NodeGroupName,
                    "node-labels": "eks.amazonaws.com/capacityType:ON_DEMAND"
                }
            }
        ]
    }'
```

> **Note**
>
> 只能将 `node-labels` 属性与 `managed-nodegroup-name` 或 `self-managed-nodegroup-name` 属性一起使用。

Jupyter Enterprise Gateway（JEG）配置选项

Amazon EMR on EKS 使用 Jupyter Enterprise Gateway（JEG）启用交互式端点。在创建端点时，您可以为允许列出的 JEG 配置设置以下值。

- **`RemoteMappingKernelManager.cull_idle_timeout`** – 超时时间（以秒为单位的整数），在此时间之后，内核会被视为处于空闲状态并予以剔除。值等于或低于 `0` 便会停用剔除。若用户的网络连接不佳，短暂超时可能会导致内核被剔除。
- **`RemoteMappingKernelManager.cull_interval`** – 检查空闲内核是否超过剔除超时值的间隔（以秒为单位的整数）。

修改 PySpark 会话参数

从 EKS 版本 6.9.0 上的 Amazon EMR 开始，在 Amazon EMR Studio 中，你可以通过在 EMR 笔记本单元中执行神奇`%%configure`命令来调整与 PySpark 会话关联的 Spark 配置。

下例显示了一个示例负载，您可以使用该负载修改 Spark 驱动程序和执行程序的内存、内核和其他属性。对于 `conf` 设置，您可以配置 [Apache Spark 配置文档]中提及的任何 Spark 配置。

```
%%configure -f
{
  "driverMemory": "16G",
  "driverCores" 4,
  "executorMemory" : "32G"
  "executorCores": 2,
  "conf": {
     "spark.dynamicAllocation.maxExecutors" : 10,
     "spark.dynamicAllocation.minExecutors": 1
  }
}
```

下例显示了一个示例负载，您可以使用该负载向 Spark 运行时添加文件、PyFile 和 jar 依赖项。

```
%%configure -f
{
  "files": "s3://amzn-s3-demo-bucket-emr-eks/sample_file.txt",
  "pyFiles": : "path-to-python-files",
  "jars" : "path-to-jars
}
```

使用交互式端点的自定义内核映像

您可以为交互式端点自定义 Docker 映像并运行自定义的基本内核映像，以便确保在从 Amazon EMR Studio 运行交互式工作负载时应用程序拥有正确的依赖项。要创建交互式端点并将其连接到自定义 Docker 映像，请执行以下步骤。

> **Note**
>
> 您只能覆盖基本映像。您无法添加新的内核映像类型。

1. 创建并发布自定义 Docker 映像。基本镜像包含 Spark 运行时，以及随之运行的 Notebook 内核。要创建映像，您可以按照 如何自定义 Docker 镜像 中的步骤 1 到 4 操作。在步骤 1 中，Docker 文件中的基本映像 URI 必须使用 `notebook-spark` 来代替 `spark`。

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

   有关如何选择 Amazon Web Services 区域 和容器镜像标签的更多信息，请参阅如何选择基础映像 URI 的详细信息。

2. 创建可与自定义映像配合使用的交互式端点。

   a. 使用以下内容创建 JSON 文件 `custom-image-managed-endpoint.json`。此示例使用了 Amazon EMR 发行版 6.9.0。

      Example

      ```
      {
          "name": "endpoint-name",
          "virtualClusterId": "virtual-cluster-id",
          "type": "JUPYTER_ENTERPRISE_GATEWAY",
          "releaseLabel": "emr-6.9.0-latest",
      ```

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

b. 按如下示例所示，使用该 JSON 文件中指定的配置来创建交互式端点。有关更多信息，请参阅 使用 `create-managed-endpoint` 命令创建交互式端点。

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

3. 通过 EMR Studio 连接到交互式端点。有关更多信息和要完成的步骤，请参阅 Worksho Amazon p Studio 文档中 EKS 上的 Amazon EMR 部分中的 “从 Studio 连接”。

监控交互式端点

在 EKS 6.10 及更高版本上的 Amazon EMR 中，交互式终端节点会发出用于监控内核生命周期操作和故障排除的亚马逊 CloudWatch 指标。指标由交互式客户端（例如 EMR Studio 或自托管式 Jupyter notebook）触发。交互式端点支持的每个操作都有与之关联的指标。此操作作为每个指标的

维度进行建模，如下表所示。交互式终端节点发出的指标在您的账户中的自定义命名空间下可见。EMRContainers

指标	描述	单位
RequestCount	交互式端点处理的操作的累积请求数。	计数
RequestLatency	从请求到达交互式端点到交互式端点发送响应的时间。	毫秒
4 XXError	当操作请求在处理过程中导致 4xx 错误时发出。	计数
5XXError	当操作请求导致 5Xxx 服务端错误时发出。	计数
KernelLaunchSuccess	仅适用于该 CreateKernel 操作。它表示在此请求（包括此请求）之前内核启动成功的累积次数。	计数
KernelLaunchFailure	仅适用于该 CreateKernel 操作。它表示直到此请求（包括此请求）之前内核启动失败的累积次数。	计数

每个交互式端点指标都附加了以下维度：

- **`ManagedEndpointId`** – 交互式端点的标识符
- **`OperationName`** – 交互式客户端触发的操作

下表显示了 **`OperationName`** 维度的可能值：

`operationName`	操作描述
`CreateKernel`	请求交互式端点启动内核。

operationName	操作描述
ListKernels	请求交互式端点列出之前使用相同会话令牌启动的内核。
GetKernel	请求交互式端点获取有关之前启动的特定内核的详细信息。
ConnectKernel	请求交互式端点在 Notebook 客户端和内核之间建立连接。
ConfigureKernel	在 pyspark 内核上发布 %%configure magic request。
ListKernelSpecs	请求交互式端点列出可用的内核规范。
GetKernelSpec	请求交互式端点获取有关之前启动的内核的内核规范。
GetKernelSpecResource	请求交互式端点获取与之前启动的内核规范关联的特定资源。

示例

要访问在给定日期为交互式端点启动的内核总数，请执行以下操作：

1. 选择自定义命名空间：EMRContainers
2. 选择 ManagedEndpointId、OperationName – CreateKernel
3. RequestCount 指标以及统计数据 SUM 和周期 1 day 将提供过去 24 小时内发出的所有内核启动请求。
4. KernelLaunchSuccess 带有统计数据SUM和周期的指标1 day将提供过去 24 小时内发出的所有成功内核启动请求。

要访问给定日期交互式端点的内核故障数，请执行以下操作：

1. 选择自定义命名空间： EMRContainers
2. 选择 ManagedEndpointId、OperationName – CreateKernel

3. `KernelLaunchFailure` 指标以及统计数据 `SUM` 和周期 `1 day` 将提供过去 24 小时内发出的所有失败的内核启动请求。您也可以选择 `4XXError` 和 `5XXError` 指标来了解发生的内核启动失败类型。

使用自托管式 Jupyter notebook

您可以在亚马逊 EC2 实例上托管和管理 Jupyter 或 JupyterLab 笔记本电脑，也可以作为自托管 Jupyter 笔记本在自己的 Amazon EKS 集群上托管和管理。然后，您可以使用自托管式 Jupyter notebook 运行交互式工作负载。以下各节介绍在 Amazon EKS 集群上设置和部署自托管式 Jupyter notebook 的过程。

在 EKS 集群上创建自托管式 Jupyter notebook

- 创建安全组
- 创建 Amazon EMR on EKS 交互式端点
- 检索交互式端点的网关服务器 URL
- 检索身份验证令牌以连接到交互式端点
- 示例：部署 JupyterLab笔记本
- 删除自托管式 Jupyter notebook

创建安全组

在创建交互式终端节点并运行自托管的 Jupyter 或 JupyterLab 笔记本之前，必须创建一个安全组来控制笔记本和交互式终端节点之间的流量。要使用亚马逊 EC2 控制台或 Amazon EC2 SDK 创建安全组，请参阅亚马逊 EC2 用户指南中创建安全组中的步骤。您应该在要部署 Notebook 服务器的 VPC 中创建安全组。

要按照本指南中的示例进行操作，请使用与 Amazon EKS 集群相同的 VPC。如果您想将笔记本电脑托管在不同于 Amazon EKS 集群的 VPC 的 VPC 中，则可能需要在这两 VPCs者之间创建对等连接。有关在两者之间创建对等连接的步骤 VPCs，请参阅 Amazon VPC 入门指南中的创建 VPC 对等连接。

您需要安全组的 ID 才能在下一步中创建 Amazon EMR on EKS 交互式端点。

创建 Amazon EMR on EKS 交互式端点

为 Notebook 创建安全组后，使用 为虚拟集群创建交互式端点 中的步骤创建交互式端点。您必须提供在 创建安全组 中为 Notebook 创建的安全组 ID。

在以下配置覆盖设置中插入安全 ID 来代替：*your-notebook-security-group-id*

```
--configuration-overrides '{
    "applicationConfiguration": [
        {
            "classification": "endpoint-configuration",
            "properties": {
                "notebook-security-group-id": "your-notebook-security-group-id"
            }
        }
    ],
    "monitoringConfiguration": {
    ...'
```

检索交互式端点的网关服务器 URL

创建交互式端点后，使用 Amazon CLI中的 `describe-managed-endpoint` 命令检索网关服务器 URL。您需要此 URL 才能将 Notebook 连接到端点。网关服务器 URL 是私有端点。

```
aws emr-containers describe-managed-endpoint \
--region region \
--virtual-cluster-id virtualClusterId \
--id endpointId
```

最初，您的终端节点位于 CREATING州。几分钟后，它会过渡到 ACTIVE州。当终端节点为 ACTIVE，它已经准备好使用了。

记下 `aws emr-containers describe-managed-endpoint` 命令从活动端点返回的 `serverUrl` 属性。部署自托管 Jupyter 或笔记本时，需要此 URL 才能将笔记本电脑连接到终端节点。 JupyterLab

检索身份验证令牌以连接到交互式端点

要从 Jupyter 或 JupyterLab 笔记本连接到交互式端点，必须使用 API 生成会话令牌。`GetManagedEndpointSessionCredentials`此令牌作为连接到交互式端点服务器的身份验证证明。

下面的输出示例将更详细地解释以下命令。

```
aws emr-containers get-managed-endpoint-session-credentials \
--endpoint-identifier endpointArn \
```

```
--virtual-cluster-identifier virtualClusterArn \
--execution-role-arn executionRoleArn \
--credential-type "TOKEN" \
--duration-in-seconds durationInSeconds \
--region region
```

endpointArn

端点 ARN。您可以在 `describe-managed-endpoint` 调用中找到 ARN。

virtualClusterArn

虚拟集群的 ARN。

executionRoleArn

执行角色的 ARN。

durationInSeconds

令牌的有效持续时间（以秒为单位）。默认持续时间为 15 分钟 (`900`)，最大持续时间为 12 小时 (`43200`)。

region

与端点相同的区域。

输出应与以下示例类似。记下部署自托管 Jupyter 或笔记本时将使用的*session-token*值。JupyterLab

```
{
    "id": "credentialsId",
    "credentials": {
        "token": "session-token"
    },
    "expiresAt": "2022-07-05T17:49:38Z"
}
```

示例：部署 JupyterLab笔记本

完成上述步骤后，您可以尝试此示例过程，使用您的交互式终端节点将 JupyterLab 笔记本部署到 Amazon EKS 集群中。

1. 创建命名空间来运行 Notebook 服务器。
2. 在本地创建文件 `notebook.yaml`，包含以下内容。文件内容说明如下。

```
apiVersion: v1
kind: Pod
metadata:
  name: jupyter-notebook
  namespace: namespace
spec:
  containers:
  - name: minimal-notebook
    image: jupyter/all-spark-notebook:lab-3.1.4 # open source image
    ports:
    - containerPort: 8888
    command: ["start-notebook.sh"]
    args: ["--LabApp.token=''"]
    env:
    - name: JUPYTER_ENABLE_LAB
      value: "yes"
    - name: KERNEL_LAUNCH_TIMEOUT
      value: "400"
    - name: JUPYTER_GATEWAY_URL
      value: "serverUrl"
    - name: JUPYTER_GATEWAY_VALIDATE_CERT
      value: "false"
    - name: JUPYTER_GATEWAY_AUTH_TOKEN
      value: "session-token"
```

如果您要将 Jupyter notebook 部署到仅限 Fargate 的集群，请使用 `role` 标签标记 Jupyter Pod，如以下示例所示：

```
...
metadata:
  name: jupyter-notebook
  namespace: default
  labels:
    role: example-role-name-label
spec:
            ...
```

namespace

Notebook 部署所在的 Kubernetes 命名空间。

serverUrl

`describe-managed-endpoint` 命令在 检索交互式端点的网关服务器 URL 中返回的 `serverUrl` 属性。

session-token

`get-managed-endpoint-session-credentials` 命令在 检索身份验证令牌以连接到交互式端点 中返回的 `session-token` 属性。

KERNEL_LAUNCH_TIMEOUT

交互式端点等待内核进入的时间（以秒为单位）RUNNING州。将内核启动超时设置为适当的值（最长 400 秒），确保有足够的时间完成内核启动。

KERNEL_EXTRA_SPARK_OPTS

或者，您可以为 Spark 内核传递其他 Spark 配置。使用 Spark 配置属性的值设置此环境变量，如以下示例所示：

```
- name: KERNEL_EXTRA_SPARK_OPTS
  value: "--conf spark.driver.cores=2
          --conf spark.driver.memory=2G
          --conf spark.executor.instances=2
          --conf spark.executor.cores=2
          --conf spark.executor.memory=2G
          --conf spark.dynamicAllocation.enabled=true
          --conf spark.dynamicAllocation.shuffleTracking.enabled=true
          --conf spark.dynamicAllocation.minExecutors=1
          --conf spark.dynamicAllocation.maxExecutors=5
          --conf spark.dynamicAllocation.initialExecutors=1
          "
```

3. 将 Pod 规范部署到 Amazon EKS 集群：

```
kubectl apply -f notebook.yaml -n namespace
```

这将启动一台在 EKS 交互式终端节点上连接到你的 Amazon EMR 的最小 JupyterLab 笔记本电脑。等到 pod 变成 RUNNING。 您可以使用以下命令检查其状态：

```
kubectl get pod jupyter-notebook -n namespace
```

Pod 准备就绪时，`get pod` 命令会返回类似于以下内容的输出：

```
NAME              READY   STATUS    RESTARTS   AGE
jupyter-notebook  1/1     Running   0          46s
```

4. 将 Notebook 安全组附加到调度 Notebook 的节点。

   a. 首先，使用 `describe pod` 命令确定调度 `jupyter-notebook` Pod 的节点。

   ```
   kubectl describe pod jupyter-notebook -n namespace
   ```

   b. 在 https://console.aws.amazon.com/eks/home#/ 集群中打开 Amazon EKS 控制台。

   c. 导航到 Amazon EKS 集群的计算选项卡，然后选择 `describe pod` 命令标识的节点。选择节点的实例 ID。

   d. 在操作菜单中，选择安全 > 更改安全组，附加您在 [创建安全组](#) 中创建的安全组。

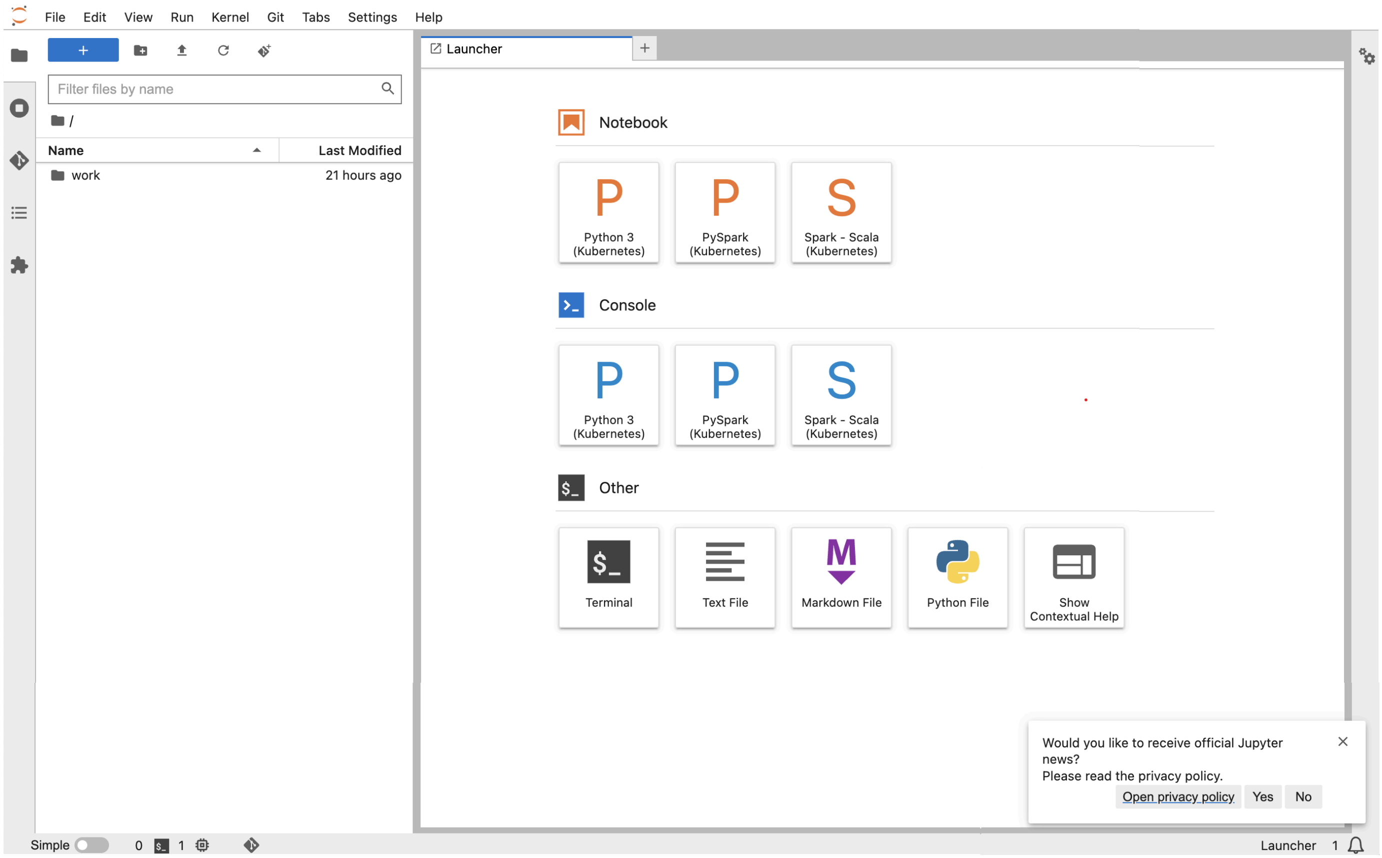
   e. 如果您要在上部署 Jupyter 笔记本窗格 Amazon Fargate，请使用角色[`SecurityGroupPolicy`](#)标签创建一个应用于 Jupyter 笔记本窗格：

   ```
   cat >my-security-group-policy.yaml <<EOF
   apiVersion: vpcresources.k8s.aws/v1beta1
   kind: SecurityGroupPolicy
   metadata:
     name: example-security-group-policy-name
     namespace: default
   spec:
     podSelector:
       matchLabels:
         role: example-role-name-label
     securityGroups:
       groupIds:
         - your-notebook-security-group-id
   EOF
   ```

5. 现在，进行端口转发，这样你就可以在本地访问 JupyterLab 接口：

```
kubectl port-forward jupyter-notebook 8888:8888 -n namespace
```

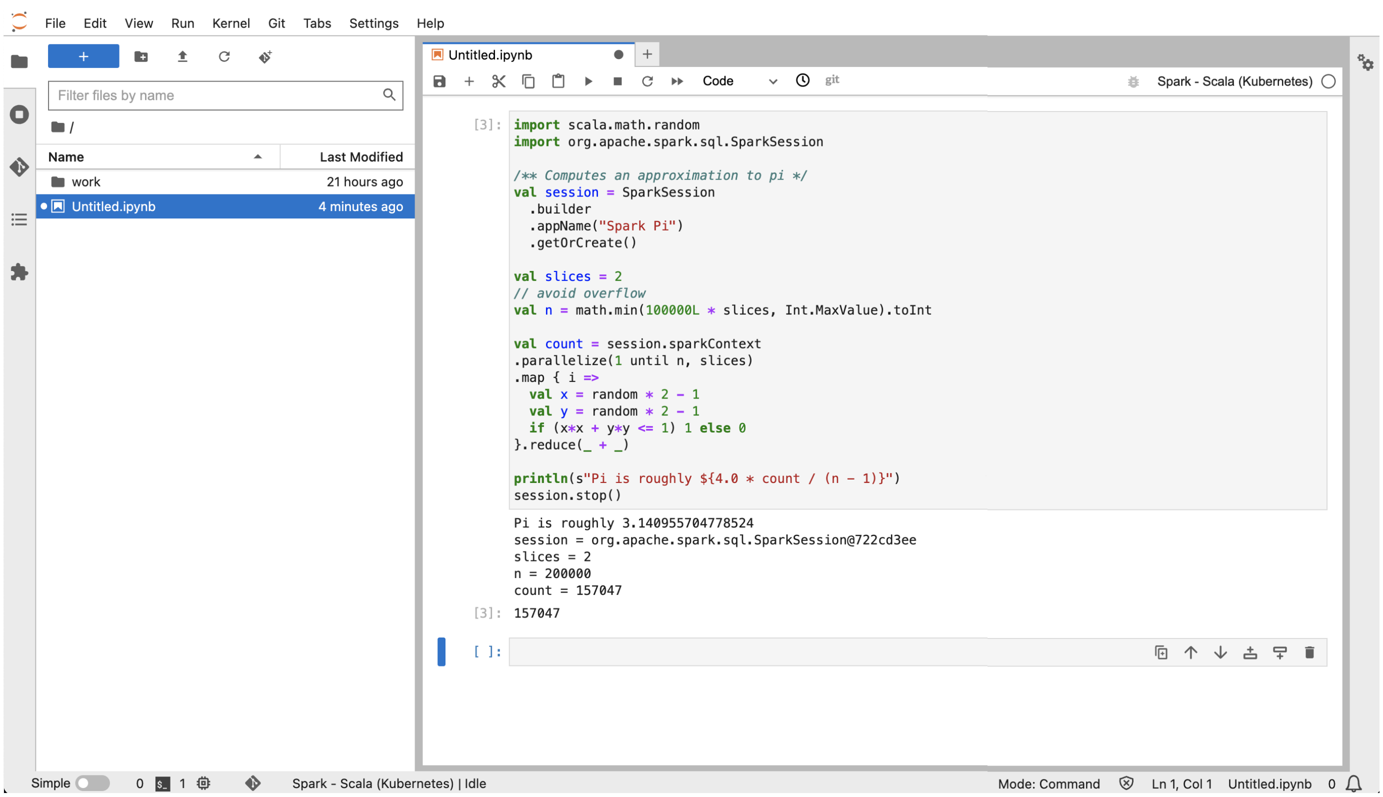
运行后，导航到您的本地浏览器并访问`localhost:8888`以查看 JupyterLab 界面：



6. 从 JupyterLab，创建一个新的 Scala 笔记本。下面是一个示例代码片段，您可以运行它来近似计算 Pi 的值：

```
import scala.math.random
import org.apache.spark.sql.SparkSession

/** Computes an approximation to pi */
val session = SparkSession
  .builder
  .appName("Spark Pi")
  .getOrCreate()

val slices = 2
// avoid overflow
val n = math.min(100000L * slices, Int.MaxValue).toInt

val count = session.sparkContext
.parallelize(1 until n, slices)
.map { i =>
```

```
  val x = random * 2 - 1
  val y = random * 2 - 1
  if (x*x + y*y <= 1) 1 else 0
}.reduce(_ + _)

println(s"Pi is roughly ${4.0 * count / (n - 1)}")
session.stop()
```



删除自托管式 Jupyter notebook

准备好删除自托管式 Notebook 时，也可以删除交互式端点和安全组。按以下顺序执行操作：

1. 使用以下命令删除 `jupyter-notebook` Pod：

```
kubectl delete pod jupyter-notebook -n namespace
```

2. 然后，使用 `delete-managed-endpoint` 命令删除交互式端点。有关删除交互式端点的步骤，请参阅 删除交互式端点。最初，您的终端节点将位于 TERMINATING州。清理完所有资源后，它将过渡到 TERMINATED州。

3. 如果您不打算将在 创建安全组 中创建的 Notebook 安全组用于其他 Jupyter notebook 部署，则可以将其删除。有关更多信息，请参阅 Amazon EC2 用户指南中的删除安全组。

使用 CLI 命令获取有关交互式端点的信息

本主题介绍交互式端点上支持的 `create-managed-endpoint` 以外的操作。

获取交互式端点详细信息

创建交互式终端节点后，您可以使用describe-managed-endpoint Amazon CLI 命令检索其详细信息。为*managed-endpoint-id*、*virtual-cluster-id*和：插入您自己的值*region*：

```
aws emr-containers describe-managed-endpoint --id managed-endpoint-id \
 --virtual-cluster-id virtual-cluster-id --region region
```

输出类似于以下内容，具有指定的端点，例如 ARN、ID 和名称。

```
{
   "id": "as3ys2xxxxxxx",
   "name": "endpoint-name",
    "arn": "arn:aws:emr-containers:us-east-1:1828xxxxxxxx:/virtualclusters/
lbhl6kwwyoxxxxxxxxxxxxxxx/endpoints/as3ysxxxxxxxx",
    "virtualClusterId": "lbhl6kwwyoxxxxxxxxxxxxxxx",
    "type": "JUPYTER_ENTERPRISE_GATEWAY",
    "state": "ACTIVE",
    "releaseLabel": "emr-6.9.0-latest",
   "executionRoleArn": "arn:aws:iam::1828xxxxxxxx:role/RoleName",
    "certificateAuthority": {
        "certificateArn": "arn:aws:acm:us-east-1:1828xxxxxxxx:certificate/zzzzzzzz-
e59b-4ed0-aaaa-bbbbbbbbbbbb",
        "certificateData": "certificate-data"
    },
    "configurationOverrides": {
        "applicationConfiguration": [
            {
                "classification": "spark-defaults",
                "properties": {
                    "spark.driver.memory": "8G"
                }
            }
        ],
```

```
        "monitoringConfiguration": {
            "persistentAppUI": "ENABLED",
            "cloudWatchMonitoringConfiguration": {
                "logGroupName": "log-group-name",
                "logStreamNamePrefix": "log-stream-name-prefix"
            },
            "s3MonitoringConfiguration": {
                "logUri": "s3-bucket-name"
            }
        }
    },
   "serverUrl": "https://internal-k8s-namespace-ingressa-aaaaaaaaaa-
zzzzzzzzzz.us-east-1.elb.amazonaws.com:18888 (https://internal-k8s-nspluto-
ingressa-51e860abbd-1620715833.us-east-1.elb.amazonaws.com:18888/)",
    "createdAt": "2022-09-19T12:37:49+00:00",
    "securityGroup": "sg-aaaaaaaaaaaaaa",
    "subnetIds": [
        "subnet-11111111111",
        "subnet-22222222222",
        "subnet-33333333333"
    ],
    "stateDetails": "Endpoint created successfully. It took 3 Minutes 15 Seconds",
    "tags": {}
 }
```

列出与虚拟集群关联的所有交互式端点

使用`list-managed-endpoints` Amazon CLI 命令获取与指定虚拟集群关联的所有交互式终端节点的列表。用虚拟集群的 ID 来替换 `virtual-cluster-id`。

```
aws emr-containers list-managed-endpoints --virtual-cluster-id virtual-cluster-id
```

`list-managed-endpoint` 命令的输出如下所示：

```
{
    "endpoints": [{
        "id": "as3ys2xxxxxxx",
        "name": "endpoint-name",
        "arn": "arn:aws:emr-containers:us-east-1:1828xxxxxxxx:/virtualclusters/
lbhl6kwwyoxxxxxxxxxxxxxxx/endpoints/as3ysxxxxxxxx",
        "virtualClusterId": "lbhl6kwwyoxxxxxxxxxxxxxxx",
        "type": "JUPYTER_ENTERPRISE_GATEWAY",
```

```
        "state": "ACTIVE",
        "releaseLabel": "emr-6.9.0-latest",
        "executionRoleArn": "arn:aws:iam::1828xxxxxxxx:role/RoleName",
        "certificateAuthority": {
            "certificateArn": "arn:aws:acm:us-east-1:1828xxxxxxxx:certificate/zzzzzzzz-
e59b-4ed0-aaaa-bbbbbbbbbbbb",
            "certificateData": "certificate-data"
        },
        "configurationOverrides": {
            "applicationConfiguration": [{
                "classification": "spark-defaults",
                "properties": {
                    "spark.driver.memory": "8G"
                }
            }],
            "monitoringConfiguration": {
                "persistentAppUI": "ENABLED",
                "cloudWatchMonitoringConfiguration": {
                    "logGroupName": "log-group-name",
                    "logStreamNamePrefix": "log-stream-name-prefix"
                },
                "s3MonitoringConfiguration": {
                    "logUri": "s3-bucket-name"
                }
            }
        },
        "serverUrl": "https://internal-k8s-namespace-ingressa-aaaaaaaaaa-
zzzzzzzzzz.us-east-1.elb.amazonaws.com:18888 (https://internal-k8s-nspluto-
ingressa-51e860abbd-1620715833.us-east-1.elb.amazonaws.com:18888/)",
        "createdAt": "2022-09-19T12:37:49+00:00",
        "securityGroup": "sg-aaaaaaaaaaaaaa",
        "subnetIds": [
            "subnet-11111111111",
            "subnet-22222222222",
            "subnet-33333333333"
        ],
        "stateDetails": "Endpoint created successfully. It took 3 Minutes 15 Seconds",
        "tags": {}
    }]
}
```

删除交互式端点

要删除与 EKS 虚拟集群上的 Amazon EMR 关联的交互式终端节点，请使用命令。`delete-managed-endpoint` Amazon CLI 删除交互式端点时，Amazon EMR on EKS 会删除为该端点创建的默认安全组。

为命令指定以下参数的值：

- --id：要删除的交互式端点的标识符。
- -# virtual-cluster-id —— 与要删除的交互式终端节点关联的虚拟集群的标识符。这与创建交互式端点时指定的虚拟集群 ID 相同。

```
aws emr-containers delete-managed-endpoint --id managed-endpoint-id --virtual-cluster-id virtual-cluster-id
```

该命令会返回类似于以下内容的输出，以确认交互式端点已删除：

```
{
    "id":"8gai4l4exxxxx",
    "virtualClusterId":"0b0qvauoy3ch1nqodxxxxxxxx"
}
```

使用 Amazon EMR on EKS 将数据上传到 Amazon S3 Express One Zone 存储类

对于 Amazon EMR 7.2.0 及更高版本，可将 Amazon EMR on EKS 与 Amazon S3 Express One Zone 存储类结合使用，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

在将 S3 Express One Zone 与 Amazon EMR on EKS 结合使用之前，必须满足以下先决条件：

- 完成 Amazon EMR on EKS 设置。
- 设置 Amazon EMR on EKS 后，创建一个虚拟集群。

开始使用 S3 Express One Zone

按照以下步骤开始使用 S3 Express One Zone

1. 为作业执行角色添加 `CreateSession` 权限。当 S3 Express One Zone 最初在 S3 对象上执行 `GET`、`LIST` 或 `PUT` 等操作时，存储类会代表您调用 `CreateSession`。以下示例展示了如何授予 `CreateSession` 权限。

```
{
  "Version":"2012-10-17",
  "Statement": [
        {
            "Effect": "Allow",
            "Resource": "arn:aws:s3express:<AWS_REGION>:<ACCOUNT_ID>:bucket/DOC-
EXAMPLE-BUCKET",
            "Action": [
                "s3express:CreateSession"
            ]
        }
    ]
 }
```

2. 您必须使用 Apache Hadoop 连接器 S3A 才能访问 S3 Express 存储桶，因此请将您的 Amazon S3 更改 URIs 为使用该s3a连接器的方案。如果未使用此方案，则可以更改用于 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"
    }
  }
]
```

3. 在 spark-submit 配置中，使用 Web 身份凭证提供程序。

```
"spark.hadoop.fs.s3a.aws.credentials.provider=com.amazonaws.auth.WebIdentityTokenCredential
```

监控任务

您可以使用 Amazon E CloudWatch vents 来跟踪在 EKS 虚拟集群上的 Amazon EMR 上运行的任务。您可以使用事件来跟踪在虚拟集群上运行的任务活动及其运行状况。以下主题展示了如何有效配置监视以维护资源运行状况。

主题

- 通过 Amazon Events CloudWatch 监控作业
- 通过事件在 EKS 上自动执行 Amazon EMR CloudWatch
- 示例：设置调用 Lambda 的规则
- 使用 Amazon CloudWatch Events 使用重试策略监控作业的驱动程序窗格

通过 Amazon Events CloudWatch 监控作业

当任务运行状态发生变化时，Amazon EMR on EKS 会发出事件。每个事件都提供事件发生日期及时间等信息，以及有关事件的更多详情，如受影响的虚拟集群 ID 和任务运行 ID。

您可以使用事件来跟踪在虚拟集群上运行的任务活动及其运行状况。您还可以使用 Amazon CloudWatch Events 来定义在任务运行生成与您指定的模式相匹配的事件时要采取的操作。事件对于在任务运行的生命周期中监控特定事件非常有用。例如，您可以监控任务运行的状态何时从 `submitted` 更改为 `running`。有关 CloudWatch 活动的更多信息，请参阅 Amazon EventBridge 用户指南。

下表列出了 Amazon EMR on EKS 事件，以及事件指示的状态或状态更改、事件的严重性和事件消息。每个事件都以 JSON 对象表示，后者会自动发送到事件流。JSON 对象包含有关事件的更多详情。当您使用事件设置事件处理规则时，JSON 对象尤其重要，因为规则 CloudWatch 会寻求匹配 JSON 对象中的模式。有关更多信息，请参阅亚马逊 EventBridge 用户指南中的亚马逊 EventBridge 事件模式和 EKS 事件上的 Ama zon EMR。

任务运行状态更改事件

状态	严重性	消息
SUBMITTED	INFO	Job Run *JobRunId* (*JobRunName*) 已*VirtualClusterId* 在 *Time* UTC 成功提交到虚拟集群。

状态	严重性	消息
RUNNING (正在运行)	INFO	虚拟集群中的 Job Run *JobRunId* (*JobRunName*) *VirtualClusterId* 开始在运行*Time*。
COMPLETED	INFO	虚拟群集中的 Job Run *jobRunId* (*JobRunName*) *VirtualClusterId* 已于完成*Time*。Job Run 开始运行*Time*并花了*Num*几分钟才完成。
CANCELLED	警告	在的虚拟集群*VirtualClusterId* 中，Job Run *JobRunId* (*JobRunName*) 的取消请求已成功，任务运行现已取消。*Time*
FAILED	ERROR	虚拟群集中的 Job Run *JobRunId* (*JobRunName*) *VirtualClusterId* 失败了*Time*。

通过事件在 EKS 上自动执行 Amazon EMR CloudWatch

您可以使用 Amazon CloudWatch Events 自动 Amazon 提供服务，以响应系统事件，例如应用程序可用性问题或资源更改。来自 Amazon 服务的事件几乎实时地传递到 CloudWatch 活动。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。可自动触发的操作包括：

- 调用函数 Amazon Lambda
- 调用 Amazon EC2 运行命令
- 将事件中继到 Amazon Kinesis Data Streams
- 激活 Amazon Step Functions 状态机
- 通知 Amazon Simple Notification Service (SNS) 主题或 (SQS) Amazon Simple Queue Service 队列

在 EKS 上使用 Amazon EMR CloudWatch 事件的一些示例包括：

- 当任务运行成功时激活 Lambda 函数
- 当任务运行失败时通知 Amazon SNS 主题

CloudWatch “” `detail-type:EMR Job Run State Change`” 的事件由 Amazon EMR 在 EKS 上生成，用于`SUBMITTED`、`RUNNINGCANCELLED`、`FAILED`和`COMPLETED`状态变更。

示例：设置调用 Lambda 的规则

使用以下步骤设置 CloudWatch 事件规则，该规则在出现 “EMR Job 运行状态更改” 事件时调用 Lambda。

```
aws events put-rule \
--name cwe-test \
--event-pattern '{"detail-type": ["EMR Job Run State Change"]}'
```

将您拥有的 Lambda 函数添加为新目标，并按如下方式授予 CloudWatch 事件调用 Lambda 函数的权限。请将 *123456789012* 替换为您的账户 ID。

```
aws events put-targets \
--rule cwe-test \
--targets Id=1,Arn=arn:aws:lambda:us-east-1:123456789012:function:MyFunction
```

```
aws lambda add-permission \
--function-name MyFunction \
--statement-id MyId \
--action 'lambda:InvokeFunction' \
--principal events.amazonaws.com
```

> **Note**
> 您不能编写取决于通知事件的顺序或存在的程序，因为它们可能是乱序或缺失的。尽最大努力发出事件。

使用 Amazon CloudWatch Events 使用重试策略监控作业的驱动程序窗格

使用 CloudWatch 事件，您可以监控在具有重试策略的作业中创建的驱动程序 Pod。有关更多信息，请参阅本指南中的使用重试策略监控作业。

管理虚拟集群

虚拟集群是 Amazon EMR 注册的 Kubernetes 命名空间。您可以创建、描述、列出和删除虚拟集群。它们不会消耗您系统中的任何额外资源。一个虚拟集群映射到一个 Kubernetes 命名空间。鉴于这种关系，您可以按照对 Kubernetes 命名空间进行建模的方式对虚拟集群进行建模，来满足您的需求。请参阅 Kubernetes 概念概览文档中可能的使用案例。

要在 Amazon EKS 集群上使用 Kubernetes 命名空间注册 Amazon EMR，您需要 EKS 集群的名称以及为运行您的工作负载而设置的命名空间。Amazon EMR 中的这些注册集群称为虚拟集群，因为它们不管理物理计算或存储，而是指向计划工作负载的 Kubernetes 命名空间。

> **Note**
> 在创建虚拟集群之前，您必须首先完成设置 Amazon EMR on EKS中的步骤 1-8。

主题

- 创建虚拟集群
- 列出虚拟集群
- 描述虚拟集群
- 删除虚拟集群
- 虚拟集群状态

创建虚拟集群

运行以下命令，通过使用 EKS 集群上的命名空间注册 Amazon EMR 来创建虚拟集群。*virtual_cluster_name*用您为虚拟集群提供的名称替换。*eks_cluster_name*替换为 EKS 集群的名称。将*namespace_name*替换为您想要注册 Amazon EMR 的命名空间。

```
aws emr-containers create-virtual-cluster \
--name virtual_cluster_name \
--container-provider '{
    "id": "eks_cluster_name",
    "type": "EKS",
    "info": {
```

```
            "eksInfo": {
                "namespace": "namespace_name"
            }
        }
}'
```

或者，您可以创建包含虚拟集群所需参数的 JSON 文件，如以下示例所示。

```
{
    "name": "virtual_cluster_name",
    "containerProvider": {
        "type": "EKS",
        "id": "eks_cluster_name",
        "info": {
            "eksInfo": {
                "namespace": "namespace_name"
            }
        }
    }
}
```

然后使用 JSON 文件路径运行以下 `create-virtual-cluster` 命令。

```
aws emr-containers create-virtual-cluster \
--cli-input-json file://./create-virtual-cluster-request.json
```

> **Note**
> 要验证虚拟集群是否成功创建，请通过运行 `list-virtual-clusters` 命令或转到 Amazon EMR 控制台中的 Virtual clusters (虚拟集群) 页面，以查看虚拟集群状态。

列出虚拟集群

运行以下命令来查看虚拟集群的状态。

```
aws emr-containers list-virtual-clusters
```

描述虚拟集群

运行以下命令以获取有关虚拟集群的详细信息，例如命名空间、状态和注册日期。*123456*替换为您的虚拟集群 ID。

```
aws emr-containers describe-virtual-cluster --id 123456
```

删除虚拟集群

运行以下命令来删除虚拟集群。*123456*替换为您的虚拟集群 ID。

```
aws emr-containers delete-virtual-cluster --id 123456
```

虚拟集群状态

下表描述了虚拟集群的四种可能状态。

State	描述
RUNNING	虚拟集群处于 RUNNING 状态。
TERMINATING	正在请求终止虚拟集群。
TERMINATED	已完成请求的终止。
ARRESTED	由于权限不足，请求的终止失败。

Amazon EMR on EKS 教程

本部分介绍使用 Amazon EMR on EKS 应用程序时的常见用例。每个应用程序都是专门的，可采取独特的步骤进行配置。这些主题提供了每个应用程序的使用说明。

主题

- 将 Delta Lake 与 Amazon EMR on EKS 结合使用
- 将 Apache Iceberg 与 Amazon EMR on EKS 结合使用
- 使用 PyFlink
- 在 Flink Amazon 中使用 Glue
- 将 Apache Hudi 与 Apache Flink 结合使用
- 将适用于 Apache Spark 的 RAPIDS Accelerator 与 Amazon EMR on EKS 结合使用
- 在 Amazon EMR on EKS 上使用适用于 Apache Spark 的 Amazon Redshift 集成
- 在 Amazon EMR on EKS 上将 Volcano 用作 Apache Spark 自定义调度器
- 在 EK YuniKorn S 上用作 Amazon EMR 上的 Apache Spark 的自定义调度程序

将 Delta Lake 与 Amazon EMR on EKS 结合使用

Delta Lake 是一个开源存储框架，用于构建湖屋架构。下面介绍了如何设置使用。

将 Delta Lake 与 Amazon EMR on EKS 应用程序结合使用

1. 启动任务运行以在应用程序配置中提交 Spark 任务时，请包含 Delta Lake JAR 文件：

```
--job-driver '{"sparkSubmitJobDriver" : {
      "sparkSubmitParameters" : "--jars local:///usr/share/aws/delta/lib/delta-
core.jar,local:///usr/share/aws/delta/lib/delta-storage.jar,local:///usr/share/aws/
delta/lib/delta-storage-s3-dynamodb.jar"}}'
```

> Note
>
> Amazon EMR 7.0.0 及更高版本使用 Delta Lake 3.0，该版本将 `delta-core.jar` 更名为 `delta-spark.jar`。如果您使用的是 Amazon EMR 7.0.0 或更高版本，请务必使用正确的文件名，如以下示例所示：

```
--jars local:///usr/share/aws/delta/lib/delta-spark.jar
```

2. 包括 Delta Lake 的其他配置，并使用 Amazon Glue 数据目录作为您的元数据仓。

```
--configuration-overrides '{
        "applicationConfiguration": [
        {
          "classification" : "spark-defaults",
          "properties" : {
            "spark.sql.extensions" : "io.delta.sql.DeltaSparkSessionExtension",

 "spark.sql.catalog.spark_catalog":"org.apache.spark.sql.delta.catalog.DeltaCatalog",
"spark.hadoop.hive.metastore.client.factory.class":"com.amazonaws.glue.catalog.metastore.AW
           }
        }]}'
```

将 Apache Iceberg 与 Amazon EMR on EKS 结合使用

Iceberg 的运行时 JAR 文件包含 Spark 运行时支持所需的 Iceberg 类。以下过程展示了如何使用 Iceberg spark 运行时启动作业运行。

将 Apache Iceberg 与 Amazon EMR on EKS 应用程序结合使用

1. 启动任务运行以在应用程序配置中提交 Spark 任务时，请包含 Iceberg Spark 运行时 JAR 文件：

```
--job-driver '{"sparkSubmitJobDriver" : {"sparkSubmitParameters" : "--jars
 local:///usr/share/aws/iceberg/lib/iceberg-spark3-runtime.jar"}}'
```

2. 包含 Iceberg 额外配置：

```
--configuration-overrides '{
    "applicationConfiguration": [
    "classification" : "spark-defaults",
    "properties" : {
        "spark.sql.catalog.dev.warehouse" : "s3://amzn-s3-demo-bucket/EXAMPLE-
PREFIX/ ",
        "spark.sql.extensions ":"
 org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions ",
        "spark.sql.catalog.dev" : "org.apache.iceberg.spark.SparkCatalog",
```

```
        "spark.sql.catalog.dev.catalog-impl" :
 "org.apache.iceberg.aws.glue.GlueCatalog",
        "spark.sql.catalog.dev.io-impl": "org.apache.iceberg.aws.s3.S3FileIO"
        }
    ]
}'
```

要了解有关 EMR Apache Iceberg 发行版的更多信息，请参阅 Iceberg 发布历史记录。

用于目录集成的 Spark 会话配置

Iceberg Amazon Glue 目录集成的 Spark 会话配置

此示例演示如何将 Iceberg 与以下各项集成： Amazon Glue 爬网程序

```
spark-sql \
  --conf spark.sql.catalog.rms = org.apache.iceberg.spark.SparkCatalog \
  --conf spark.sql.catalog.rms.type = glue \
  --conf spark.sql.catalog.rms.glue.id = glue RMS catalog ID \
  --conf spark.sql.catalog.rms.glue.account-id = Amazon account ID \

  --conf spark.sql.extensions=
    org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions
```

下面显示了一个示例查询：

```
SELECT * FROM rms.rmsdb.table1
```

Iceberg REST Amazon Glue 目录集成的 Spark 会话配置

此示例展示了如何将 Iceberg REST 与以下各项集成： Amazon Glue 爬网程序

```
spark-sql \
  --conf spark.sql.catalog.rms = org.apache.iceberg.spark.SparkCatalog \
  --conf spark.sql.catalog.rms.type = rest \
  --conf spark.sql.catalog.rms.warehouse = glue RMS catalog ID \
  --conf spark.sql.catalog.rms.uri = glue endpoint URI/iceberg \
  --conf spark.sql.catalog.rms.rest.sigv4-enabled = true \
  --conf spark.sql.catalog.rms.rest.signing-name = glue \
```

```
  --conf spark.sql.extensions=
    org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions
```

下面显示了一个示例查询：

```
SELECT * FROM rms.rmsdb.table1
```

此配置仅适用于 Redshift 托管存储。不支持 Amazon S3 的 FGAC。

使用 PyFlink

EKS 上的 Amazon EMR 版本 6.15.0 及更高版本支持。 PyFlink如果您已经有 PyFlink 脚本，则可以执行以下操作之一：

- 创建包含 PyFlink 脚本的自定义镜像。
- 将脚本上传到 Amazon S3 位置

如果您还没有脚本，则可以使用以下示例启动作 PyFlink 业。该示例从 S3 检索脚本。如果使用的自定义映像中已包含脚本，则必须将脚本路径更新为存储脚本的位置。如果脚本位于 S3 位置，Amazon EMR on EKS 将检索该脚本并将其放在 Flink 容器中的 `/opt/flink/usrlib/` 目录下。

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: python-example
spec:
  flinkVersion: v1_17
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "1"
  executionRoleArn: job-execution-role
  emrReleaseLabel: "emr-6.15.0-flink-latest"
  jobManager:
    highAvailabilityEnabled: false
    replicas: 1
    resource:
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
      memory: "2048m"
```

```
      cpu: 1
  job:
    jarURI: s3://S3 bucket with your script/pyflink-script.py
    entryClass: "org.apache.flink.client.python.PythonDriver"
    args: ["-py", "/opt/flink/usrlib/pyflink-script.py"]
    parallelism: 1
    upgradeMode: stateless
```

在 Flink Amazon 中使用 Glue

带有 Apache Flink 版本 6.15.0 及更高版本的 EKS 上的 Amazon EMR 支持使用 G Amazon lue 数据目录作为流式处理和批处理 SQL 工作流程的元数据存储。

你必须先创建一个名为 Gl Amazon ue 的数据库`default`，用作你的 Flink SQL 目录。此 Flink 目录存储元数据，例如数据库、表、分区、视图、函数以及访问其他外部系统数据所需的其他信息。

```
aws glue create-database \
    --database-input "{\"Name\":\"default\"}"
```

要启用 Amazon Glue 支持，请使用`FlinkDeployment`规范。此示例规范使用 Python 脚本快速发出一些 Flink SQL 语句来与 Amazon Glue 目录进行交互。

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: python-example
spec:
  flinkVersion: v1_17
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "1"
    aws.glue.enabled: "true"
  executionRoleArn: job-execution-role-arn;
  emrReleaseLabel: "emr-6.15.0-flink-latest"
  jobManager:
    highAvailabilityEnabled: false
    replicas: 1
    resource:
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
```

```
      memory: "2048m"
      cpu: 1
  job:
    jarURI: s3://<S3_bucket_with_your_script/pyflink-glue-script.py
    entryClass: "org.apache.flink.client.python.PythonDriver"
    args: ["-py", "/opt/flink/usrlib/pyflink-glue-script.py"]
    parallelism: 1
    upgradeMode: stateless
```

以下是您的 PyFlink 脚本可能的样子示例。

```
import logging
import sys
from pyflink.datastream import StreamExecutionEnvironment
from pyflink.table import StreamTableEnvironment

def glue_demo():
    env = StreamExecutionEnvironment.get_execution_environment()
    t_env = StreamTableEnvironment.create(stream_execution_environment=env)
    t_env.execute_sql("""
          CREATE CATALOG glue_catalog WITH (
          'type' = 'hive',
          'default-database' = 'default',
          'hive-conf-dir' = '/glue/confs/hive/conf',
          'hadoop-conf-dir' = '/glue/confs/hadoop/conf'
          )
                        """)
    t_env.execute_sql("""
          USE CATALOG glue_catalog;
                        """)
    t_env.execute_sql("""
          DROP DATABASE IF EXISTS eks_flink_db CASCADE;
                        """)
    t_env.execute_sql("""
          CREATE DATABASE IF NOT EXISTS eks_flink_db WITH ('hive.database.location-
uri'= 's3a://S3-bucket-to-store-metadata/flink/flink-glue-for-hive/warehouse/');
                        """)
    t_env.execute_sql("""
          USE eks_flink_db;
                    """)
    t_env.execute_sql("""
          CREATE TABLE IF NOT EXISTS eksglueorders (
            order_number BIGINT,
```

```
            price        DECIMAL(32,2),
            buyer        RO first_name STRING, last_name STRING,
            order_time   TIMESTAMP(3)
          ) WITH (
            'connector' = 'datagen'
          );
                    """)
    t_env.execute_sql("""
          CREATE TABLE IF NOT EXISTS eksdestglueorders (
            order_number BIGINT,
            price        DECIMAL(32,2),
            buyer        ROW first_name STRING, last_name STRING,
            order_time   TIMESTAMP(3)
          ) WITH (
            'connector' = 'filesystem',
            'path' = 's3://S3-bucket-to-store-metadata/flink/flink-glue-for-hive/
warehouse/eksdestglueorders',
            'format' = 'json'
          );
                """)
    t_env.execute_sql("""
          CREATE TABLE IF NOT EXISTS print_table (
            order_number BIGINT,
            price        DECIMAL(32,2),
            buyer        ROW first_name STRING, last_name STRING,
            order_time   TIMESTAMP(3)
          ) WITH (
            'connector' = 'print'
          );
              """)
    t_env.execute_sql("""
          EXECUTE STATEMENT SET
          BEGIN
          INSERT INTO eksdestglueorders SELECT * FROM  eksglueorders LIMIT 10;
          INSERT INTO print_table SELECT * FROM eksdestglueorders;
          END;
            """)


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

将 Apache Hudi 与 Apache Flink 结合使用

Apache Hudi 是一个开源数据管理框架，包含插入、更新、更新插入和删除等记录级操作，可用于简化数据管理和数据管道开发。与 Amazon S3 中的高效数据管理相结合，Hudi 允许实时摄取和更新数据。Hudi 会维护在数据集上运行的所有操作的元数据，因此所有操作都能保持原子性和一致性。

Apache Hudi 已在 Amazon EMR on EKS 上投入使用，搭配 Apache Flink 和 Amazon EMR 7.2.0 及更高版本。请参阅以下步骤，了解如何开始和提交 Apache Hudi 作业。

提交 Apache Hudi 作业

请参阅以下步骤，了解如何提交 Apache Hudi 作业。

1. 创建一个名为的 Amazon Glue 数据库`default`。

```
aws glue create-database --database-input "{\"Name\":\"default\"}"
```

2. 按照 Flink Kubernetes Operator SQL 示例构建 `flink-sql-runner.jar` 文件。
3. 创建如下所示的 Hudi SQL 脚本。

```
CREATE CATALOG hudi_glue_catalog WITH (
'type' = 'hudi',
'mode' = 'hms',
'table.external' = 'true',
'default-database' = 'default',
'hive.conf.dir' = '/glue/confs/hive/conf/',
'catalog.path' = 's3://<hudi-example-bucket>/FLINK_HUDI/warehouse/'
);

USE CATALOG hudi_glue_catalog;
CREATE DATABASE IF NOT EXISTS hudi_db;
use hudi_db;

CREATE TABLE IF NOT EXISTS hudi-flink-example-table(
    uuid VARCHAR(20),
    name VARCHAR(10),
    age INT,
    ts TIMESTAMP(3),
    `partition` VARCHAR(20)
)
PARTITIONED BY (`partition`)
WITH (
```

```
  'connector' = 'hudi',
  'path' = 's3://<hudi-example-bucket>/hudi-flink-example-table',
  'hive_sync.enable' = 'true',
  'hive_sync.mode' = 'glue',
  'hive_sync.table' = 'hudi-flink-example-table',
  'hive_sync.db' = 'hudi_db',
  'compaction.delta_commits' = '1',
  'hive_sync.partition_fields' = 'partition',
  'hive_sync.partition_extractor_class' =
 'org.apache.hudi.hive.MultiPartKeysValueExtractor',
  'table.type' = 'COPY_ON_WRITE'
);

EXECUTE STATEMENT SET
BEGIN

INSERT INTO hudi-flink-example-table VALUES
    ('id1','Alex',23,TIMESTAMP '1970-01-01 00:00:01','par1'),
    ('id2','Stephen',33,TIMESTAMP '1970-01-01 00:00:02','par1'),
    ('id3','Julian',53,TIMESTAMP '1970-01-01 00:00:03','par2'),
    ('id4','Fabian',31,TIMESTAMP '1970-01-01 00:00:04','par2'),
    ('id5','Sophia',18,TIMESTAMP '1970-01-01 00:00:05','par3'),
    ('id6','Emma',20,TIMESTAMP '1970-01-01 00:00:06','par3'),
    ('id7','Bob',44,TIMESTAMP '1970-01-01 00:00:07','par4'),
    ('id8','Han',56,TIMESTAMP '1970-01-01 00:00:08','par4');

END;
```

4. 将 Hudi SQL 脚本和 `flink-sql-runner.jar` 文件上传到 S3 位置。

5. 在 `FlinkDeployments` YAML 文件中，将 `hudi.enabled` 设置为 `true`。

```
spec:
  flinkConfiguration:
    hudi.enabled: "true"
```

6. 创建一个 YAML 文件来运行配置。本示例文件名为 `hudi-write.yaml`。

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

```
flinkConfiguration:
  taskmanager.numberOfTaskSlots: "2"
  hudi.enabled: "true"
executionRoleArn: "<JobExecutionRole>"
emrReleaseLabel: "emr-7.9.0-flink-latest"
jobManager:
  highAvailabilityEnabled: false
  replicas: 1
  resource:
    memory: "2048m"
    cpu: 1
taskManager:
  resource:
    memory: "2048m"
    cpu: 1
job:
  jarURI: local:///opt/flink/usrlib/flink-sql-runner.jar
  args: ["/opt/flink/scripts/hudi-write.sql"]
  parallelism: 1
  upgradeMode: stateless
podTemplate:
  spec:
    initContainers:
      - name: flink-sql-script-download
        args:
          - s3
          - cp
          - s3://<s3_location>/hudi-write.sql
          - /flink-scripts
        image: amazon/aws-cli:latest
        imagePullPolicy: Always
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
        volumeMounts:
          - mountPath: /flink-scripts
            name: flink-scripts
      - name: flink-sql-runner-download
        args:
          - s3
          - cp
          - s3://<s3_location>/flink-sql-runner.jar
          - /flink-artifacts
        image: amazon/aws-cli:latest
```

```
          imagePullPolicy: Always
          resources: {}
          terminationMessagePath: /dev/termination-log
          terminationMessagePolicy: File
          volumeMounts:
            - mountPath: /flink-artifacts
              name: flink-artifact
      containers:
        - name: flink-main-container
          volumeMounts:
            - mountPath: /opt/flink/scripts
              name: flink-scripts
            - mountPath: /opt/flink/usrlib
              name: flink-artifact
      volumes:
        - emptyDir: {}
          name: flink-scripts
        - emptyDir: {}
          name: flink-artifact
```

7. 将 Flink Hudi 作业提交到 Flink Kubernetes Operator。

```
kubectl apply -f hudi-write.yaml
```

将适用于 Apache Spark 的 RAPIDS Accelerator 与 Amazon EMR on EKS 结合使用

使用 Amazon EMR on EKS，您可以为适用于 Apache Spark 的 Nvidia RAPIDS Accelerator 运行任务。本教程介绍如何在 EC2 图形处理单元 (GPU) 实例类型上使用 RAPIDS 运行 Spark 作业。本教程使用以下版本：

- Amazon EMR on EKS 发行版 6.9.0 及更高版本
- Apache Spark 3.x

你可以使用适用于 Apache Spark 的 Nvidia RAPIDS 加速器插件使用亚马逊 EC2 GPU 实例类型加速 Spark。当您结合使用这些技术时，您可以加速数据科学管道，而无需更改任何代码。这可以减少数据处理和模型训练所需的运行时间。通过在更短的时间内完成更多工作，降低基础设施成本。

在开始之前，请确保您具有以下资源。

- Amazon EMR on EKS 虚拟集群
- 包含支持 GPU 的节点组的 Amazon EKS 集群

Amazon EKS 虚拟集群是 Amazon EKS 集群上 Kubernetes 命名空间的注册句柄，由 Amazon EMR on EKS 管理。该句柄允许 Amazon EMR 使用 Kubernetes 命名空间作为运行任务的目标。有关如何设置虚拟集群的更多信息，请参阅本指南中的 设置 Amazon EMR on EKS。

您必须使用具有 GPU 实例的节点组配置 Amazon EKS 虚拟集群。您必须使用 Nvidia 设备插件配置节点。请参阅 managed node groups（托管节点组）以了解更多信息。

要将 Amazon EKS 集群配置为添加支持 GPU 的节点组，请执行以下步骤：

添加支持 GPU 的节点组

1. 使用以下 create-nodegroup 命令创建支持 GPU 的节点组。请务必为 Amazon EKS 集群替换正确的参数。使用支持 Spark RAPIDS 的实例类型，例如 P4、P3、G5 或 G4dn。

```
aws eks create-nodegroup \
 --cluster-name EKS_CLUSTER_NAME \
 --nodegroup-name NODEGROUP_NAME \
 --scaling-config minSize=0,maxSize=5,desiredSize=2 CHOOSE_APPROPRIATELY \
 --ami-type AL2_x86_64_GPU \
 --node-role NODE_ROLE \
 --subnets SUBNETS_SPACE_DELIMITED  \
 --remote-access ec2SshKey= SSH_KEY \
 --instance-types GPU_INSTANCE_TYPE \
 --disk-size DISK_SIZE \
 --region AWS_REGION
```

2. 在集群中安装 Nvidia 设备插件，以在集群的每个节点 GPUs 上发出数量，并在集群中运行支持 GPU 的容器。运行以下命令以安装插件：

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/v0.9.0/
nvidia-device-plugin.yml
```

3. 要验证集群的每个节点上 GPUs 有多少可用节点，请运行以下命令：

```
kubectl get nodes  "-o=custom-
columns=NAME:.metadata.name,GPU:.status.allocatable.nvidia\.com/gpu"
```

运行 Spark RAPIDS 任务

1. 将 Spark RAPIDS 任务提交到 Amazon EMR on EKS 集群。以下代码显示了用于启动任务的命令示例。首次运行任务时，可能需要几分钟下载映像并将其缓存到节点上。

```
aws emr-containers start-job-run \
--virtual-cluster-id VIRTUAL_CLUSTER_ID \
--execution-role-arn JOB_EXECUTION_ROLE \
--release-label emr-6.9.0-spark-rapids-latest \
--job-driver '{"sparkSubmitJobDriver": {"entryPoint": "local:///usr/lib/
spark/examples/jars/spark-examples.jar","entryPointArguments":  ["10000"],
 "sparkSubmitParameters":"--class org.apache.spark.examples.SparkPi "}}' \
---configuration-overrides '{"applicationConfiguration": [{"classification":
 "spark-defaults","properties": {"spark.executor.instances":
 "2","spark.executor.memory": "2G"}}],"monitoringConfiguration":
 {"cloudWatchMonitoringConfiguration": {"logGroupName": "LOG_GROUP
 _NAME"},"s3MonitoringConfiguration": {"logUri": "LOG_GROUP_STREAM"}}}'
```

2. 要验证 Spark RAPIDS Accelerator 是否已启用，请检查 Spark 驱动程序日志。这些日志存储在您运行`start-job-run`命令时指定的 S3 位置 CloudWatch 或位置。以下示例大致显示了日志行的具体形式：

```
22/11/15 00:12:44 INFO RapidsPluginUtils: RAPIDS Accelerator build:
 {version=22.08.0-amzn-0, user=release, url=, date=2022-11-03T03:32:45Z, revision=,
 cudf_version=22.08.0, branch=}
22/11/15 00:12:44 INFO RapidsPluginUtils: RAPIDS Accelerator JNI build:
 {version=22.08.0, user=, url=https://github.com/NVIDIA/spark-rapids-jni.git,
 date=2022-08-18T04:14:34Z, revision=a1b23cd_sample, branch=HEAD}
22/11/15 00:12:44 INFO RapidsPluginUtils: cudf build: {version=22.08.0,
 user=, url=https://github.com/rapidsai/cudf.git, date=2022-08-18T04:14:34Z,
 revision=a1b23ce_sample, branch=HEAD}
22/11/15 00:12:44 WARN RapidsPluginUtils: RAPIDS Accelerator 22.08.0-amzn-0 using
 cudf 22.08.0.
22/11/15 00:12:44 WARN RapidsPluginUtils:
 spark.rapids.sql.multiThreadedRead.numThreads is set to 20.
22/11/15 00:12:44 WARN RapidsPluginUtils: RAPIDS Accelerator is enabled, to disable
 GPU support set `spark.rapids.sql.enabled` to false.
22/11/15 00:12:44 WARN RapidsPluginUtils: spark.rapids.sql.explain is set to
 `NOT_ON_GPU`. Set it to 'NONE' to suppress the diagnostics logging about the query
 placement on the GPU.
```

3. 要查看将在 GPU 上运行的操作，请执行以下步骤以启用额外的日志记录。请注意“spark.rapids.sql.explain : ALL”配置。

```
aws emr-containers start-job-run \
--virtual-cluster-id VIRTUAL_CLUSTER_ID \
--execution-role-arn JOB_EXECUTION_ROLE \
--release-label emr-6.9.0-spark-rapids-latest \
--job-driver '{"sparkSubmitJobDriver": {"entryPoint": "local:///usr/lib/
spark/examples/jars/spark-examples.jar","entryPointArguments":  ["10000"],
 "sparkSubmitParameters":"--class org.apache.spark.examples.SparkPi "}}' \
---configuration-overrides '{"applicationConfiguration":
 [{"classification": "spark-defaults","properties":
 {"spark.rapids.sql.explain":"ALL","spark.executor.instances":
 "2","spark.executor.memory": "2G"}}],"monitoringConfiguration":
 {"cloudWatchMonitoringConfiguration": {"logGroupName":
 "LOG_GROUP_NAME"},"s3MonitoringConfiguration": {"logUri": "LOG_GROUP_STREAM"}}}'
```

上一个命令是使用 GPU 的任务示例。其输出类似于以下示例。请参阅此密钥以帮助了解输出：

- * – 标记适用于 GPU 的操作
- ! – 标记无法在 GPU 上运行的操作
- @ – 标记适用于 GPU 但无法运行的操作，因为其所处的计划无法在 GPU 上运行

```

 22/11/15 01:22:58 INFO GpuOverrides: Plan conversion to the GPU took 118.64 ms
 22/11/15 01:22:58 INFO GpuOverrides: Plan conversion to the GPU took 4.20 ms
 22/11/15 01:22:58 INFO GpuOverrides: GPU plan transition optimization took 8.37 ms
 22/11/15 01:22:59 WARN GpuOverrides:
    *Exec <ProjectExec> will run on GPU
      *Expression <Alias> substring(cast(date#149 as string), 0, 7) AS month#310
 will run on GPU
        *Expression <Substring> substring(cast(date#149 as string), 0, 7) will run
 on GPU
          *Expression <Cast> cast(date#149 as string) will run on GPU
      *Exec <SortExec> will run on GPU
        *Expression <SortOrder> date#149 ASC NULLS FIRST will run on GPU
        *Exec <ShuffleExchangeExec> will run on GPU
          *Partitioning <RangePartitioning> will run on GPU
            *Expression <SortOrder> date#149 ASC NULLS FIRST will run on GPU
          *Exec <UnionExec> will run on GPU
```

```
            !Exec <ProjectExec> cannot run on GPU because not all expressions can
 be replaced
              @Expression <AttributeReference> customerID#0 could run on GPU
              @Expression <Alias> Charge AS kind#126 could run on GPU
                @Expression <Literal> Charge could run on GPU
              @Expression <AttributeReference> value#129 could run on GPU
              @Expression <Alias> add_months(2022-11-15, cast(-(cast(_we0#142 as
 bigint) + last_month#128L) as int)) AS date#149 could run on GPU
                ! <AddMonths> add_months(2022-11-15, cast(-
(cast(_we0#142 as bigint) + last_month#128L) as int)) cannot run
 on GPU because GPU does not currently support the operator class
 org.apache.spark.sql.catalyst.expressions.AddMonths
                  @Expression <Literal> 2022-11-15 could run on GPU
                  @Expression <Cast> cast(-(cast(_we0#142 as bigint) +
 last_month#128L) as int) could run on GPU
                    @Expression <UnaryMinus> -(cast(_we0#142 as bigint) +
 last_month#128L) could run on GPU
                      @Expression <Add> (cast(_we0#142 as bigint) +
 last_month#128L) could run on GPU
                        @Expression <Cast> cast(_we0#142 as bigint) could run on
 GPU
                          @Expression <AttributeReference> _we0#142 could run on
 GPU
                        @Expression <AttributeReference> last_month#128L could run
 on GPU
```

在 Amazon EMR on EKS 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。这样，您就可以在 Amazon EMR on EKS 上使用 Spark 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 on EKS，适用于 Apache Spark 的 Amazon Redshift 集成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

要使用集成，必须通过 Spark 任务传递所需的 Spark Redshift 依赖项。您必须使用 `--jars` 将 Redshift 连接器相关的库包含在内。要查看 `--jars` 选项支持的其他文件位置，请参阅 Apache Spark 文档的 Advanced Dependency Management（高级依赖项管理）部分。

- `spark-redshift.jar`
- `spark-avro.jar`
- `RedshiftJDBC.jar`
- `minimal-json.jar`

要在 Amazon EMR on EKS 发行版 6.9.0 或更高版本上使用适用于 Apache Spark 的 Amazon Redshift 集成启动 Spark 应用程序，请使用以下示例命令。请注意，`--conf spark.jars` 选项列出的路径是 JAR 文件的默认路径。

```
aws emr-containers start-job-run \

--virtual-cluster-id cluster_id \
--execution-role-arn arn \
--release-label emr-6.9.0-latest\
--job-driver '{
    "sparkSubmitJobDriver": {
        "entryPoint": "s3://script_path",
            "sparkSubmitParameters":
            "--conf spark.kubernetes.file.upload.path=s3://upload_path
             --conf spark.jars=
                /usr/share/aws/redshift/jdbc/RedshiftJDBC.jar,
                /usr/share/aws/redshift/spark-redshift/lib/spark-redshift.jar,
                /usr/share/aws/redshift/spark-redshift/lib/spark-avro.jar,
                /usr/share/aws/redshift/spark-redshift/lib/minimal-json.jar"
                            }
            }'
```

使用适用于 Apache Spark 的 Amazon Redshift 集成进行身份验证

以下部分展示了与 Apache Spark 集成后 Amazon Redshift 提供的身份验证选项。这些部分展示了如何检索登录凭证，以及将 JDBC 驱动程序与 IAM 身份验证结合使用的详细信息。

用于 Amazon 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，以便安全地对 Amazon Redshift 进行身份验证。您可以使 Spark 任务调用 `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
```

将基于 IAM 的身份验证与 Amazon EMR on EKS 任务执行角色结合使用

从 Amazon EMR on EKS 发行版 6.9.0 开始，Amazon Redshift JDBC 驱动程序版本 2.1 或更高版本将被打包到环境中。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，您可以指定 `jdbc:redshift:iam://` 方案。从而命令 JDBC 驱动程序使用 Amazon EMR on EKS 任务执行角色自动获取凭证。

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]。

以下示例 URL 使用 `jdbc:redshift:iam://` 方案。

```
jdbc:redshift:iam://examplecluster.abc123xyz789.us-west-2.redshift.amazonaws.com:5439/
dev
```

如果任务执行角色满足提供的条件，则需要以下权限。

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 Amazon Web Services 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

您的任务执行角色策略应具备以下权限。

```
{
            "Effect": "Allow",
            "Action": [
                "redshift:GetClusterCredentials",
                "redshift:DescribeCluster",
                "redshift-serverless:GetCredentials",
                "redshift-serverless:GetWorkgroup"
            ],
            "Resource": [
 "arn:aws:redshift:AWS_REGION:ACCOUNT_ID:dbname:CLUSTER_NAME/DATABASE_NAME",
                "arn:aws:redshift:AWS_REGION:ACCOUNT_ID:dbuser:DATABASE_NAME/USER_NAME"
            ]
        }
```

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

要向 Amazon Redshift 集群验证 Spark 任务，您可以在 JDBC URL 中指定 Amazon Redshift 数据库的名称和密码。

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，则可以使用以下格式在 JDBC URL 中设置用户名和密码：

```
jdbc:redshift://redshifthost:5439/database?user=username&password=password
```

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::accountID:role/roleName"

df = sql_context.read \
    .format("io.github.spark_redshift_community.spark.redshift") \
    .option("url", url) \
    .option("dbtable", "tableName") \
    .option("tempdir", "s3://path/for/temp/data") \
    .option("aws_iam_role", "aws_iam_role_arn") \
    .load()

df.write \
    .format("io.github.spark_redshift_community.spark.redshift") \
    .option("url", url) \
    .option("dbtable", "tableName_copy") \
    .option("tempdir", "s3://path/for/temp/data") \
```

```
    .option("aws_iam_role", "aws_iam_role_arn") \
    .mode("error") \
    .save()
```

SparkSQL

PySpark 用于使用 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::accountID:role/roleName"

bucket = "s3://path/for/temp/data"
tableName = "tableName" # Redshift table name

s = f"""CREATE TABLE IF NOT EXISTS {tableName} (country string, data string) 
    USING io.github.spark_redshift_community.spark.redshift 
    OPTIONS (dbtable '{tableName}', tempdir '{bucket}', url '{url}', aws_iam_role
 '{aws_iam_role_arn}' ); """

spark.sql(s)
         
columns = ["country" ,"data"]
data = [("test-country","test-data")]
df = spark.sparkContext.parallelize(data).toDF(columns)

# Insert data into table
df.write.insertInto(tableName, overwrite=False)
df = spark.sql(f"SELECT * FROM {tableName}")
df.show()
```

使用 Spark 连接器时的注意事项和限制

Spark 连接器支持多种方式来管理凭据、配置安全性以及连接其他 Amazon 服务。熟悉此列表中的建议，以配置功能正常且具有弹性的连接。

- 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 Amazon Secrets Manager 中管理 Amazon Redshift 集群的凭证。有关示例 Amazon Secrets Manager ，请参阅使用检索连接至亚马逊 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 S3 存储桶。
  - 建议使用 Amazon S3 生命周期策略定义 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。为了安全起见，我们不支持将 `tempdir` URI 中的 Amazon 访问密钥编码为从 Spark 到 Amazon S3 的身份验证方法。

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

在 Amazon EMR on EKS 上将 Volcano 用作 Apache Spark 自定义调度器

您可以在 Amazon EMR on EKS 上使用 Spark Operator 或 spark-submit，通过 Kubernetes 自定义调度器运行 Spark 任务。本教程旨在介绍如何使用 Volcano 调度器在自定义队列上运行 Spark 任务。

概览

Volcano 可以通过队列调度、公平分享调度和资源预留等高级功能来帮助管理 Spark 调度。有关 Volcano 优势的更多信息，请参阅 The Linux Foundation CNCF 博客上的 Why Spark chooses Volcano as built-in batch scheduler on Kubernetes? 一文。

安装并设置 Volcano

1. 根据自己的架构需求，从以下 kubectl 命令中择一来安装 Volcano：

```
# x86_64
kubectl apply -f https://raw.githubusercontent.com/volcano-sh/volcano/v1.5.1/
installer/volcano-development.yaml
# arm64:
kubectl apply -f https://raw.githubusercontent.com/volcano-sh/volcano/v1.5.1/
installer/volcano-development-arm64.yaml
```

2. 准备好一个 Volcano 队列示例。队列是以下内容的集合PodGroups。队列采用 FIFO 原则，是资源划分的基础。

```
cat << EOF > volcanoQ.yaml
apiVersion: scheduling.volcano.sh/v1beta1
kind: Queue
metadata:
  name: sparkqueue
spec:
  weight: 4
  reclaimable: false
  capability:
    cpu: 10
    memory: 20Gi
EOF

kubectl apply -f volcanoQ.yaml
```

3. 将示例 PodGroup 清单上传到 Amazon S3。 PodGroup 是一组关联性很强的 pod。您通常使用 PodGroup 进行批量调度。将以下示例提交 PodGroup 到您在上一步中定义的队列。

```
cat << EOF > podGroup.yaml
apiVersion: scheduling.volcano.sh/v1beta1
kind: PodGroup
spec:
```

```
  # Set minMember to 1 to make a driver pod
  minMember: 1
  # Specify minResources to support resource reservation.
  # Consider the driver pod resource and executors pod resource.
  # The available resources should meet the minimum requirements of the Spark job
  # to avoid a situation where drivers are scheduled, but they can't schedule
  # sufficient executors to progress.
  minResources:
    cpu: "1"
    memory: "1Gi"
  # Specify the queue. This defines the resource queue that the job should be
 submitted to.
  queue: sparkqueue
EOF

aws s3 mv podGroup.yaml s3://bucket-name
```

使用 Volcano 调度器和 Spark Operator 运行 Spark 应用程序

1. 如果尚未完成设置，请按照以下各节中的步骤完成设置：

   a. 安装并设置 Volcano

   b. 设置 Amazon EMR on EKS 的 Spark Operator

   c. 安装 Spark Operator

   运行 `helm install spark-operator-demo` 命令时包括以下参数：

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

2. 创建配置了 `batchScheduler` 的 `SparkApplication` 定义文件 `spark-pi.yaml`。

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: spark-pi
  namespace: spark-operator
spec:
  type: Scala
  mode: cluster
  image: "895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.10.0:latest"
```

```
  imagePullPolicy: Always
  mainClass: org.apache.spark.examples.SparkPi
  mainApplicationFile: "local:///usr/lib/spark/examples/jars/spark-examples.jar"
  sparkVersion: "3.3.1"
  batchScheduler: "volcano"   #Note: You must specify the batch scheduler name as
 'volcano'
  restartPolicy:
    type: Never
  volumes:
    - name: "test-volume"
      hostPath:
        path: "/tmp"
        type: Directory
  driver:
    cores: 1
    coreLimit: "1200m"
    memory: "512m"
    labels:
      version: 3.3.1
    serviceAccount: emr-containers-sa-spark
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
  executor:
    cores: 1
    instances: 1
    memory: "512m"
    labels:
      version: 3.3.1
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
```

3. 使用以下命令提交 Spark 应用程序。此操作还会创建名为 `spark-pi` 的 `SparkApplication` 对象：

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

4. 使用以下命令检查 `SparkApplication` 对象的事件：

```
kubectl describe pods spark-pi-driver --namespace spark-operator
```

第一个 Pod 事件会显示 Volcano 已安排 Pod：

```
Type    Reason     Age   From                Message
----    ------     ----  ----                -------
Normal  Scheduled  23s   volcano             Successfully assigned default/spark-
pi-driver to integration-worker2
```

使用 Volcano 调度器和 **spark-submit** 运行 Spark 应用程序

1. 首先，完成 设置 Amazon EMR on EKS 的 spark-submit 小节中的步骤。必须在 Volcano 支持下构建 `spark-submit` 分配。有关更多信息，请参阅 Apache Spark 文档中 Using Volcano as Customized Scheduler for Spark on Kubernetes 的 Build 小节。
2. 设置以下环境变量的值：

```
export SPARK_HOME=spark-home
export MASTER_URL=k8s://Amazon-EKS-cluster-endpoint
```

3. 使用以下命令提交 Spark 应用程序：

```
$SPARK_HOME/bin/spark-submit \
 --class org.apache.spark.examples.SparkPi \
 --master $MASTER_URL \
 --conf spark.kubernetes.container.image=895885662937.dkr.ecr.us-
west-2.amazonaws.com/spark/emr-6.10.0:latest \
 --conf spark.kubernetes.authenticate.driver.serviceAccountName=spark \
 --deploy-mode cluster \
 --conf spark.kubernetes.namespace=spark-operator \
 --conf spark.kubernetes.scheduler.name=volcano \
 --conf spark.kubernetes.scheduler.volcano.podGroupTemplateFile=/path/to/podgroup-
template.yaml \
 --conf
 spark.kubernetes.driver.pod.featureSteps=org.apache.spark.deploy.k8s.features.VolcanoFeatu
 \
 --conf
 spark.kubernetes.executor.pod.featureSteps=org.apache.spark.deploy.k8s.features.VolcanoFea
 \
 local:///usr/lib/spark/examples/jars/spark-examples.jar 20
```

4. 使用以下命令检查 `SparkApplication` 对象的事件：

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

第一个 Pod 事件会显示 Volcano 已安排 Pod：

```
Type     Reason      Age    From                    Message
----     ------      ----   ----                    -------
Normal  Scheduled   23s    volcano                 Successfully assigned default/spark-
pi-driver to integration-worker2
```

在 EK YuniKorn S 上用作 Amazon EMR 上的 Apache Spark 的自定义调度程序

您可以在 Amazon EMR on EKS 上使用 Spark Operator 或 spark-submit，通过 Kubernetes 自定义调度器运行 Spark 任务。本教程介绍如何在自定义队列中使用 YuniKorn调度器运行 Spark 作业和群组调度。

概览

Apache YuniKorn 可以通过应用感知调度来帮助管理 Spark 调度，这样你就可以精细地控制资源配额和优先级。使用帮派排程时， YuniKorn 只有在可以满足应用程序的最低资源请求时，才会对应用程序进行调度。有关更多信息，请参阅 Apache YuniKorn 文档网站上的什么是帮派计划。

创建您的集群并进行设置 YuniKorn

请按照以下步骤部署 Amazon EKS 集群。 Amazon Web Services 区域 (`region`) 和可用区 (`availabilityZones`) 皆可更改。

1. 定义 Amazon EKS 集群：

```
cat <<EOF >eks-cluster.yaml
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: emr-eks-cluster
  region: eu-west-1

vpc:
  clusterEndpoints:
```

```
    publicAccess: true
    privateAccess: true

iam:
  withOIDC: true

nodeGroups:
  - name: spark-jobs
    labels: { app: spark }
    instanceType: m5.xlarge
    desiredCapacity: 2
    minSize: 2
    maxSize: 3
    availabilityZones: ["eu-west-1a"]
EOF
```

2. 创建集群：

```
eksctl create cluster -f eks-cluster.yaml
```

3. 创建要在其中执行 Spark 任务的命名空间 `spark-job`：

```
kubectl create namespace spark-job
```

4. 接下来，创建 Kubernetes 角色和角色绑定。这是 Spark 任务运行所用服务账户的必要项目。

   a. 定义 Spark 任务的服务账户、角色和角色绑定。

```
cat <<EOF >emr-job-execution-rbac.yaml
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: spark-sa
  namespace: spark-job
automountServiceAccountToken: false
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: spark-role
  namespace: spark-job
rules:
```

```
  - apiGroups: ["", "batch","extensions"]
    resources: ["configmaps","serviceaccounts","events","pods","pods/
exec","pods/log","pods/
portforward","secrets","services","persistentvolumeclaims"]
    verbs: ["create","delete","get","list","patch","update","watch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: spark-sa-rb
  namespace: spark-job
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: spark-role
subjects:
  - kind: ServiceAccount
    name: spark-sa
    namespace: spark-job
EOF
```

b. 使用以下命令应用 Kubernetes 角色和角色绑定的定义：

```
kubectl apply -f emr-job-execution-rbac.yaml
```

安装和设置 YuniKorn

1. 使用以下 kubectl 命令创建用于部署 Yunikorn 调度器的命名空间 `yunikorn`：

```
kubectl create namespace yunikorn
```

2. 执行以下 Helm 命令安装调度器：

```
helm repo add yunikorn https://apache.github.io/yunikorn-release
```

```
helm repo update
```

```
helm install yunikorn yunikorn/yunikorn --namespace yunikorn
```

使用 Spark 运算符运行带有 YuniKorn调度程序的 Spark 应用程序

1. 如果尚未完成设置，请按照以下各节中的步骤完成设置：

   a. 创建您的集群并进行设置 YuniKorn

   b. 安装和设置 YuniKorn

   c. 设置 Amazon EMR on EKS 的 Spark Operator

   d. 安装 Spark Operator

   运行 `helm install spark-operator-demo` 命令时包括以下参数：

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

2. 创建 `SparkApplication` 定义文件 `spark-pi.yaml`。

   要 YuniKorn 用作作业的调度器，您必须在应用程序定义中添加某些注释和标签。注释和标签会指定任务的队列和要使用的调度策略。

   在以下示例中，注释 `schedulingPolicyParameters` 为应用程序设置了分组调度。然后，该示例创建任务组或任务分组，指定在调度 Pod 开始任务执行之前必须可用的最小容量。最后，在任务组定义中指定使用带 `"app": "spark"` 标签的节点组，如 创建您的集群并进行设置 YuniKorn 小节所述。

   ```
   apiVersion: "sparkoperator.k8s.io/v1beta2"
   kind: SparkApplication
   metadata:
     name: spark-pi
     namespace: spark-job
   spec:
     type: Scala
     mode: cluster
     image: "895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.10.0:latest"
     imagePullPolicy: Always
     mainClass: org.apache.spark.examples.SparkPi
     mainApplicationFile: "local:///usr/lib/spark/examples/jars/spark-examples.jar"
     sparkVersion: "3.3.1"
     restartPolicy:
       type: Never
     volumes:
       - name: "test-volume"
   ```

```
      hostPath:
        path: "/tmp"
        type: Directory
  driver:
    cores: 1
    coreLimit: "1200m"
    memory: "512m"
    labels:
      version: 3.3.1
    annotations:
      yunikorn.apache.org/schedulingPolicyParameters: "placeholderTimeoutSeconds=30 
 gangSchedulingStyle=Hard"
      yunikorn.apache.org/task-group-name: "spark-driver"
      yunikorn.apache.org/task-groups: |-
        [{
            "name": "spark-driver",
            "minMember": 1,
            "minResource": {
              "cpu": "1200m",
              "memory": "1Gi"
            },
            "nodeSelector": {
              "app": "spark"
            }
          },
          {
            "name": "spark-executor",
            "minMember": 1,
            "minResource": {
              "cpu": "1200m",
              "memory": "1Gi"
            },
            "nodeSelector": {
              "app": "spark"
            }
        }]
    serviceAccount: spark-sa
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
  executor:
    cores: 1
    instances: 1
    memory: "512m"
```

```
    labels:
      version: 3.3.1
    annotations:
      yunikorn.apache.org/task-group-name: "spark-executor"
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
```

3. 使用以下命令提交 Spark 应用程序。此操作还会创建名为 `spark-pi` 的 `SparkApplication` 对象：

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

4. 使用以下命令检查 `SparkApplication` 对象的事件：

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

第一个 pod 事件将显示 YuniKorn 已调度 pod：

```
Type    Reason             Age   From                          Message
----    ------             ----  ----                          -------
Normal Scheduling         3m12s yunikorn   spark-operator/org-apache-spark-examples-
sparkpi-2a777a88b98b8a95-driver is queued and waiting for allocation
Normal GangScheduling     3m12s yunikorn   Pod belongs to the taskGroup spark-
driver, it will be scheduled as a gang member
Normal Scheduled          3m10s yunikorn   Successfully assigned spark
Normal PodBindSuccessful 3m10s yunikorn   Pod spark-operator/
Normal TaskCompleted      2m3s  yunikorn   Task spark-operator/
Normal Pulling            3m10s kubelet    Pulling
```

使用 YuniKorn调度器运行 Spark 应用程序 **`spark-submit`**

1. 首先，完成 设置 Amazon EMR on EKS 的 spark-submit 小节中的步骤。
2. 设置以下环境变量的值：

```
export SPARK_HOME=spark-home
export MASTER_URL=k8s://Amazon-EKS-cluster-endpoint
```

3. 使用以下命令提交 Spark 应用程序：

在以下示例中，注释 `schedulingPolicyParameters` 为应用程序设置了分组调度。然后，该示例创建任务组或任务分组，指定在调度 Pod 开始任务执行之前必须可用的最小容量。最后，在任务组定义中指定使用带 `"app": "spark"` 标签的节点组，如 创建您的集群并进行设置 YuniKorn 小节所述。

```
$SPARK_HOME/bin/spark-submit \
 --class org.apache.spark.examples.SparkPi \
 --master $MASTER_URL \
 --conf spark.kubernetes.container.image=895885662937.dkr.ecr.us-
west-2.amazonaws.com/spark/emr-6.10.0:latest \
 --conf spark.kubernetes.authenticate.driver.serviceAccountName=spark-sa \
 --deploy-mode cluster \
 --conf spark.kubernetes.namespace=spark-job \
 --conf spark.kubernetes.scheduler.name=yunikorn \
 --conf spark.kubernetes.driver.annotation.yunikorn.apache.org/
schedulingPolicyParameters="placeholderTimeoutSeconds=30 gangSchedulingStyle=Hard"
 \
 --conf spark.kubernetes.driver.annotation.yunikorn.apache.org/task-group-
name="spark-driver" \
 --conf spark.kubernetes.executor.annotation.yunikorn.apache.org/task-group-
name="spark-executor" \
 --conf spark.kubernetes.driver.annotation.yunikorn.apache.org/task-groups='[{
            "name": "spark-driver",
            "minMember": 1,
            "minResource": {
              "cpu": "1200m",
              "memory": "1Gi"
            },
            "nodeSelector": {
              "app": "spark"
            }
          },
          {
            "name": "spark-executor",
            "minMember": 1,
            "minResource": {
              "cpu": "1200m",
              "memory": "1Gi"
            },
            "nodeSelector": {
              "app": "spark"
            }
```

```
        }]' \
  local:///usr/lib/spark/examples/jars/spark-examples.jar 20
```

4. 使用以下命令检查 SparkApplication 对象的事件：

```
kubectl describe pod spark-driver-pod --namespace spark-job
```

第一个 pod 事件将显示 YuniKorn 已调度 pod：

```
Type    Reason             Age   From                          Message
----    ------             ----  ----                          -------
Normal Scheduling          3m12s yunikorn   spark-operator/org-apache-spark-examples-
sparkpi-2a777a88b98b8a95-driver is queued and waiting for allocation
Normal GangScheduling      3m12s yunikorn   Pod belongs to the taskGroup spark-
driver, it will be scheduled as a gang member
Normal Scheduled           3m10s yunikorn   Successfully assigned spark
Normal PodBindSuccessful 3m10s yunikorn     Pod spark-operator/
Normal TaskCompleted       2m3s  yunikorn   Task spark-operator/
Normal Pulling             3m10s kubelet    Pulling
```

Amazon EMR on EKS 安全性

云安全 Amazon 是重中之重。作为 Amazon 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 Amazon 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 — Amazon 负责保护在 Amazon 云中运行 Amazon 服务的基础架构。 Amazon 还为您提供可以安全使用的服务。作为的一部分，第三方审计师定期测试和验证我们安全的有效性。要了解适用于 Amazon EMR 的合规计划，请参阅按合规计划提供的划分的范围内的Amazon 服务。
- 云端安全-您的责任由您使用的 Amazon 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。

该文档帮助您了解如何在使用 Amazon EMR on EKS 时应用责任共担模式。以下主题说明如何配置 Amazon EMR on EKS 以实现您的安全性和合规性目标。您还将学习如何使用其他 Amazon 服务来帮助您监控和保护您的 EKS 资源上的 Amazon EMR。

主题

- Amazon EMR on EKS 安全最佳实践
- 数据保护
- 身份和访问管理
- 在 EKS 上使用 Amazon EMR 和 L Amazon ake Formation 进行精细的访问控制
- 日志记录和监控
- 将 Amazon S3 Access Grants 与 Amazon EMR on EKS 结合使用
- Amazon EMR on EKS 的合规性验证
- Amazon EMR on EKS 的恢复能力
- Amazon EMR on EKS 中的基础设施安全性
- 配置和漏洞分析
- 使用接口 VPC 端点连接到 Amazon EMR on EKS
- 为 Amazon EMR on EKS 设置跨账户访问权限

Amazon EMR on EKS 安全最佳实践

Amazon EMR on EKS 提供了在您开发和实施自己的安全策略时需要考虑的大量安全功能。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。

> **Note**
> 有关更多安全最佳实践，请参阅 Amazon EMR on EKS 安全最佳实践。

采用最低特权原则

Amazon EMR on EKS 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。这些执行角色通过 IAM 角色的信任策略映射到 Kubernetes 服务账户。Amazon EMR on EKS 在已注册的 Amazon EKS 命名空间中创建 Pod，用于执行用户提供的应用程序代码。运行应用程序代码的作业 pod 在连接到其他 Amazon 服务时扮演执行角色。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。

终端节点的访问控制列表

只能为已配置的 EKS 集群创建托管式终端节点，以至少在 VPC 的私有子网上使用。此配置限制对托管式终端节点创建的负载均衡器的访问，以便只能从您的 VPC 访问它们。为了进一步增强安全性，我们建议您使用这些负载均衡器配置安全组，以便它们可以将传入流量限制到一组选定的 IP 地址。

获取自定义镜像的最新安全更新

要通过 Amazon EMR on EKS 使用自定义镜像，您可以在镜像上安装任何二进制文件和库。您负责添加到镜像中的二进制文件的安全修补。Amazon EMR on EKS 镜像会使用最新的安全补丁程序进行定期修补。要获取最新映像，您必须在 Amazon EMR 的新基础映像版本发布时重新构建自定义映像。有关更多信息，请参阅Amazon EMR on EKS 版本和如何选择基础映像 URI 的详细信息。

限制 Pod 凭证访问

Kubernetes 支持将凭证分配给 Pod 的多种方法。预置多个凭证提供程序可能会增加您安全模型的复杂性。Amazon EMR on EKS 已采用服务账户的 IAM 角色（IRSA）作为注册 EKS 命名空间中的标准凭

证提供程序。不支持其他方法，包括 kube2iam、k iam 以及使用集群上运行的 EC2 实例的实例配置文件。

隔离不受信任的应用程序代码

Amazon EMR on EKS 不检查系统用户提交应用程序代码的完整性。如果您运行的多租户虚拟集群是使用多个执行角色进行配置的，而运行任意代码的不受信任的租户可使用这些角色来提交任务，则会存在恶意应用程序升级其权限的风险。在这种情况下，请考虑将具有相似权限的执行角色隔离到不同的虚拟集群中。

基于角色的访问控制（RBAC）权限

管理员应针对 Amazon EMR on EKS 托管式命名空间，来严格控制基于角色的访问控制（RBAC）权限。至少不应向 Amazon EMR on EKS 托管式命名空间中的任务提交者授予以下权限。

- Kubernetes RBAC 具有修改 configmap 的权限，因为 Amazon EMR on EKS 使用 Kubernetes configmaps 来生成具有托管式服务账户名称的托管式 Pod 模板。此属性应不可变。
- Kubernetes RBAC 具有可执行 Amazon EMR on EKS Pod 的权限，以避免对具有托管 SA 名称的托管 Pod 模板授予访问权限。此属性应不可变。此权限还可对挂载到 Pod 中的 JWT 令牌授予访问权限，然后可使用该令牌检索执行角色凭证。
- Kubernetes RBAC 创建容器的权限——防止用户使用 Kubernetes 创建 Pod， ServiceAccount 该容器可能映射到权限比用户更多的 IAM 角色。 Amazon
- Kubernetes RBAC 部署变异 webhook 的权限——以防止用户使用变异的 webhook 来改变亚马逊 EMR 在 EKS 上创建的 pod 的 Kubernetes 名称。 ServiceAccount
- Kubernetes RBAC 具有读取 Kubernetes 机密信息的权限，以防止用户读取存储在这些机密中的机密性数据。

限制对节点组 IAM 角色或实例配置文件凭证的访问

- 我们建议您为节点组的 IAM 角色分配最低 Amazon 权限。通过使用 EKS 工作线程节点的实例配置文件凭证所运行的代码，这有助于避免权限升级。
- 要完全阻止对 Amazon EMR on EKS 托管式命名空间中运行的所有 Pod 的实例配置文件凭证的访问，我们建议您在 EKS 节点上运行 `iptables` 命令。有关更多信息，请参阅限制访问 Amazon EC2 实例配置文件凭证。但是，正确确定服务账户 IAM 角色的范围非常重要，以便您的 Pod 具有所有必要的权限。例如，为节点 IAM 角色分配了从 Amazon ECR 提取容器镜像的权限。如果未向某个 Pod 分配这些权限，则该 Pod 无法从 Amazon ECR 提取容器镜像。VPC CNI 插件也需要更新。有关更多信息，请参阅演练：更新 VPC CNI 插件以便为服务账户使用 IAM 角色。

数据保护

分 Amazon 担责任模型适用于 EKS 上的 Amazon EMR 中的数据保护。如本模型所述 Amazon ，负责保护运行所有 Amazon 云的全球基础架构。您负责维护对托管在此基础设施上的内容的控制。此内容包括您所使用的 Amazon 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 Amazon 安全博客上的责任 Amazon 共担模型和 GDPR 博客文章。

出于数据保护目的，我们建议您使用 Amazon 身份与访问管理 (IAM) and Identity Access Management 来保护 Amazon 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）。
- 使用 SSL/TLS 与资源通信。 Amazon 建议使用 TLS 1.2 或更高版本。
- 使用设置 API 和用户活动日志 Amazon CloudTrail。
- 使用 Amazon 加密解决方案以及 Amazon 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 on EKS 加密选项对静态数据和传输中的数据进行加密。
- 如果您在 Amazon 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，在 EKS 或其他 Amazon 服务上使用 Amazon EMR 时。 Amazon CLI Amazon SDKs您输入到 Amazon EMR on EKS 或其它服务中的任何数据都可能被选取以包含在诊断日志中。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 Amazon 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。Amazon 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 Amazon 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

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。在您启用静态加密时，请指定 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）您可以使用 Per bucket encryption overrides (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。有关 Amazon S3 加密的更多信息，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 Amazon 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Amazon KMS 定价。

Amazon S3 服务器端加密

设置 Amazon S3 服务器端加密时，Amazon S3 在向磁盘写入数据时会在对象级别加密数据，并在访问数据时对数据进行解密。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 on EKS 中指定 SSE 时，您可以在两个不同的密钥管理系统之间进行选择：

- SSE-S3 - Amazon S3 为您管理密钥。
- SSE-KMS-您可以使用 Amazon KMS key 来设置适用于 EKS 上亚马逊 EMR 的政策。

客户提供密钥的 SSE（SSE-C）不能用于 Amazon EMR on EKS。

Amazon S3 客户端加密

对于 Amazon S3 客户端加密，Amazon S3 加密和解密过程在您的 EMR 集群上的 EMRFS 客户端中进行。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 Amazon 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。有关这些区别的更多信息，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。

> **Note**
>
> Amazon S3 CSE 仅确保与 Amazon S3 交换的 EMRFS 数据已加密；不确保集群实例卷上的所有数据都已加密。此外，由于 Hue 不使用 EMRFS，因此 Hue S3 文件浏览器写入 Amazon S3 的对象没有加密。

本地磁盘加密

Apache Spark 支持对写入本地磁盘的临时数据进行加密。其中包括随机文件、随机溢出以及存储在磁盘上的缓存和广播变量的数据块。它不包括使用`saveAsHadoopFile`或 APIs `saveAsTable`等对应用程序生成的输出数据进行加密。它也可能不包括用户明确创建的临时文件。有关更多信息，请参阅 Spark 文档中的本地存储加密。Spark 不支持本地磁盘上的加密数据，例如，当数据不适合内存时，执行程序进程将中间数据写入本地磁盘。保留到磁盘的数据的作用域为任务运行时，用于加密数据的密钥由 Spark 为每次任务运行动态生成。一旦 Spark 任务终止，任何其它进程都无法解密数据。

对于驱动程序和执行程序 Pod，您可以加密保留到挂载卷的静态数据。您可以在 Kubernetes 上使用三种不同的 Amazon 原生存储选项：E BS、EFS 和 For Lustre。FSx 这三者都使用服务托管式密钥或 Amazon KMS key进行静态加密。有关更多信息，请参阅 EKS 最佳实践指南。使用此方法，所有保留到挂载卷的数据都会加密。

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，请参阅旋转 Amazon KMS keys。

传输中加密

传输中加密会启用几种加密机制。这些是特定于应用程序的开源功能，可能因 Amazon EMR on EKS 版本而异。Amazon EMR on EKS 可以启用以下特定于应用程序的加密功能：

- Spark
  - 在 Amazon EMR 版本 5.9.0 及更高版本中，使用 AES-256 密码对 Spark 组件（例如，数据块传输服务和外部随机服务）之间的内部 RPC 通信进行加密。在早期版本中，内部 RPC 通信使用以 DIGEST-MD5 作为密码的 SASL 进行加密。
  - 使用 Spark 的 SSL 配置对与用户界面（如 Spark History Server 和支持 HTTPS 的文件服务器）的 HTTP 协议通信进行加密。有关更多信息，请参阅 Spark 文档中的 SSL 配置。

  有关更多信息，请参阅 Spark 安全设置。

- 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。
- 流式传输到 JDBC 或 ODBC 客户端的查询结果使用 TLS 进行加密。

身份和访问管理

Amazon Identity and Access Management (IAM) Amazon Web Services 服务 可帮助管理员安全地控制对 Amazon 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（具有权限）使用 Amazon EMR on EKS 资源。您可以使用 IAM Amazon Web Services 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
- Amazon EMR on EKS 如何与 IAM 配合使用
- 对 Amazon EMR on EKS 使用服务相关角色
- Amazon EMR on EKS 的托管式策略
- 将任务执行角色与 Amazon EMR on EKS 结合使用
- Amazon EMR on EKS 基于身份的策略示例
- 基于标签的访问控制策略
- 对 Amazon EMR on EKS 身份和访问进行故障排除

受众

您的使用方式 Amazon Identity and Access Management (IAM) 会有所不同，具体取决于您在 EKS 上的 Amazon EMR 中所做的工作。

服务用户 - 如果您使用 Amazon EMR on EKS 服务来完成任务，则您的管理员会为您提供所需的凭证和权限。随着您使用更多 Amazon EMR on EKS 功能来完成工作，您可能需要额外权限。了解如何管理访问权限有助于您向管理员请求适合的权限。如果您无法访问 Amazon EMR on EKS 中的功能，请参阅对 Amazon EMR on EKS 身份和访问进行故障排除。

服务管理员 – 如果您在公司负责管理 Amazon EMR on EKS 资源，您可能对 Amazon EMR on EKS 具有完全访问权限。您有责任确定您的服务用户应访问哪些 Amazon EMR on EKS 功能和资源。然后，

您必须向 IAM 管理员提交请求以更改服务用户的权限。请查看该页面上的信息以了解 IAM 的基本概念。要了解有关您的公司如何将 IAM 与 Amazon EMR on EKS 搭配使用的更多信息，请参阅Amazon EMR on EKS 如何与 IAM 配合使用。

IAM 管理员 – 如果您是 IAM 管理员，您可能希望了解如何编写策略以管理对 Amazon EMR on EKS 的访问的详细信息。要查看您可在 IAM 中使用的 Amazon EMR on EKS 基于身份的策略示例，请参阅Amazon EMR on EKS 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 Amazon 使用身份凭证登录的方式。您必须以 IAM 用户身份或通过担 Amazon Web Services 账户根用户任 IAM 角色进行身份验证（登录 Amazon）。

如果您 Amazon 以编程方式访问，则会 Amazon 提供软件开发套件 (SDK) 和命令行接口 (CLI)，以便使用您的凭据对请求进行加密签名。如果您不使用 Amazon 工具，则必须自己签署请求。有关使用推荐的方法自行签署请求的更多信息，请参阅《IAM 用户指南》中的用于签署 API 请求的Amazon 签名版本 4。

无论使用何种身份验证方法，您都可能需要提供其他安全信息。例如， Amazon 建议您使用多重身份验证 (MFA) 来提高账户的安全性。要了解更多信息，请参阅《IAM 用户指南》中的 IAM 中的Amazon 多重身份验证。

Amazon Web Services 账户 root 用户

创建时 Amazon Web Services 账户，首先要有一个登录身份，该身份可以完全访问账户中的所有资源 Amazon Web Services 服务 和资源。此身份被称为 Amazon Web Services 账户 root 用户，使用您创建账户时使用的电子邮件地址和密码登录即可访问该身份。强烈建议您不要使用根用户执行日常任务。保护好根用户凭证，并使用这些凭证来执行仅根用户可以执行的任务。有关要求您以根用户身份登录的任务的完整列表，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（包括需要管理员访问权限的用户）使用与身份提供商的联合身份验证 Amazon Web Services 服务 通过临时证书进行访问。

联合身份是指您的企业用户目录、Web 身份提供商、Identity C 或者任何使用 Amazon Web Services 服务 通过身份源提供的凭据进行访问的用户。 Amazon Directory Service当联合身份访问时 Amazon Web Services 账户，他们将扮演角色，角色提供临时证书。

IAM 用户和群组

I AM 用户是您 Amazon Web Services 账户 内部对个人或应用程序具有特定权限的身份。在可能的情况下，我们建议使用临时凭证，而不是创建具有长期凭证（如密码和访问密钥）的 IAM 用户。但是，如果您有一些特定的使用场景需要长期凭证以及 IAM 用户，建议您轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的对于需要长期凭证的用例，应在需要时更新访问密钥。

IAM 组是一个指定一组 IAM 用户的身份。您不能使用组的身份登录。您可以使用组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，您可以拥有一个名为的群组，IAMAdmins并向该群组授予管理 IAM 资源的权限。

用户与角色不同。用户唯一地与某个人员或应用程序关联，而角色旨在让需要它的任何人代入。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅《IAM 用户指南》中的 IAM 用户的使用案例。

IAM 角色

I AM 角色是您内部具有特定权限 Amazon Web Services 账户 的身份。它类似于 IAM 用户，但与特定人员不关联。要在中临时担任 IAM 角色 Amazon Web Services Management Console，您可以从用户切换到 IAM 角色（控制台）。您可以通过调用 Amazon CLI 或 Amazon API 操作或使用自定义 URL 来代入角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的代入角色的方法。

具有临时凭证的 IAM 角色在以下情况下很有用：

- 联合用户访问：要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关用于联合身份验证的角色的信息，请参阅《IAM 用户指南》中的针对第三方身份提供商创建角色（联合身份验证）。
- 临时 IAM 用户权限：IAM 用户可代入 IAM 用户或角色，以暂时获得针对特定任务的不同权限。
- 跨账户存取：您可以使用 IAM 角色以允许不同账户中的某个人（可信主体）访问您的账户中的资源。角色是授予跨账户访问权限的主要方式。但是，对于某些资源 Amazon Web Services 服务，您可以将策略直接附加到资源（而不是使用角色作为代理）。要了解用于跨账户访问的角色和基于资源的策略之间的差别，请参阅 IAM 用户指南中的 IAM 中的跨账户资源访问。
- 跨服务访问 — 有些 Amazon Web Services 服务 使用其他 Amazon Web Services 服务服务中的功能。例如，当您在服务中拨打电话时，该服务通常会在 Amazon 中运行应用程序 EC2 或在 Amazon S3 中存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。
  - 转发访问会话 (FAS) — 当您使用 IAM 用户或角色在中执行操作时 Amazon，您被视为委托人。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用

调用委托人的权限以及 Amazon Web Services 服务 向下游服务发出请求的请求。 Amazon Web Services 服务只有当服务收到需要与其他 Amazon Web Services 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两项操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。

  - 服务角色 - 服务角色是服务代表您在您的账户中执行操作而分派的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 Amazon Web Services 服务委派权限的角色。
  - 服务相关角色-服务相关角色是一种与服务相关联的服务角色。 Amazon Web Services 服务服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 Amazon Web Services 账户 ，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。

- 在 A@@ mazon 上运行的应用程序 EC2 — 您可以使用 IAM 角色管理在 EC2 实例上运行并发出 Amazon CLI 或 Amazon API 请求的应用程序的临时证书。这比在 EC2 实例中存储访问密钥更可取。要为 EC2 实例分配 Amazon 角色并使其可供其所有应用程序使用，您需要创建一个附加到该实例的实例配置文件。实例配置文件包含该角色，并允许在 EC2 实例上运行的程序获得临时证书。有关更多信息，请参阅 IAM 用户指南中的使用 IAM 角色向在 A mazon EC2 实例上运行的应用程序授予权限。

使用策略管理访问

您可以 Amazon 通过创建策略并将其附加到 Amazon 身份或资源来控制中的访问权限。策略是其中的一个对象 Amazon ，当与身份或资源关联时，它会定义其权限。 Amazon 在委托人（用户、root 用户或角色会话）发出请求时评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略都以 JSON 文档的 Amazon 形式存储在中。有关 JSON 策略文档的结构和内容的更多信息，请参阅 IAM 用户指南中的 JSON 策略概览。

管理员可以使用 Amazon 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。

默认情况下，用户和角色没有权限。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。管理员随后可以向角色添加 IAM 策略，用户可以代入角色。

IAM 策略定义操作的权限，无关乎您使用哪种方法执行操作。例如，假设您有一个允许 `iam:GetRole` 操作的策略。拥有该策略的用户可以从 Amazon Web Services Management Console Amazon CLI、或 Amazon API 获取角色信息。

基于身份的策略

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》中的使用客户托管策略定义自定义 IAM 权限。

基于身份的策略可以进一步归类为内联策略或托管式策略。内联策略直接嵌入单个用户、组或角色中。托管策略是独立的策略，您可以将其附加到中的多个用户、群组和角色 Amazon Web Services 账户。托管策略包括 Amazon 托管策略和客户托管策略。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。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 Amazon Web Services 服务。

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 Amazon 托管策略。

访问控制列表 (ACLs)

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。

Amazon S3 和 Amazon VPC 就是支持的服务示例 ACLs。 Amazon WAF要了解更多信息 ACLs，请参阅《亚马逊简单存储服务开发者指南》中的访问控制列表 (ACL) 概述。

其他策略类型

Amazon 支持其他不太常见的策略类型。这些策略类型可以设置更常用的策略类型向您授予的最大权限。

- 权限边界：权限边界是一个高级特征，用于设置基于身份的策略可以为 IAM 实体（IAM 用户或角色）授予的最大权限。您可为实体设置权限边界。这些结果权限是实体基于身份的策略及其权限边界的交集。在 `Principal` 中指定用户或角色的基于资源的策略不受权限边界限制。任一项策略中的显式拒绝将覆盖允许。有关权限边界的更多信息，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)- SCPs 是指定组织或组织单位 (OU) 的最大权限的 JSON 策略 Amazon Organizations。 Amazon Organizations 是一项用于对您的企业拥有的多 Amazon Web Services 账户 项进行分组和集中管理的服务。如果您启用组织中的所有功能，则可以将服务控制策略 (SCPs) 应用于您的任何或所有帐户。SCP 限制成员账户中的实体（包括每个 Amazon Web Services 账户根用户实体）的权限。有关 Organization SCPs s 和的更多信息，请参阅《Amazon Organizations 用户指南》中的服务控制策略。
- 资源控制策略 (RCPs) — RCPs 是 JSON 策略，您可以使用它来设置账户中资源的最大可用权限，而无需更新附加到您拥有的每个资源的 IAM 策略。RCP 限制成员账户中资源的权限，并可能影响身份（包括身份）的有效权限 Amazon Web Services 账户根用户，无论这些身份是否属于您的组织。有关 Organizations 的更多信息 RCPs，包括 Amazon Web Services 服务 该支持的列表 RCPs，请参阅《Amazon Organizations 用户指南》中的资源控制策略 (RCPs)。
- 会话策略：会话策略是当您以编程方式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。结果会话的权限是用户或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资源的策略。任一项策略中的显式拒绝将覆盖允许。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 Amazon 确定是否允许请求，请参阅 IAM 用户指南中的策略评估逻辑。

Amazon EMR on EKS 如何与 IAM 配合使用

在使用 IAM 管理对 Amazon EMR on EKS 的访问权限之前，您应该了解哪些 IAM 功能可用于 Amazon EMR on EKS。

将 IAM 功能与 Amazon EMR on EKS 一起使用

IAM 功能	Amazon EMR on EKS 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
策略条件键	是

IAM 功能	Amazon EMR on EKS 支持
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
服务相关角色	是

要全面了解 EKS 和其他 Amazon 服务上的 Amazon EMR 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Amazon 服务。

Amazon EMR on EKS 的基于身份的策略

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。您无法在基于身份的策略中指定主体，因为它适用于其附加的用户或角色。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。

Amazon EMR on EKS 基于身份的策略示例

要查看 Amazon EMR on EKS 基于身份的策略的示例，请参阅Amazon EMR on EKS 基于身份的策略示例。

Amazon EMR on EKS 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 Amazon Web Services 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。将跨账户主体添加到基于资源的策略只是建立信任关系工作的一半而已。当委托人和资源处于不同位置时 Amazon Web Services 账户，可信账户中的 IAM 管理员还必须向委托人实体（用户或角色）授予访问资源的权限。他们通过将基于身份的策略附加到实体以授予权限。但是，如果基于资源的策略向同一个账户中的主体授予访问权限，则不需要额外的基于身份的策略。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。

Amazon EMR on EKS 的策略操作

支持策略操作：是

管理员可以使用 Amazon 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。策略操作通常与关联的 Amazon API 操作同名。有一些例外情况，例如没有匹配 API 操作的仅限权限 操作。还有一些操作需要在策略中执行多个操作。这些附加操作称为相关操作。

在策略中包含操作以授予执行关联操作的权限。

要查看 Amazon EMR on EKS 操作的列表，请参阅《服务授权参考》中的 Amazon EMR on EKS 的操作、资源和条件键。

Amazon EMR on EKS 中的策略操作在操作前面使用以下前缀：

```
emr-containers
```

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-containers:action1",
      "emr-containers:action2"
         ]
```

要查看 Amazon EMR on EKS 基于身份的策略的示例，请参阅Amazon EMR on EKS 基于身份的策略示例。

Amazon EMR on EKS 的策略资源

支持策略资源：是

管理员可以使用 Amazon 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。语句必须包含 `Resource` 或 `NotResource` 元素。作为最佳实践，请使用其 Amazon 资源名称（ARN）指定资源。对于支持特定资源类型（称为资源级权限）的操作，您可以执行此操作。

对于不支持资源级权限的操作（如列出操作），请使用通配符（*）指示语句应用于所有资源。

```
"Resource": "*"
```

要查看 EKS 资源类型上的 Amazon EMR 及其列表 ARNs，请参阅《服务授权参考》中的 Amazon EMR 在 EKS 上定义的资源。要了解可以为每个资源的 ARN 指定哪些操作，请参阅 Amazon EMR on EKS 的操作、资源和条件键。

要查看 Amazon EMR on EKS 基于身份的策略的示例，请参阅Amazon EMR on EKS 基于身份的策略示例。

Amazon EMR on EKS 的策略条件键

支持特定于服务的策略条件键：是

管理员可以使用 Amazon 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。

在 `Condition` 元素（或 `Condition` 块）中，可以指定语句生效的条件。`Condition` 元素是可选的。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。

如果您在一个语句中指定多个 `Condition` 元素，或在单个 `Condition` 元素中指定多个键，则 Amazon 使用逻辑 `AND` 运算评估它们。如果您为单个条件键指定多个值，则使用逻辑`OR`运算来 Amazon 评估条件。在授予语句的权限之前必须满足所有的条件。

在指定条件时，您也可以使用占位符变量。例如，只有在使用 IAM 用户名标记 IAM 用户时，您才能为其授予访问资源的权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM 策略元素：变量和标签。

Amazon 支持全局条件密钥和特定于服务的条件密钥。要查看所有 Amazon 全局条件键，请参阅 IAM 用户指南中的Amazon 全局条件上下文密钥。

要查看 Amazon EMR on EKS 条件键的列表，并了解您可以对哪些操作和资源使用条件键，请参阅《服务授权参考》中的 Amazon EMR on EKS 的操作、资源和条件键。

要查看 Amazon EMR on EKS 基于身份的策略的示例，请参阅Amazon EMR on EKS 基于身份的策略示例。

EKS 上的 Amazon EMR 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。

使用 Amazon EMR on EKS 基于属性的访问控制（ABAC）

支持 ABAC（策略中的标签）	是

基于属性的访问控制 (ABAC) 是一种授权策略，该策略基于属性来定义权限。在中 Amazon，这些属性称为标签。您可以将标签附加到 IAM 实体（用户或角色）和许多 Amazon 资源。标记实体和资源是 ABAC 的第一步。然后设计 ABAC 策略，以在主体的标签与他们尝试访问的资源标签匹配时允许操作。

ABAC 在快速增长的环境中非常有用，并在策略管理变得繁琐的情况下可以提供帮助。

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）。

使用 Amazon EMR on EKS 的临时凭证

支持临时凭证：是

当你使用临时证书登录时，有些 Amazon Web Services 服务 不起作用。有关更多信息，包括哪些 Amazon Web Services 服务 适用于临时证书，请参阅 IAM 用户指南中的Amazon Web Services 服务与 IA M 配合使用的信息。

如果您使用除用户名和密码之外的任何方法登录，则 Amazon Web Services Management Console 使用的是临时证书。例如，当您 Amazon 使用公司的单点登录 (SSO) 链接进行访问时，该过程会自动创建临时证书。当您以用户身份登录控制台，然后切换角色时，您还会自动创建临时凭证。有关切换角色的更多信息，请参阅《IAM 用户指南》中的从用户切换到 IAM 角色（控制台）。

您可以使用 Amazon CLI 或 Amazon API 手动创建临时证书。然后，您可以使用这些临时证书进行访问 Amazon。 Amazon 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 中的临时安全凭证。

Amazon EMR on EKS 的跨服务主要权限

支持转发访问会话（FAS）：是

当您使用 IAM 用户或角色在中执行操作时 Amazon，您被视为委托人。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用调用委托人的权限以及 Amazon Web Services 服务 向下游服务发出请求的请求。 Amazon Web Services 服务只有当服务收到需要与其他 Amazon Web Services 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两项操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。

Amazon EMR on EKS 的服务角色

支持服务角色	否

Amazon EMR on EKS 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，请参阅使用 IAM 的Amazon 服务。在表中查找服务相关角色列中包含 Yes 的表。选择是链接以查看该服务的服务相关角色文档。

对 Amazon EMR on EKS 使用服务相关角色

EKS 上的 Amazon EMR 使用 Amazon Identity and Access Management (IAM) 服务相关角色。服务相关角色是一种独特类型的 IAM 角色，它与 Amazon EMR on EKS 直接相关。服务相关角色由 Amazon EMR 在 EKS 上预定义，包括该服务代表您调用 Amazon 其他服务所需的所有权限。

服务相关角色可让您更轻松地设置 Amazon EMR on EKS，因为您不必手动添加必要的权限。Amazon EMR on EKS 定义其服务相关角色的权限，除非另外定义，否则只有 Amazon EMR on EKS 可以承担该角色。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它 IAM 实体的权限策略。

只有在首先删除相关资源后，您才能删除服务相关角色。这将保护您的 Amazon EMR on EKS 资源，因为您不会无意中删除对资源的访问权限。

有关支持服务关联的角色的其它服务的信息，请参阅与 IAM 配合使用的Amazon 服务，并查找 Service-Linked Role (服务相关角色) 列为 Yes (是) 的服务。选择是和链接，查看该服务的服务相关角色文档。

Amazon EMR on EKS 的服务相关角色权限

EKS 上的 Amazon EMR 使用名为的服务相关角色。AWSServiceRoleForAmazonEMRContainers

`AWSServiceRoleForAmazonEMRContainers` 服务相关角色信任以下服务代入该角色：

- `emr-containers.amazonaws.com`

角色权限策略 `AmazonEMRContainersServiceRolePolicy` 允许 Amazon EMR on EKS 完成对指定资源的一组操作，如以下策略声明所示。

> **Note**
>
> 托管策略内容会发生变化，因此此处显示的政策可能有所变化 out-of-date。在《Amazon 托管 up-to-date政策参考指南》EMRContainersServiceRolePolicy中查看 Amazon 上最多的政策文档。

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

```
            "Action": [
                "eks:DescribeCluster",
                "eks:ListNodeGroups",
                "eks:DescribeNodeGroup",
                "ec2:DescribeRouteTables",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "elasticloadbalancing:DescribeInstanceHealth",
                "elasticloadbalancing:DescribeLoadBalancers",
                "elasticloadbalancing:DescribeTargetGroups",
                "elasticloadbalancing:DescribeTargetHealth",
                "eks:ListPodIdentityAssociations",
                "eks:DescribePodIdentityAssociation"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "acm:ImportCertificate",
                "acm:AddTagsToCertificate"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/emr-container:endpoint:managed-certificate": "true"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "acm:DeleteCertificate"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/emr-container:endpoint:managed-certificate":
 "true"
                }
            }
        }
    ]
```

```
}
```

您必须配置权限，允许 IAM 实体（如用户、组或角色）创建、编辑或删除服务相关角色。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 Amazon EMR on EKS 创建服务相关角色

您无需手动创建服务相关角色。当您创建虚拟集群时，Amazon EMR on EKS 会为您创建服务相关角色。

如果您删除该服务相关角色，然后需要再次创建，您可以使用相同流程在账户中重新创建此角色。当您创建虚拟集群时，Amazon EMR on EKS 会再次为您创建服务相关角色。

您也可以使用 IAM 控制台为 Amazon EMR on EKS 使用案例创建服务相关角色。在 Amazon CLI 或 Amazon API 中，使用服务名称创建服务相关角色。`emr-containers.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，可以使用同样的过程再次创建角色。

为 Amazon EMR on EKS 编辑服务相关角色

Amazon EMR on EKS 不允许您编辑 `AWSServiceRoleForAmazonEMRContainers` 服务相关角色。创建服务相关角色后，您将无法更改角色的名称，因为可能有多种实体引用该角色。但是可以使用 IAM 编辑角色描述。有关更多信息，请参阅《IAM 用户指南》中的编辑服务相关角色。

删除适用于 Amazon EMR on EKS 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功能或服务，我们建议您删除该角色。这样就没有未被主动监控或维护的未使用实体。但是，必须先清除服务相关角色的资源，然后才能手动删除它。

> **Note**
> 如果在您试图删除资源时，Amazon EMR on EKS 服务正在使用该角色，则删除操作可能会失败。如果发生这种情况，请等待几分钟后重试。

要删除 **`AWSServiceRoleForAmazonEMRContainers`** 使用的 Amazon EMR on EKS 资源

1. 打开 Amazon EMR 控制台。
2. 选择一个虚拟集群。

3. 在 `Virtual Cluster` 页面上，选择 Delete (删除)。
4. 对您账户中的任何其它虚拟集群重复此步骤。

使用 IAM 手动删除服务相关角色

使用 IAM 控制台 Amazon CLI、或 Amazon API 删除`AWSServiceRoleForAmazonEMRContainers`服务相关角色。有关更多信息，请参见《IAM 用户指南》中的删除服务相关角色。

Amazon EMR on EKS 服务相关角色支持的区域

Amazon EMR on EKS 支持在该服务可用的所有区域中使用服务相关角色。有关更多信息，请参阅 Amazon EMR on EKS 服务端点和限额。

Amazon EMR on EKS 的托管式策略

查看自 2021 年 3 月 1 日以来在 EKS 上对亚马逊 EMR Amazon 托管政策更新的详细信息。

更改	描述	日期
`AmazonEMRContainersServiceRolePolicy` -添加了读取权限以列出集群中的 EKS pod 身份关联，并添加了另一项读取权限以返回有关集群中 pod 身份关联的描述性信息。有关更多信息，请参阅 Amazon EMRContainers ServiceRolePolicy。	策略中添加了以下权限：`eks:ListPodIdentityAssociations` 、`eks:DescribePodIdentityAssociation` 。	2023 年 2 月 3 日
`AmazonEMRContainersServiceRolePolicy` – 增加了描述并列出 Amazon EKS 节点组以及描述负载均衡器目标组和目标运行状况的权限。	以下权限已添加到策略：`eks:ListNodeGroups` 、`eks:DescribeNodeGroup` 、`elasticloadbalancing:DescribeTargetG`	2023 年 3 月 13 日

更改	描述	日期
	roups 、elasticloadbalancing:DescribeTargetHealth 。	
AmazonEMRContainersServiceRolePolicy -增加了在中导入和删除证书的权限 Amazon Certificate Manager。	以下权限已添加到策略：acm:ImportCertificate 、acm:AddTagsToCertificate 、acm:DeleteCertificate 。	2021 年 12 月 3 日
Amazon EMR on EKS 已开始跟踪更改	EKS 上的 Amazon EMR 开始跟踪其 Amazon 托管策略的变更。	2021 年 3 月 1 日

将任务执行角色与 Amazon EMR on EKS 结合使用

要使用 StartJobRun 命令在 EKS 集群上提交任务运行，则必须首先引导任务执行角色与虚拟集群一起使用。有关更多信息，请参阅设置 Amazon EMR on EKS中的创建任务执行角色。此外，您也可以按照 Amazon EMR on EKS 研讨会的为任务执行创建 IAM 角色部分中的说明进行操作。

任务执行角色的信任策略中必须包含以下权限。

```
  {
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::AWS_ACCOUNT_ID:oidc-provider/OIDC_PROVIDER"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringLike": {
          "OIDC_PROVIDER:sub": "system:serviceaccount:NAMESPACE:emr-containers-sa-*-
*-AWS_ACCOUNT_ID-BASE36_ENCODED_ROLE_NAME"
        }
      }
    }
  ]
}
```

前述示例中的信任策略仅向 Amazon EMR 托管的 Kubernetes 服务账户授予权限，并且服务账户的名称与 emr-containers-sa-*-*-*AWS_ACCOUNT_ID*-*BASE36_ENCODED_ROLE_NAME* 模式相匹配。具有此模式的服务账户将在任务提交时自动创建，作用域为提交任务的命名空间。此信任策略允许这些服务账户担任执行角色并获取执行角色的临时凭证。不同 Amazon EKS 集群的服务账户，或相同 EKS 集群内不同命名空间的服务账户，都不能担任执行角色。

您可以运行以下命令来自动更新上述给定格式的信任策略。

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

控制访问执行角色

Amazon EKS 集群管理员可以创建多租户 Amazon EMR on EKS 虚拟集群，IAM 管理员可以向其添加多个执行角色。因为不受信任的租户可以使用这些执行角色来提交运行任意代码的任务，所以您可能需要对这些租户加以限制，让他们无法运行代码来为一个或多个执行角色分配权限。要限制添加到 IAM 身份的 IAM policy，IAM 管理员可使用可选 Amazon 资源名称（ARN）条件键 emr-containers:ExecutionRoleArn。此条件接受具有虚拟集群权限的执行角色 ARNs 列表，如以下权限策略所示。

```
  {
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "emr-containers:StartJobRun",
      "Resource": "arn:aws:emr-containers:REGION:AWS_ACCOUNT_ID:/
virtualclusters/VIRTUAL_CLUSTER_ID",
      "Condition": {
        "ArnEquals": {
          "emr-containers:ExecutionRoleArn": [
            "execution_role_arn_1",
            "execution_role_arn_2",
            ...
          ]
        }
```

```
        }
      }
    ]
  }
```

如果您想允许以特定前缀开头（如 `MyRole`）的所有执行角色，则条件运算符 `ArnEquals` 可以替换为 `ArnLike` 运算符，并且条件中的 `execution_role_arn` 值可以替换为通配符 `*`。例如，`arn:aws:iam::AWS_ACCOUNT_ID:role/MyRole*`。其余 ARN 条件键也受支持。

> **Note**
>
> 您无法在 Amazon EMR on EKS 上根据标签或属性授予执行角色权限。Amazon EMR on EKS 目前不支持对执行角色应用基于标签的访问控制（TBAC）或基于属性的访问控制（ABAC）。

Amazon EMR on EKS 基于身份的策略示例

默认情况下，用户和角色没有创建或修改 Amazon EMR on EKS 资源的权限。他们也无法使用 Amazon Web Services Management Console、 Amazon Command Line Interface (Amazon CLI) 或 Amazon API 执行任务。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。管理员随后可以向角色添加 IAM 策略，用户可以代入角色。

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 在 EKS 上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 E KS 上的 Amazon EMR 的操作、资源和条件密钥。 ARNs

主题

- 策略最佳实践
- 使用 Amazon EMR on EKS 控制台
- 允许用户查看他们自己的权限

策略最佳实践

基于身份的策略确定某个人是否可以创建、访问或删除您账户中的 Amazon EMR on EKS 资源。这些操作可能会使 Amazon Web Services 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 Amazon 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Amazon 托管策略。它们在你的版本中可用 Amazon Web Services 账户。我们建议您通过定义针对您的用例的 Amazon 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 Amazon 托管式策略或工作职能的Amazon 托管式策略。
- 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 Amazon Web Services 服务，例如 Amazon 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 Amazon Web Services 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。

使用 Amazon EMR on EKS 控制台

要访问 Amazon EMR on EKS 控制台，您必须具有一组最低的权限。这些权限必须允许您列出和查看有关您 Amazon Web Services 账户账户中 Amazon EMR on EKS 资源的详细信息。如果创建比必需的最低权限更为严格的基于身份的策略，对于附加了该策略的实体（用户或角色），控制台将无法按预期正常运行。

对于仅调用 Amazon CLI 或 Amazon API 的用户，您无需为其设置最低控制台权限。相反，只允许访问与其尝试执行的 API 操作相匹配的操作。

为确保用户和角色仍然可以在 EKS 控制台上使用 Amazon EMR，还要将 EKS 上的 Amazon EMR `ConsoleAccess` 或`ReadOnly` Amazon 托管策略附加到实体。有关更多信息，请参阅《IAM 用户指南》中的为用户添加权限。

允许用户查看他们自己的权限

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 Amazon CLI 或 Amazon 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": "*"
        }
    ]
}
```

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，以便根据标签控制对虚拟集群和任务运行的访问。有关标记的更多信息，请参阅为您的 Amazon EMR on EKS 资源添加标签。

以下示例说明将条件运算符与 Amazon EMR on EKS 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 policy 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。

以下示例演示基于身份的权限策略，用于控制允许对 Amazon EMR on EKS 虚拟集群执行的操作。

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 与标签部门的值进行匹配。如果标签部分尚未添加到虚拟集群或不包含值 dev，则策略不会应用，并且此策略将不允许这些操作。如果任何其它策略声明允许操作，则用户只能使用标签中包含此值的虚拟集群。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:DescribeVirtualCluster"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/department": "dev"
        }
      }
    }
  ]
}
```

您也可以使用条件运算符指定多个标签值。例如，要运行其 department 标签包含值 dev 或 test 的虚拟集群上的操作，您可以将上一个示例中的条件数据块替换为以下内容。

```
"Condition": {
        "StringEquals": {
          "aws:ResourceTag/department": ["dev", "test"]
        }
      }
```

创建资源时需要标签

在以下示例中，创建虚拟集群时需要应用标签。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:CreateVirtualCluster"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/department": "dev"
        }
      }
    }
  ]
}
```

以下策略声明仅允许用户在集群具有 department 标签（可包含任何值）的情况下创建虚拟集群。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:CreateVirtualCluster"
      ],
      "Resource": "*",
```

```
        "Condition": {
          "Null": {
            "aws:RequestTag/department": "false"
          }
        }
      }
    ]
}
```

拒绝添加和删除标签的访问权限

此策略旨在拒绝用户在添加了 department 标签（包含 dev 值）的虚拟集群中添加或删除任何标签的权限。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "emr-containers:TagResource",
        "emr-containers:UntagResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:ResourceTag/department": "dev"
        }
      }
    }
  ]
}
```

对 Amazon EMR on EKS 身份和访问进行故障排除

可以使用以下信息，以帮助您诊断和修复在使用 Amazon EMR on EKS 和 IAM 时可能遇到的常见问题。

主题

- 我无权在 Amazon EMR on EKS 中执行操作
- 我无权执行 iam：PassRole

- 我想允许 Amazon 账户之外的人通过 EKS 资源访问我的 Amazon EMR

我无权在 Amazon EMR on EKS 中执行操作

如果 Amazon Web Services Management Console 告诉您您无权执行某项操作，则必须联系管理员寻求帮助。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查看有关虚构 *my-example-widget* 资源的详细信息，但不拥有虚构 `emr-containers:`*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-
containers: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-containers:`*GetWidget* 操作访问 *my-example-widget* 资源。

我无权执行 iam：PassRole

如果您收到一个错误，指明您无权执行 `iam:PassRole` 操作，则必须更新策略以允许您将角色传递给 Amazon EMR on EKS。

有些 Amazon Web Services 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 on EKS 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 Amazon 管理员。您的管理员是提供登录凭证的人。

我想允许 Amazon 账户之外的人通过 EKS 资源访问我的 Amazon EMR

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 on EKS 是否支持这些功能，请参阅Amazon EMR on EKS 如何与 IAM 配合使用。
- 要了解如何提供对您拥有的资源的访问权限 Amazon Web Services 账户 ，请参阅 IAM 用户指南中的向您拥有 Amazon Web Services 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 Amazon Web Services 账户，请参阅 IAM 用户指南中的向第三方提供访问权限。 Amazon Web Services 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。

在 EKS 上使用 Amazon EMR 和 L Amazon ake Formation 进行精细的访问控制

在 Amazon EMR 7.7 及更高版本中，您可以利用 Amazon Lake Formation 对由 Amazon S3 存储桶支持的 Gl Amazon ue 数据目录表应用精细的访问控制。此功能允许您在 EKS Spark Jobs 上的 Amazon EMR 中为读取查询配置表、行、列和单元格级别的访问控制。

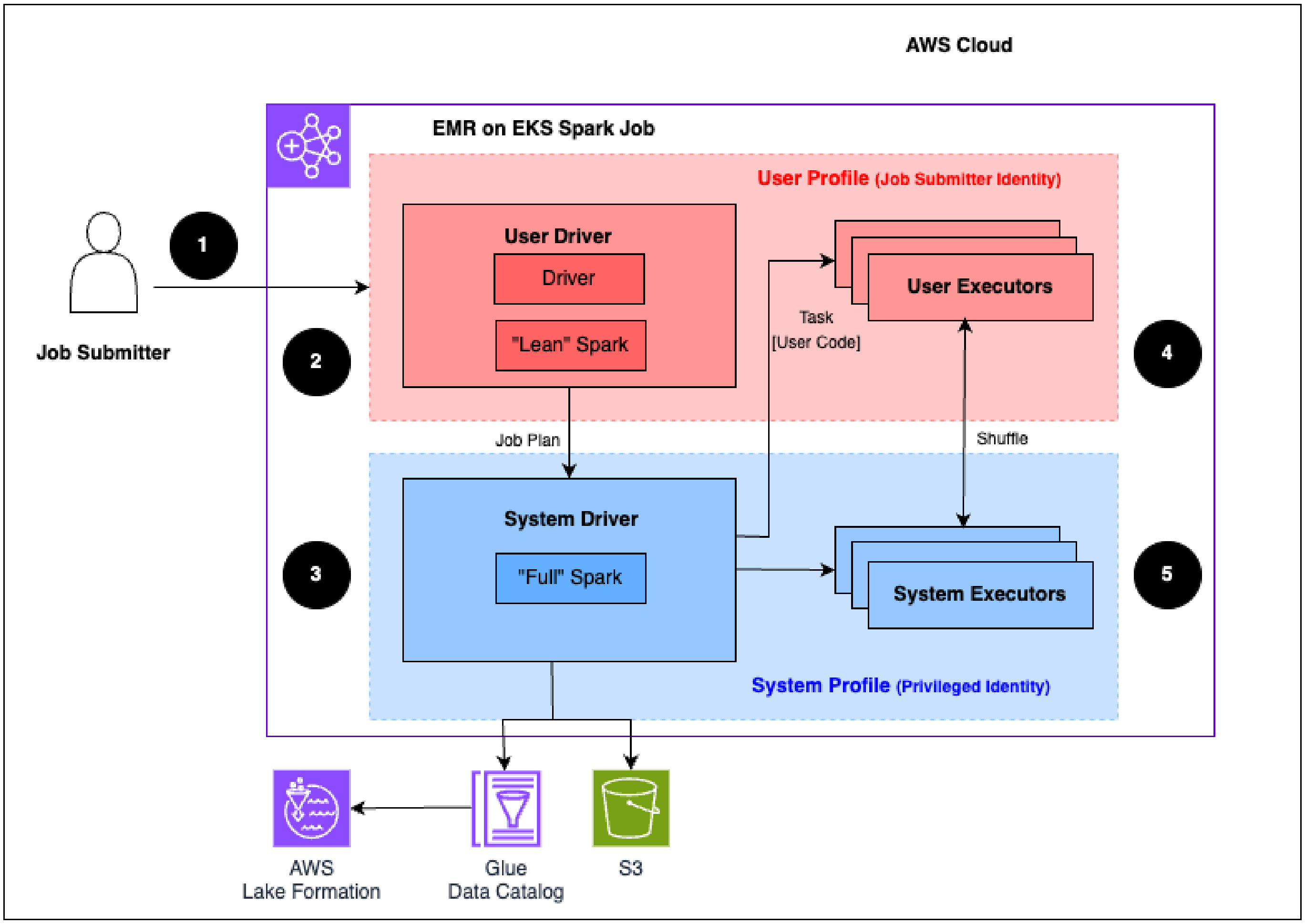
主题

- EKS 上的亚马逊 EMR 如何与 Lake Formation 配合使用 Amazon
- 在 EKS 上使用亚马逊 EMR 启用 Lake Formation
- 注意事项和限制
- 故障排除

EKS 上的亚马逊 EMR 如何与 Lake Formation 配合使用 Amazon

在 EKS 上使用 Amazon EMR 和 Lake Formation 可以让你对每个 Spark Job 强制执行一层权限，以便在 EKS 上的亚马逊 EMR 执行任务时应用 Lake Formation 权限控制。EKS 上的 Amazon EMR 使用 Spark 资源配置文件创建两个配置文件以有效执行任务。用户配置文件执行用户提供的代码，而系统配置文件则强制执行 Lake Formation 策略。每个启用 Lake Formation 的 Job 都使用两个 Spark 驱动程序，一个用于用户配置文件，另一个用于系统配置文件。有关更多信息，请参阅什么是 Amazon Lake Formation。

以下简要概述了 EKS 上的 Amazon EMR 如何访问受 Lake Formation 安全策略保护的数据。



以下步骤描述了此过程：

1. 用户在 EKS 虚拟集群上向支持 Amazon Lake Formation 的 Amazon EMR 提交 Spark Job。
2. EKS 上的 Amazon EMR 服务会设置用户驱动程序并在用户配置文件中运行任务。用户驱动程序运行精简版的 Spark，它无法启动任务、请求执行者、访问 Amazon S3 或 Glue 数据目录。它只会制定一个 Job 计划。
3. EKS 上的 Amazon EMR 服务会设置第二个名为系统驱动程序的驱动程序，并在系统配置文件中运行该驱动程序（使用特权身份）。Amazon EKS 在两个驱动程序之间设置了一个加密的 TLS 通道以进行通信。用户驱动程序使用该通道将工作计划发送给系统驱动程序。系统驱动程序不运行用户提交的代码。它运行完整 Spark，并与 Amazon S3 和数据目录通信以进行数据访问。它请求执行者并将Job Plan编译成一系列执行阶段。
4. 然后，EKS 服务上的 Amazon EMR 在执行者身上运行这些阶段。任何阶段的用户代码都只能在用户配置文件执行器上运行。

5. 从受 Lake Formation 保护的数据目录表中读取数据的阶段或应用安全筛选器的阶段将委托给系统执行者。

在 EKS 上使用亚马逊 EMR 启用 Lake Formation

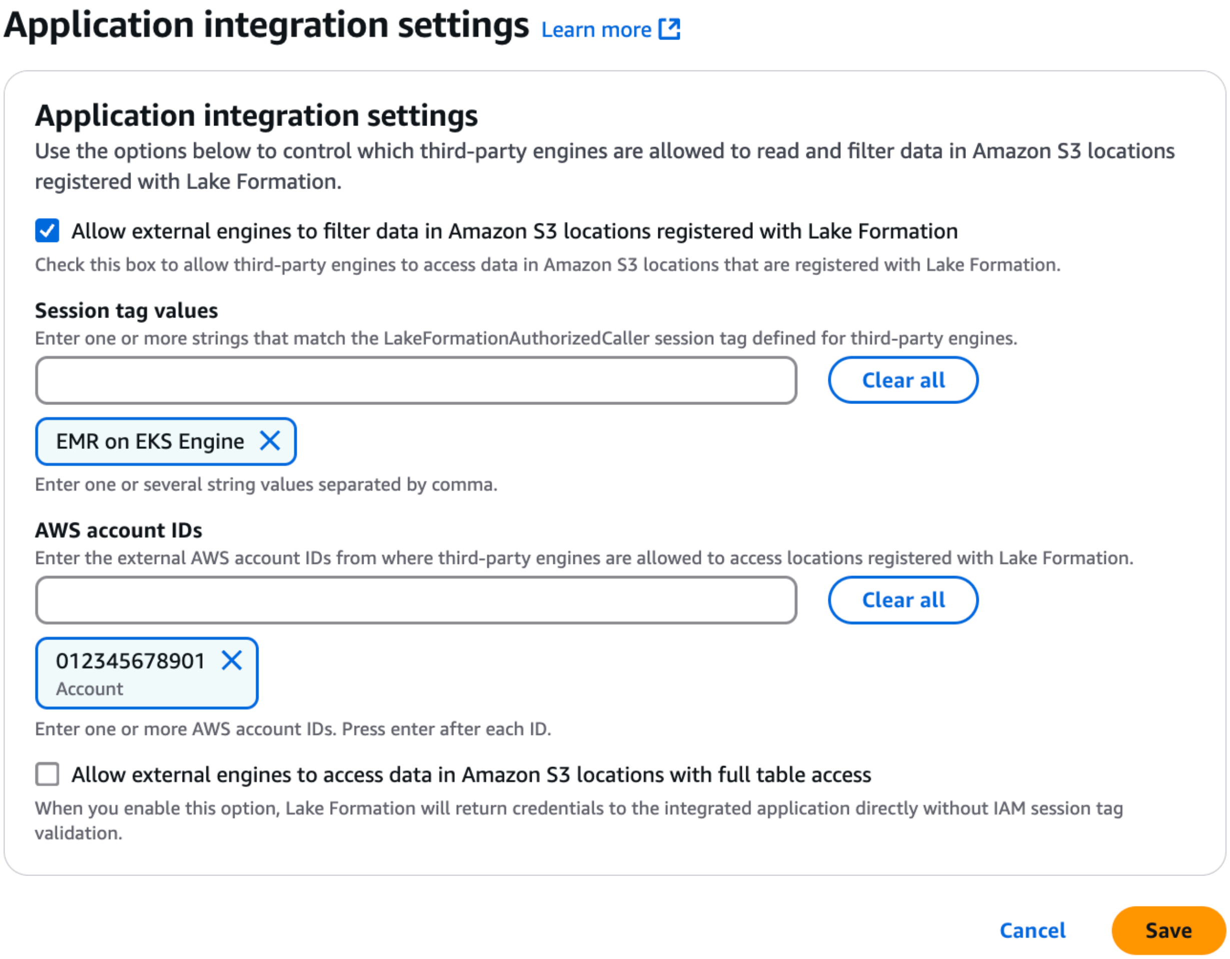
在 Amazon EMR 7.7 及更高版本中，您可以利用 Amazon Lake Formation 对由 Amazon S3 支持的数据目录表应用精细的访问控制。此功能允许您在 EKS Spark Jobs 上的 Amazon EMR 中为读取查询配置表、行、列和单元级别的访问控制。

本节介绍如何创建安全配置以及如何设置 Lake Formation 以与亚马逊 EMR 配合使用。它还描述了如何使用您为 Lake Formation 创建的安全配置创建虚拟集群。这些部分应按顺序完成。

步骤 1：设置基于 Lake Formation 的列、行或单元格级权限

首先，要对 Lake Formation 应用行和列级权限，Lake Formation 的数据湖管理员必须设置LakeFormationAuthorizedCaller会话标签。Lake Formation 使用该会话标签来授权调用者并提供对数据湖的访问权限。

导航到 Amazon Lake Formation 控制台，然后从侧栏的 “管理” 部分中选择 “应用程序集成设置” 选项。然后，选中 “允许外部引擎筛选在 Lake Formation 中注册的 Amazon S3 位置中的数据” 复选框。添加要运行 Spark 任务的Amazon 账户 IDs 和会话标签值。



请注意，此处传递的LakeFormationAuthorizedCaller会话标签SecurityConfiguration将在稍后设置 IAM 角色时在第 3 节中传递。

第 2 步：设置 EKS RBAC 权限

其次，您可以为基于角色的访问控制设置权限。

在 EKS 服务上向 Amazon EMR 提供 EKS 集群权限

EKS 服务上的 Amazon EMR 必须具有 EKS 集群角色权限，这样它才能为系统驱动程序创建跨命名空间权限，以分拆用户命名空间中的用户执行者。

创建集群角色

此示例定义了资源集合的权限。

```
vim emr-containers-cluster-role.yaml
---
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: emr-containers
rules:
  - apiGroups: [""]
    resources: ["namespaces"]
    verbs: ["get"]
  - apiGroups: [""]
    resources: ["serviceaccounts", "services", "configmaps", "events", "pods", "pods/
log"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
 "deletecollection", "annotate", "patch", "label"]
  - apiGroups: [""]
    resources: ["secrets"]
    verbs: ["create", "patch", "delete", "watch"]
  - apiGroups: ["apps"]
    resources: ["statefulsets", "deployments"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete", "annotate",
 "patch", "label"]
  - apiGroups: ["batch"]
    resources: ["jobs"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete", "annotate",
 "patch", "label"]
  - apiGroups: ["extensions", "networking.k8s.io"]
    resources: ["ingresses"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete", "annotate",
 "patch", "label"]
  - apiGroups: ["rbac.authorization.k8s.io"]
    resources: ["clusterroles","clusterrolebindings","roles", "rolebindings"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
 "deletecollection", "annotate", "patch", "label"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
 "deletecollection", "annotate", "patch", "label"]
---
```

```
kubectl apply -f emr-containers-cluster-role.yaml
```

创建集群角色绑定

```
vim emr-containers-cluster-role-binding.yaml
```

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: emr-containers
subjects:
- kind: User
  name: emr-containers
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: emr-containers
  apiGroup: rbac.authorization.k8s.io
---
```

```
kubectl apply -f emr-containers-cluster-role-binding.yaml
```

提供对 EKS 上的 Amazon EMR 服务的命名空间访问权限

创建两个 Kubernetes 命名空间，一个用于用户驱动程序和执行器，另一个用于系统驱动程序和执行器，并启用 EKS 上的 Amazon EMR 服务访问权限，以便在用户和系统命名空间中提交作业。按照现有指南为每个命名空间提供访问权限，该指南可在使用启用集群访问中找到`aws-auth`。

步骤 3：为用户和系统配置文件组件设置 IAM 角色

第三，您可以为特定组件设置角色。支持 Lake Formation 的 Spark Job 有两个组件，即用户和系统。用户驱动程序和执行器在 User 命名空间中运行，并与 StartJobRun API 中传递的 JobExecutionRole 命名空间相关联。系统驱动程序和执行器在 System 命名空间中运行，并与QueryEngine角色相关联。

配置查询引擎角色

该 QueryEngine 角色与系统空间组件相关联，并且有权担任 with Sess JobExecutionRolei LakeFormationAuthorizedCalleron 标签。查询引擎角色的 IAM 权限策略如下：

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

```
                "sts:TagSession"
            ],
            "Resource": "arn:aws:iam::Account:role/JobExecutionRole",
            "Condition": {
                "StringLike": {
                    "aws:RequestTag/LakeFormationAuthorizedCaller": "EMR on EKS Engine"
                }
            }
        },
        {
            "Sid": "AssumeJobRoleWithSessionTagAccessForSystemExecutor",
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Resource": "arn:aws:iam::Account:role/JobExecutionRole",
        },
        {
            "Sid": "CreateCertificateAccessForTLS",
            "Effect": "Allow",
            "Action": "emr-containers:CreateCertificate",
            "Resource": "*"
        }
    ]
}
```

将查询引擎角色的信任策略配置为信任 Kubernetes 系统命名空间。

```
aws emr-containers update-role-trust-policy \
    --cluster-name eks cluster \
    --namespace eks system namespace \
    --role-name query_engine_iam_role_name
```

有关更多信息，请参阅更新角色信任策略。

配置 Job 执行角色

Lake For Amazon 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 Amazon Glue APIs 以及资源的访问。尽管您可能拥有 Lake Formation 访问数据目录（SELECT）中表的权限，但如果您没有 `glue:Get*` API 操作的 IAM 权限，则操作将失败。

IAM 权限策略JobExecutionRole为：JobExecution角色的权限策略中应包含策略声明。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "GlueCatalogAccess",
            "Effect": "Allow",
            "Action": [
                "glue:Get*",
                "glue:Create*",
                "glue:Update*"
            ],
            "Resource": ["*"]
        },
        {
            "Sid": "LakeFormationAccess",
            "Effect": "Allow",
            "Action": [
                "lakeformation:GetDataAccess"
            ],
            "Resource": ["*"]
        },
        {
            "Sid": "CreateCertificateAccessForTLS",
            "Effect": "Allow",
            "Action": "emr-containers:CreateCertificate",
            "Resource": "*"
        }
    ]
}
```

适用于以下内容的 IAM 信任政策 JobExecutionRole：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "TrustQueryEngineRoleForSystemDriver",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::your_account:role/QueryExecutionRole"
            },
```

```
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ],
            "Condition": {
                "StringLike": {
                    "aws:RequestTag/LakeFormationAuthorizedCaller": "EMR on EKS Engine"
                }
            }
        },
        {
            "Sid": "TrustQueryEngineRoleForSystemExecutor",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::your_account:role/QueryEngineRole"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

将 Job 执行角色的信任策略配置为信任 Kubernetes 用户命名空间：

```
aws emr-containers update-role-trust-policy \
    --cluster-name eks cluster \
    --namespace eks User namespace \
    --role-name job_execution_role_name
```

有关更多信息，请参阅更新任务执行角色的信任策略。

步骤 4：设置安全配置

要运行启用 Lake Formation 的作业，您必须创建安全配置。

```
aws emr-containers create-security-configuration \
    --name 'security-configuration-name' \
    --security-configuration '{
        "authorizationConfiguration": {
            "lakeFormationConfiguration": {
                "authorizedSessionTagValue": "SessionTag configured in LakeFormation",
                "secureNamespaceInfo": {
                    "clusterId": "eks-cluster-name",
                    "namespace": "system-namespace-name"
```

```
                },
                "queryEngineRoleArn": "query-engine-IAM-role-ARN"
            }
        }
    }'
```

确保在 “authorizedSessionTag值” 字段中传递的会话标签可以授权 Lake Formation。将该值设置为在 Lake Formation 中配置的值步骤 1：设置基于 Lake Formation 的列、行或单元格级权限。

步骤 5：创建虚拟集群

使用安全配置在 EKS 虚拟集群上创建 Amazon EMR。

```
aws emr-containers create-virtual-cluster \
--name my-lf-enabled-vc \
--container-provider '{
    "id": "eks-cluster",
    "type": "EKS",
    "info": {
        "eksInfo": {
            "namespace": "user-namespace"
        }
    }
}' \
--security-configuration-id SecurityConfiguraionId
```

确保通过上一步中的 SecurityConfigurationID，以便将 Lake Formation 授权配置应用于虚拟集群上运行的所有作业。有关更多信息，请参阅使用亚马逊 EMR 注册 Amazon EKS 集群。

步骤 6：在 FGAC 中提交 Job 已启用 VirtualCluster

非 Lake Formation 和 Lake Formation 职位的提交流程相同。有关更多信息，请参阅提交使用运行的作业`StartJobRun`。

系统驱动程序的 Spark 驱动程序、执行器和事件日志存储在 Amazon 服务帐户的 S3 存储桶中以进行调试。我们建议在 Job Run 中配置客户管理的 KMS 密钥，以加密存储在 Amazon 服务存储桶中的所有日志。有关启用日志加密的更多信息，请参阅在 EKS 日志上加密 Amazon EMR。

注意事项和限制

在 EKS 上使用 Lake Formation 和 Amazon EMR 时，请注意以下注意事项和限制：

- EKS 上的 Amazon EMR 仅支持通过 Lake Formation 对 Apache Hive、Apache Iceberg、Apache Hudi 和 Delta 表格式进行精细访问控制。Apache Hive 格式包括 Parquet、ORC 和 xSV。
- `DynamicResourceAllocation`默认情况下处于启用状态，您无法关闭 Lake For `DynamicResourceAllocation` mation 作业。由于 DRA `spark.dynamicAllocation.maxExecutors` 配置的默认值为无穷大，因此请根据您的工作负载配置适当的值。
- 启用 Lake Formation 的作业不支持在系统驱动程序和系统执行器中的 EKS 映像上使用自定义 EMR。
- 您只能将 Lake Formation 与 Spark 作业结合使用。
- 带有 Lake Formation 的 EKS 上的 EMR 在整个作业中仅支持单个 Spark 会话。
- 带有 Lake Formation 的 EKS 上的 EMR 仅支持通过资源链接共享的跨账户表格查询。
- 不支持以下项：
  - 弹性分布式数据集（RDD）
  - Spark 流
  - 使用 Lake Formation 授予的权限写入
  - 嵌套列的访问控制
- EKS 上的 EMR 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、连接器或元数据`ANALYZE TABLE`仓命令提供额外的 jar
- 为了强制执行访问控制，`EXPLAIN PLAN` 和 DDL 操作（如 `DESCRIBE TABLE`）不会公开受限信息。
- EKS 上的 Amazon EMR 限制在支持 Lake Formation 的作业上访问系统驱动程序 Spark 日志。由于系统驱动程序的运行权限更大，因此系统驱动程序生成的事件和日志可能包含敏感信息。为了防止未经授权的用户或代码访问这些敏感数据，EKS 上的 EMR 禁止访问系统驱动程序日志。要进行故障排除，请联系 Amazon 支持人员。
- 如果您在 Lake Formation 中注册了表位置，则不管 EKS 上的 EMR 任务执行角色的 IAM 权限如何，数据访问路径都将通过 Lake Formation 存储的证书。如果您错误配置了在表位置中注册的角色，则提交的、使用该角色且对该表位置拥有 S3 IAM 权限的任务将失败。
- 写入 Lake Formation 表使用的是 IAM 权限，而不是 Lake Formation 授予的权限。如果您的任务执行角色具有必要的 S3 权限，则可以使用它来运行写入操作。

以下是使用 Apache Iceberg 时的注意事项和限制：

- 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。

有关更多信息，请参阅了解 EKS 概念和术语上的 Amazon EMR 和在 EKS 上启用 Amazon EMR 的集群访问权限。

数据管理员免责声明

> **Note**
>
> 当您向 EKS 上的 EMR 的 IAM 角色授予对 Lake Formation 资源的访问权限时，必须确保 EMR 集群管理员或操作员是可信管理员。这对于在多个组织和 Amazon 账户之间共享的 Lake Formation 资源尤其重要。

EKS 管理员的责任

- `System`命名空间应该受到保护。不允许任何用户、资源、实体或工具对命名空间中的 Kubernetes 资源拥有任何 Kubernetes RBAC 权限。`System`
- 除了 EKS 服务上的 EMR 之外，任何用户、资源或实体都不应有权访问命名空间中的 POD、CONFIG_MAP 和 SECRET。`CREATE User`
- `System`驱动程序和`System`执行程序包含敏感数据。因此，不应将`System`命名空间中的 Spark 事件、Spark 驱动程序日志和 Spark 执行器日志转发到外部日志存储系统。

故障排除

日志记录

EKS 上的 EMR 使用 Spark 资源配置文件来拆分任务执行。EKS 上的 Amazon EMR 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。您可以通过配置 StartJobRun 请求来访问作为用户配置文件运行的容器的日志[MonitoringConfiguration](MonitoringConfiguration)。

Spark 历史记录服务器

Spark History Server 包含根据用户配置文件生成的所有 Spark 事件和由系统驱动程序生成的经过编辑的事件。您可以在 “E xecutors” 选项卡中查看用户和系统驱动程序中的所有容器。但日志链接仅适用于用户配置文件。

由于 Lake Formation 权限不足，作业失败

确保您的任务执行角色具有运行权限`SELECT`以及您正在访问`DESCRIBE`的表的权限。

RDD 作业执行失败

EKS 上的 EMR 目前不支持对启用 Lake Formation 的作业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

EKS 上的 EMR 检测到安全验证错误。请联系 Amazon 支持人员寻求帮助。

跨账户 Amazon 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 Amazon Lake Formation 跨账户共享 Amazon Glue 数据目录和表格？。

Iceberg Job 在未设置区域时抛出初始化错误 Amazon

消息如下：

```
25/02/25 13:33:19 ERROR SparkFGACExceptionSanitizer: Client received error with id =
 b921f9e6-f655-491f-b8bd-b2842cdc20c7,
```

```
reason = IllegalArgumentException, message = Cannot initialize
LakeFormationAwsClientFactory, please set client.region to a valid aws region
```

确保 Spark 配置设置`spark.sql.catalog.catalog_name.client.region`为有效的区域。

投掷 Iceberg Job SparkUnsupportedOperationException

消息如下：

```
25/02/25 13:53:15 ERROR SparkFGACExceptionSanitizer: Client received error with id =
 921fef42-0800-448b-bef5-d283d1278ce0,
reason = SparkUnsupportedOperationException, message = Either glue.id or glue.account-
id is set with non-default account.
Cross account access with fine-grained access control is only supported with AWS
 Resource Manager.
```

确保将 Spark 配置设置`spark.sql.catalog.catalog_name.glue.account-id`为有效的账户 ID。

日志记录和监控

要检测事件、在发生事件时接收警报并进行响应，请针对 Amazon EMR on EKS 使用以下选项：

- 在 EKS 上监控 Amazon EMR Amazon CloudTrail — Amazon CloudTrail提供用户、角色或 Amazon 服务在 EKS 上的 Amazon EMR 中采取的操作记录。它捕获来自 Amazon EMR 控制台的调用以及以事件方式对 Amazon EMR on EKS API 操作的代码调用。您可确定向 Amazon EMR on EKS 发出的请求、发出请求的 IP 地址、何人发出的请求、请求的发出时间以及其它详细信息。有关更多信息，请参阅 使用在 EKS API 调用上记录 Amazon EMR Amazon CloudTrail。
- 在 EKS 上将 CloudWatch 事件与 Amazon EMR 配合使用 — CloudWatch Events 提供描述资源变化的近乎实时的系统事件流。 Amazon CloudWatch 通过发送消息以响应环境、激活函数、进行更改和捕获状态信息，事件会在操作变化发生时意识到操作变化，并对其做出响应，并在必要时采取纠正措施。要在 EKS 上使用带有 Amazon EMR CloudWatch 的事件，请创建一个规则，通过调用 EKS API 在 Amazon EMR 上触发。 CloudTrail有关更多信息，请参阅 通过 Amazon Events CloudWatch 监控作业。

使用托管存储对 EKS 日志上的 Amazon EMR 进行加密

以下各节介绍如何为日志配置加密。

启用加密

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以下配置。

```
"monitoringConfiguration": {
            "managedLogs": {
                "allowAWSToRetainLogs":"ENABLED",
                "encryptionKeyArn":"KMS key arn"
            },
            "persistentAppUI": "ENABLED"
        }
```

该`allowAWSToRetainLogs`配置 Amazon 允许在使用 Native FGAC 运行作业时保留系统命名空间日志。该`persistentAppUI`配置 Amazon 允许保存用于生成 Spark UI 的事件日志。`encryptionKeyArn`用于指定您要使用的 KMS 密钥 ARN 来加密存储的日志。 Amazon

日志加密所需的权限

必须允许提交作业或查看 Spark UI 的用户使用加密密钥执行操作`kms:DescribeKey``kms:GenerateDataKey`、和`kms:Decrypt`。这些权限用于验证密钥的有效性，并检查用户是否具有读取和写入使用 KMS 密钥加密的日志的必要权限。如果提交任务的用户缺少必要的密钥权限，则 EKS 上的 Amazon EMR 会拒绝提交任务运行。

用于调用的角色的 IAM 策略示例 StartJobRun

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": "emr-containers:StartJobRun",
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "kms:DescribeKey",
                "kms:Decrypt",
                "kms:GenerateDataKey"
            ],
            "Resource": "KMS key ARN",
            "Effect": "Allow"
```

```
        }
    ]
}
```

您还必须配置 KMS 密钥以允许persistentappui.elasticmapreduce.amazonaws.com和elasticmapreduce.amazonaws.com服务主体使用kms:GenerateDataKey和kms:Decrypt。这允许 EMR 读取使用 KMS 密钥加密的日志并将其写入托管存储。

KMS 密钥策略示例

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "IAM role ARN used to call StartJobRun"
            },
            "Action": "kms:DescribeKey",
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "kms:viaService": "emr-containers.region.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "IAM role ARN used to call StartJobRun"
            },
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "kms:viaService": "emr-containers.region.amazonaws.com",
                    "kms:EncryptionContext:aws:emr-containers:virtualClusterId":
 "virtual cluster id"
                }
```

```
            }
        },
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "persistentappui.elasticmapreduce.amazonaws.com",
                    "elasticmapreduce.amazonaws.com"
                ]
            },
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "kms:EncryptionContext:aws:emr-containers:virtualClusterId":
 "virtual cluster id",
                    "aws:SourceArn": "virtual cluster ARN"
                }
            }
        }
    ]
}
```

作为安全最佳实践，我们建议您添加`kms:viaServicekms:EncryptionContext`、和`aws:SourceArn`条件。这些条件有助于确保密钥仅由 EKS 上的 Amazon EMR 使用，并且仅用于在特定虚拟集群中运行的任务生成的日志。

使用在 EKS API 调用上记录 Amazon EMR Amazon CloudTrail

EKS 上的 Amazon EMR 与 Amazon CloudTrail一项服务集成，该服务提供用户、角色或 Amazon 服务在 EKS 上的 Amazon EMR 中采取的操作的记录。 CloudTrail 将 EKS 上的 Amazon EMR 的所有 API 调用捕获为事件。这些捕获包括来自 Amazon EMR on EKS 控制台的调用和对 Amazon EKS API 操作的代码调用。如果您创建了跟踪，则可以允许将 CloudTrail 事件持续传输到 Amazon S3 存储桶，包括 EKS 上的 Amazon EMR 事件。如果您未配置跟踪，您仍然可以在 CloudTrail 控制台的 "事件历史记录" 中查看最新的事件。使用收集的信息 CloudTrail，您可以确定在 EKS 上向 Amazon EMR 发出的请求、发出请求的 IP 地址、谁提出了请求、何时提出请求以及其他详细信息。

要了解更多信息 CloudTrail，请参阅[Amazon CloudTrail 用户指南](Amazon CloudTrail 用户指南)。

EKS 上的 Amazon EMR 信息位于 CloudTrail

CloudTrail 在您创建 Amazon 账户时已在您的账户上启用。当 EKS 上的 Amazon EMR 中发生活动时，该活动会与其他 Amazon 服务 CloudTrail 事件一起记录在事件历史记录中。您可以在自己的 Amazon 账户中查看、搜索和下载最近发生的事件。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 Amazon 账户中的事件，包括 EKS 上的 Amazon EMR 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，当您在控制台中创建跟踪时，该跟踪将应用于所有 Amazon 区域。跟踪记录 Amazon 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，您可以配置其他 Amazon 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，请参阅下列内容：

- 创建跟踪记录概述
- CloudTrail 支持的服务和集成
- 配置 Amazon SNS 通知 CloudTrail
- 接收来自多个地区的 CloudTrail 日志文件和接收来自多个账户的 CloudTrail 日志文件

EKS 上的所有亚马逊 EMR 操作均由亚马逊 EMR 在 EKS API 文档中记录 CloudTrail 并记录在亚马逊 EMR 中。例如，调用`StartJobRun`和`ListJobRuns`操作会在 CloudTrail 日志文件中生成条目。`CreateVirtualCluster`

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 Amazon 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 Amazon 服务发出。

有关更多信息，请参阅CloudTrail 用户身份元素。

了解 Amazon EMR on EKS 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[ListJobRuns]操作的 CloudTrail 日志条目。


```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE:admin",
    "arn": "arn:aws:sts::012345678910:assumed-role/Admin/admin",
    "accountId": "012345678910",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::012345678910:role/Admin",
        "accountId": "012345678910",
        "userName": "Admin"
      },
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2020-11-04T21:49:36Z"
      }
    }
  },
  "eventTime": "2020-11-04T21:52:58Z",
  "eventSource": "emr-containers.amazonaws.com",
  "eventName": "ListJobRuns",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "203.0.113.1",
  "userAgent": "aws-cli/1.11.167 Python/2.7.10 Darwin/16.7.0 botocore/1.7.25",
  "requestParameters": {
    "virtualClusterId": "1K48XXXXXXHCB"
  },
  "responseElements": null,
  "requestID": "890b8639-e51f-11e7-b038-EXAMPLE",
  "eventID": "874f89fa-70fc-4798-bc00-EXAMPLE",
  "readOnly": true,
  "eventType": "AwsApiCall",
  "recipientAccountId": "012345678910"
}
```

将 Amazon S3 Access Grants 与 Amazon EMR on EKS 结合使用

Amazon EMR on EKS 的 S3 Access Grants 概述

在 Amazon EMR 6.15.0 及更高版本中，Amazon S3 Access Grants 提供了一种可扩展的访问控制解决方案，您可以使用该解决方案来增强对 Amazon EMR on EKS 中的 Amazon S3 数据的访问权限。如果您的 S3 数据有复杂或大规模的权限配置，则可以使用 Access Grants 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 数据的访问权限，其超出运行时系统角色或 IAM 角色授予的权限，这些权限附加到具有对 EKS 上的 Amazon EMR 集群的访问权限的身份。

有关更多信息，请参阅《Amazon EMR 管理指南》中的使用 Amazon EMR 的 S3 Access Grants 管理访问权限和《Amazon Simple Storage Service 用户指南》中的使用 S3 Access Grants 管理访问权限。

本页介绍使用 S3 Access Grants 集成在 Amazon EMR on EKS 中运行 Spark 作业的要求。使用 Amazon EMR on EKS 时，S3 Access Grants 在作业的执行角色中需要额外的 IAM policy 声明，并且需要 `StartJobRun` API 的额外覆盖配置。有关使用其他 Amazon EMR 部署设置 S3 Access Grants 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 EMR Serverless 结合使用

使用 S3 Access Grants 启动 EKS 上的 Amazon EMR 集群以进行数据管理

您可以在 Amazon EMR on EKS 上启用 S3 Access Grants 并启动 Spark 作业。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 EKS 上的 Amazon EMR 集群设置作业执行角色。包括运行 Spark 作业所需的 IAM 权限，`s3:GetDataAccess` 和 `s3:GetAccessGrantsInstanceForPrefix`：

    ```
    {
        "Effect": "Allow",
        "Action": [
        "s3:GetDataAccess",
        "s3:GetAccessGrantsInstanceForPrefix"
        ],
    ```

```
    "Resource": [     //LIST ALL INSTANCE ARNS THAT THE ROLE IS ALLOWED TO QUERY
         "arn:aws_partition:s3:Region:account-id1:access-grants/default",
         "arn:aws_partition:s3:Region:account-id2:access-grants/default"
    ]
}
```

> **Note**
>
> 如果您为作业执行指定的 IAM 角色具有直接访问 S3 的任何额外权限，则无论您在 S3 Access Grants 中定义了什么权限，用户都可能能够访问数据

2. 使用 Amazon EMR 发布版标签 6.15 或更高版本以及 `emrfs-site` 分类向 EKS 上的 Amazon EMR 集群提交作业，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。

```
{
  "name": "myjob",
  "virtualClusterId": "123456",
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "emr-7.8.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "entryPoint_location",
      "entryPointArguments": ["argument1", "argument2"],
       "sparkSubmitParameters": "--class main_class"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "emrfs-site",
        "properties": {
          "fs.s3.s3AccessGrants.enabled": "true",
          "fs.s3.s3AccessGrants.fallbackToIAM": "false"
         }
      }
    ],
  }
}
```

Amazon EMR on EKS 的 S3 Access Grants 注意事项

关于将 Amazon S3 Access Grants 与 Amazon EMR on EKS 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。

Amazon EMR on EKS 的合规性验证

作为多项合规计划的一部分，第三方审计师评估 EKS 上的 Amazon EMR 的安全性与 Amazon 合规性。其中包括 SOC、PCI、FedRAMP、HIPAA 及其他。

Amazon EMR on EKS 的恢复能力

Amazon 全球基础设施是围绕 Amazon 区域和可用区构建的。 Amazon 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，您可以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结构相比，可用区具有更高的可用性、容错性和可扩展性。

有关 Amazon 区域和可用区的更多信息，请参阅Amazon 全球基础设施。

除了 Amazon 全球基础设施外，EKS 上的 Amazon EMR 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 on EKS 中的基础设施安全性

作为一项托管服务，Amazon EMR 受到 Amazon 全球网络安全的保护。有关 Amazon 安全服务以及如何 Amazon 保护基础设施的信息，请参阅Amazon 云安全。要使用基础设施安全的最佳实践来设计您的 Amazon 环境，请参阅 S Amazon ecurity Pillar Well-Architected Fram ework 中的基础设施保护。

您可以使用 Amazon 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。

此外，必须使用访问密钥 ID 和与 IAM 主体关联的秘密访问密钥来对请求进行签名。或者，您可以使用 Amazon Security Token Service（Amazon STS）生成临时安全凭证来对请求进行签名。

配置和漏洞分析

Amazon 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 on EKS 的合规性验证
- 责任共担模式
- Amazon Web Services：安全过程概述（白皮书）

使用接口 VPC 端点连接到 Amazon EMR on EKS

您可以使用虚拟私有云 (VPC Amazon PrivateLink) 中的接口 VPC 终端节点 () 直接连接到 EKS 上的 Amazon EMR，而不必通过互联网进行连接。当您使用接口 VPC 终端节点时，您的 VPC 与 EKS 上的 Amazon EMR 之间的通信完全在网络内进行。 Amazon 每个 VPC 终端节点都由一个或多个弹性网络接口 (ENIs) 表示，其私有 IP 地址位于您的 VPC 子网中。

接口 VPC 终端节点将您的 VPC 直接连接到 EKS 上的 Amazon EMR，无需互联网网关、NAT 设备、VPN 连接或 Direc Amazon t Connect 连接。VPC 中的实例不需要公有 IP 地址便可与 Amazon EMR on EKS API 进行通信。

您可以使用 Amazon Web Services Management Console 或 Amazon Command Line Interface (Amazon CLI) 命令创建接口 VPC 终端节点以连接到 EKS 上的 Amazon EMR。有关更多信息，请参阅创建接口端点。

在创建接口 VPC 终端节点后，如果您为终端节点启用私有 DNS 主机名，则默认 Amazon EMR on EKS 终端节点将解析为您的 VPC 终端节点。Amazon EMR on EKS 的默认服务名称终端节点采用以下格式。

```
emr-containers.Region.amazonaws.com
```

如果您不启用私有 DNS 主机名，Amazon VPC 将提供一个您可以使用的 DNS 终端节点名称，格式如下。

```
VPC_Endpoint_ID.emr-containers.Region.vpce.amazonaws.com
```

有关更多信息，请参阅 Amazon VPC 用户指南中的接口 VPC 终端节点 (Amazon PrivateLink)。Amazon EMR on EKS 支持调用您 VPC 中的所有 API 操作。

您可以将 VPC 终端节点策略附加到 VPC 终端节点，以控制 IAM 委托人的访问权限。您还可以将安全组与 VPC 端点关联，以便根据网络流量的源和目标（例如 IP 地址范围）控制入站和出站访问。有关更多信息，请参阅使用 VPC 端点控制对服务的访问。

为 Amazon EMR on EKS 创建 VPC 终端节点策略

您可以为 Amazon EMR on EKS 的 Amazon VPC 终端节点创建一个策略，在该策略中指定以下内容：

- 可以或不能执行操作的委托人
- 可执行的操作
- 可对其执行操作的资源

有关更多信息，请参阅《Amazon VPC 用户指南》中的使用 VPC 终端节点控制对服务的访问权限。

Example 拒绝来自指定 Amazon 账户的所有访问的 VPC 终端节点策略

以下 VPC 终端节点策略拒绝 Amazon 账户使用该终端节点访问*123456789012*所有资源。

```
{
    "Statement": [
        {
            "Action": "*",
            "Effect": "Allow",
            "Resource": "*",
            "Principal": "*"
        },
        {
            "Action": "*",
            "Effect": "Deny",
            "Resource": "*",
            "Principal": {
                "AWS": [
                    "123456789012"
                ]
            }
        }
    ]
}
```

Example 仅允许 VPC 访问指定的 IAM 委托人（用户）的 VPC 终端节点策略

以下 VPC 终端节点策略仅允许 Amazon 账户*lijuan*中的 IAM 用户拥有完全访问权限*123456789012*。使用终端节点拒绝所有其它 IAM 委托人进行访问。

```
{
    "Statement": [
        {
            "Action": "*",
            "Effect": "Allow",
            "Resource": "*",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::123456789012:user/lijuan"
                ]
            }
        }
    ]
}
```

Example 允许只读 Amazon EMR on EKS 操作的 VPC 终端节点策略

以下 VPC 终端节点策略仅*123456789012*允许 Amazon 账户在 EKS 上执行指定的 Amazon EMR 操作。

指定的操作为 Amazon EMR on EKS 提供等效的只读访问权限。针对指定账户拒绝 VPC 上的所有其它操作。所有其它帐户都被拒绝进行任何访问。有关 Amazon EMR on EKS 操作列表，请参阅 Amazon EMR on EKS 的操作、资源和条件键。

```
{
    "Statement": [
        {
            "Action": [
                "emr-containers:DescribeJobRun",
                "emr-containers:DescribeVirtualCluster",
                "emr-containers:ListJobRuns",
                "emr-containers:ListTagsForResource",
                "emr-containers:ListVirtualClusters"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Principal": {
                "AWS": [
```

```
                    "123456789012"
                ]
            }
        }
    ]
}
```

Example 拒绝访问指定虚拟集群的 VPC 终端节点策略

以下 VPC 终端节点策略允许所有账户和委托人拥有完全访问权限，但拒绝 Amazon 账户*123456789012*访问在具有集群 ID *A1B2CD34EF5G* 的虚拟集群上执行的操作。对于仍然不被允许的虚拟集群，其它 Amazon EMR on EKS 操作不支持资源级权限。有关 Amazon EMR on EKS 操作及其相应资源类型的列表，请参阅《Amazon Identity and Access Management 用户指南》中的 [Amazon EMR on EKS 的操作、资源和条件键]。

```
{
    "Statement": [
        {
            "Action": "*",
            "Effect": "Allow",
            "Resource": "*",
            "Principal": "*"
        },
        {
            "Action": "*",
            "Effect": "Deny",
            "Resource": "arn:aws:emr-containers:us-west-2:123456789012:/
virtualclusters/A1B2CD34EF5G",
            "Principal": {
                "AWS": [
                    "123456789012"
                ]
            }
        }
    ]
}
```

为 Amazon EMR on EKS 设置跨账户访问权限

您可为 Amazon EMR on EKS 设置跨账户访问权限。跨账户访问允许一个 Amazon 账户的用户在 EKS 任务上运行 Amazon EMR，并访问属于 Amazon 另一个账户的基础数据。

先决条件

要在 EKS 上为 Amazon EMR 设置跨账户访问权限，您需要在登录以下账户的 Amazon 同时完成任务：

- AccountA-一个 Amazon 账户，您通过在 EKS 集群上使用命名空间注册 Amazon EMR，在 EKS 虚拟集群上创建了 Amazon EMR。
- AccountB-一个包含亚马逊 S3 存储桶或 DynamoDB 表的 Amazon 账户，您想让 EKS 上的 Amazon EMR 任务访问这些存储桶或 DynamoDB 表。

在设置跨账户访问权限之前，您必须在 Amazon 账户中准备好以下内容：

- AccountA 中的 Amazon EMR on EKS 虚拟集群，即要运行任务之处。
- AccountA 中的任务执行角色拥有在虚拟集群中运行任务所需的权限。有关更多信息，请参阅创建任务执行角色和将任务执行角色与 Amazon EMR on EKS 结合使用。

如何访问跨账户 Amazon S3 存储桶或 DynamoDB 表

要为 Amazon EMR on EKS 设置跨账户访问权限，请完成以下步骤。

1. 创建 Amazon S3 存储桶，即 AccountB 中的 cross-account-bucket。有关更多信息，请参阅创建存储桶。如果您希望跨账户访问 DynamoDB，还可以在 AccountB 中创建 DynamoDB 表。有关更多信息，请参阅创建 DynamoDB 表。
2. 在 AccountB 中创建 Cross-Account-Role-B IAM 角色，以便可以访问 cross-account-bucket。

    1. 登录到 IAM 控制台。
    2. 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。
    3. 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》中的创建新策略。

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

```
                "Effect": "Allow",
                "Action": "s3:*",
                "Resource": [
                    "arn:aws:s3:::cross-account-bucket",
                    "arn:aws:s3:::cross-account-bucket/*"
                ]
            }
        ]
}
```

如果需要 DynamoDB 访问权限，请创建 IAM policy，该策略指定访问跨账户 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》中的创建 DynamoDB 表。

以下是访问 DynamoDB 表 `CrossAccountTable` 的策略。

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

3. 编辑 `Cross-Account-Role-B` 角色的信任关系。

   1. 要为角色配置信任关系，请选择 IAM 控制台的 Trust Relationships (信任关系) 选项卡，以用于在步骤 2 中创建的角色：`Cross-Account-Role-B`。
   2. 选择 Edit Trust Relationship (编辑信任关系)。
   3. 添加以下策略文档，该文档允许使用 `AccountA` 中的 `Job-Execution-Role-A` 来假承担该 `Cross-Account-Role-B` 角色。

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

```
        "Principal": {
          "AWS": "arn:aws:iam::AccountA:role/Job-Execution-Role-A"
        },
        "Action": "sts:AssumeRole"
      }
    ]
}
```

4. 授予 AccountA 中的 Job-Execution-Role-A STS 代入角色权限来代入 Cross-Account-Role-B。

    1. 在 Amazon 账户的 IAM 控制台中AccountA，选择Job-Execution-Role-A。
    2. 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。

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

5. 对于 Amazon S3 访问权限，请设置以下 spark-submit 参数（spark conf），同时将任务提交至 Amazon EMR on EKS。

> **Note**
>
> 默认情况下，EMRFS 使用任务执行角色以访问任务的 S3 存储桶。但当 customAWSCredentialsProvider 设置为 AssumeRoleAWSCredentialsProvider 时，EMRFS 将使用您通过 ASSUME_ROLE_CREDENTIALS_ROLE_ARN 指定的相应角色，而不是通过 Job-Execution-Role-A 以访问 Amazon S3。

- --conf spark.hadoop.fs.s3.customAWSCredentialsProvider=com.amazonaws.emr.AssumeRol

- --conf spark.kubernetes.driverEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN=arn:aws:iam::*Ac* Cross-Account-Role-B \
- --conf spark.executorEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN=arn:aws:iam::*AccountB*:ro Cross-Account-Role-B \

> **Note**
>
> 您必须在任务 Spark 配置中为执行程序和驱动程序 env 设置 ASSUME_ROLE_CREDENTIALS_ROLE_ARN。

对于 DynamoDB 跨账户访问，您必须设置 --conf spark.dynamodb.customAWSCredentialsProvider=com.amazonaws.emr.AssumeRoleAWSCr

6. 使用跨账户访问权限运行 Amazon EMR on EKS 上的任务，如以下示例所示。

```
aws emr-containers start-job-run \
--virtual-cluster-id 123456 \
--name myjob \
--execution-role-arn execution-role-arn \
--release-label emr-6.2.0-latest \
--job-driver '{"sparkSubmitJobDriver": {"entryPoint": "entryPoint_location",
 "entryPointArguments": ["arguments_list"], "sparkSubmitParameters": "--class
 <main_class> --conf spark.executor.instances=2 --conf spark.executor.memory=2G
 --conf spark.executor.cores=2 --conf spark.driver.cores=1 --conf
 spark.hadoop.fs.s3.customAWSCredentialsProvider=com.amazonaws.emr.AssumeRoleAWSCredentials
 --conf
 spark.kubernetes.driverEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN=arn:aws:iam::AccountB:role/
Cross-Account-Role-B --conf
 spark.executorEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN=arn:aws:iam::AccountB:role/
Cross-Account-Role-B"}} ' \
--configuration-overrides '{"applicationConfiguration": [{"classification":
 "spark-defaults", "properties": {"spark.driver.memory": "2G"}}],
 "monitoringConfiguration": {"cloudWatchMonitoringConfiguration":
 {"logGroupName": "log_group_name", "logStreamNamePrefix": "log_stream_prefix"},
 "persistentAppUI":"ENABLED",  "s3MonitoringConfiguration": {"logUri": "s3://
my_s3_log_location" }}}'
```

为您的 Amazon EMR on EKS 资源添加标签

为了帮助您管理 Amazon EMR on EKS 资源，您可使用标签为每个资源分配您自己的元数据。本主题概览了标签功能，并说明如何创建标签。

主题

- 标签基本知识
- 标记资源
- 标签限制
- 使用标签 Amazon CLI 和在 EKS API 上使用亚马逊 EMR

标签基本知识

标签是您分配给 Amazon 资源的标签。每个标签都包含定义的一个键和一个可选值。

标签使您能够按用途、所有者或环境等属性对 Amazon 资源进行分类。在具有相同类型的许多资源时，可以根据分配给资源的标签快速识别具体的资源。例如，您可以为 Amazon EMR on EKS 集群定义一组标签，以帮助您跟踪每个集群的拥有者和堆栈级别。我们建议为每个资源类型设计一组一致的标签键。然后，您可以根据添加的标签搜索和筛选资源。

标签不会自动分配至资源。添加标签后，可以编辑标签键和值，还可以随时删除资源的标签。如果删除资源，资源的所有标签也会被删除。

标签对 Amazon EMR on EKS 没有任何语义意义，应严格按字符串进行解析。

标签值可以是空字符串，但是不能是 null。标签的键不能是空字符串。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 Amazon Identity and Access Management (IAM)，则可以控制 Amazon 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。

标记资源

您可以标记新的或现有的虚拟集群，以及处于活动状态的任务运行。任务运行的活动状态包括：`PENDING`、`SUBMITTED`、`RUNNING` 和 `CANCEL_PENDING`。虚拟集群的活动状态包

括：RUNNING、TERMINATING 和 ARRESTED。有关更多信息，请参阅任务运行状态和虚拟集群状态。

当虚拟集群终止时，将清除标签并且不再可访问。

如果您使用的是 EKS 上的 Amazon EMR API Amazon CLI、或 Amazon 软件开发工具包，则可以使用相关 API 操作中的标签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。如果您在创建时标记资源，则无需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 on EKS 资源。

资源	支持标签	支持标签传播	支持在创建时添加标签（EKS API Amazon CLI和 SDK 上的 Amazon EMR） Amazon	创建 API（可以在创建过程中添加标签）
虚拟集群	是	否。与虚拟集群关联的标签不会传播到提交给该虚拟集群的任务运行。	是	CreateVirtualCluster
任务运行	是	否	是	StartJobRun

标签限制

下面是适用于标签的基本限制：

- 每个资源的标签数上限 – 50
- 对于每个资源，每个标签键都必须是唯一的，每个标签键只能有一个值。
- 最大键长度 – 128 个 Unicode 字符（采用 UTF-8 格式）
- 最大值长度 – 256 个 Unicode 字符（采用 UTF-8 格式）

- 如果您的标记架构用于多个 Amazon 服务和资源，请记住，其他服务可能对允许的字符有限制。通常允许使用的字符包括可用 UTF-8 格式表示的字母、数字和空格，以及以下字符：+ - = . _ : / @。
- 标签键和值区分大小写。
- 标签值可以是空字符串，但是不能是 null。标签的键不能是空字符串。
- 请不要使用 `aws:`、`AWS:` 或任何大写或小写组合（例如，键或值的前缀）。这些仅供 Amazon 使用。

使用标签 Amazon CLI 和在 EKS API 上使用亚马逊 EMR

使用以下 Amazon CLI 命令或 EKS API 操作上的 Amazon EMR 来添加、更新、列出和删除资源的标签。

Task	Amazon CLI	API 操作
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

以下示例说明如何使用 Amazon CLI给资源加标签或取消标签。

示例 1：标记现有虚拟集群

以下命令标记现有虚拟集群。

```
aws emr-containers tag-resource --resource-arn resource_ARN --tags team=devs
```

示例 2：取消标记现有虚拟集群

以下命令从现有虚拟集群删除标签。

```
aws emr-containers untag-resource --resource-arn resource_ARN --tag-keys tag_key
```

示例 3：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-containers list-tags-for-resource --resource-arn resource_ARN
```

对 Amazon EMR on EKS 进行故障排除

本部分介绍如何对 Amazon EMR on EKS 的问题进行故障排除。有关如何对 Amazon EMR 的一般问题进行故障排除的相关信息，请参阅《Amazon EMR 管理指南》中的排除集群故障。

主题

- 对使用 PersistentVolumeClaims (PVC) 的作业进行故障排除
- 对 Amazon EMR on EKS 垂直自动扩展进行问题排查
- 对 Amazon EMR on EKS Spark 运算符进行故障排除

对使用 PersistentVolumeClaims (PVC) 的作业进行故障排除

如果你需要为任务创建、列出或删除 PersistentVolumeClaims (PVC)，但不向默认 Kubernetes 角色 emr- containers 添加 PVC 权限，那么当你提交任务时，任务就会失败。如果没有这些权限，则 emr-containers 角色无法为 Spark 驱动程序或 Spark 客户端创建必要的角色。如错误消息所示，向 Spark 驱动程序或客户端角色添加权限是不够的。emr-containers 主角色还必须包括所需的权限。本节介绍如何向 emr-containers 主角色添加所需的权限。

验证

要验证您的 emr-containers 角色是否具有必要的权限，请使用您自己的值设置 NAMESPACE 变量，然后运行以下命令：

```
export NAMESPACE=YOUR_VALUE
kubectl describe role emr-containers -n ${NAMESPACE}
```

此外，若要验证 Spark 和客户端角色是否具有必要的权限，请运行以下命令：

```
kubectl describe role emr-containers-role-spark-driver -n ${NAMESPACE}
kubectl describe role emr-containers-role-spark-client -n ${NAMESPACE}
```

如果没有权限，请按如下方式继续修补。

修补

1. 如果没有权限的作业当前正在运行，则停止这些作业。
2. 创建名为 RBAC_Patch.py 的文件，如下所示：

```
import os
import subprocess as sp
import tempfile as temp
import json
import argparse
import uuid

def delete_if_exists(dictionary: dict, key: str):
    if dictionary.get(key, None) is not None:
        del dictionary[key]

def doTerminalCmd(cmd):
    with temp.TemporaryFile() as f:
        process = sp.Popen(cmd, stdout=f, stderr=f)
        process.wait()
        f.seek(0)
        msg = f.read().decode()
    return msg

def patchRole(roleName, namespace, extraRules, skipConfirmation=False):
    cmd = f"kubectl get role {roleName} -n {namespace} --output json".split(" ")
    msg = doTerminalCmd(cmd)
    if "(NotFound)" in msg and "Error" in msg:
        print(msg)
        return False
    role = json.loads(msg)
    rules = role["rules"]
    rulesToAssign = extraRules[::]
    passedRules = []
    for rule in rules:
        apiGroups = set(rule["apiGroups"])
        resources = set(rule["resources"])
        verbs = set(rule["verbs"])
        for extraRule in extraRules:
            passes = 0
            apiGroupsExtra = set(extraRule["apiGroups"])
            resourcesExtra = set(extraRule["resources"])
            verbsExtra = set(extraRule["verbs"])
            passes += len(apiGroupsExtra.intersection(apiGroups)) >=
 len(apiGroupsExtra)
            passes += len(resourcesExtra.intersection(resources)) >=
 len(resourcesExtra)
            passes += len(verbsExtra.intersection(verbs)) >= len(verbsExtra)
```

```
            if passes >= 3:
                if extraRule not in passedRules:
                    passedRules.append(extraRule)
                    if extraRule in rulesToAssign:
                        rulesToAssign.remove(extraRule)
                break
    prompt_text = "Apply Changes?"
    if len(rulesToAssign) == 0:
        print(f"The role {roleName} seems to already have the necessary
 permissions!")
        prompt_text = "Proceed anyways?"
    for ruleToAssign in rulesToAssign:
        role["rules"].append(ruleToAssign)
    delete_if_exists(role, "creationTimestamp")
    delete_if_exists(role, "resourceVersion")
    delete_if_exists(role, "uid")
    new_role = json.dumps(role, indent=3)
    uid = uuid.uuid4()
    filename = f"Role-{roleName}-New_Permissions-{uid}-TemporaryFile.json"
    try:
        with open(filename, "w+") as f:
            f.write(new_role)
            f.flush()
        prompt = "y"
        if not skipConfirmation:
            prompt = input(
                doTerminalCmd(f"kubectl diff -f {filename}".split(" ")) +
 f"\n{prompt_text} y/n: "
            ).lower().strip()
            while prompt != "y" and prompt != "n":
                prompt = input("Please make a valid selection. y/n:
 ").lower().strip()
        if prompt == "y":
            print(doTerminalCmd(f"kubectl apply -f {filename}".split(" ")))
    except Exception as e:
        print(e)
    os.remove(f"./{filename}")

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument("-n", "--namespace",
                        help="Namespace of the Role. By default its the
 VirtualCluster's namespace",
                        required=True,
```

```
                        dest="namespace"
                        )

    parser.add_argument("-p", "--no-prompt",
                        help="Applies the patches without asking first",
                        dest="no_prompt",
                        default=False,
                        action="store_true"
                        )
    args = parser.parse_args()

    emrRoleRules = [
        {
            "apiGroups": [""],
            "resources": ["persistentvolumeclaims"],
            "verbs": ["list", "create", "delete", "patch"]
         }

    ]

    driverRoleRules = [
        {
            "apiGroups": [""],
            "resources": ["persistentvolumeclaims"],
            "verbs": ["list", "create", "delete", "patch", "deletecollection"]
        },
        {
            "apiGroups": [""],
            "resources": ["services"],
            "verbs": ["get", "list", "describe", "create", "delete", "watch",
 "deletecollection"]
        },
        {
            "apiGroups": [""],
            "resources": ["configmaps", "pods"],
            "verbs": ["deletecollection"]
        }
    ]

    clientRoleRules = [
        {
            "apiGroups": [""],
            "resources": ["persistentvolumeclaims"],
            "verbs": ["list", "create", "delete", "patch"]
```

```
        }
    ]

    patchRole("emr-containers", args.namespace, emrRoleRules, args.no_prompt)
    patchRole("emr-containers-role-spark-driver", args.namespace, driverRoleRules,
 args.no_prompt)
    patchRole("emr-containers-role-spark-client", args.namespace, clientRoleRules,
 args.no_prompt)
```

3. 运行 Python 脚本：

```
python3 RBAC_Patch.py -n ${NAMESPACE}
```

4. 显示新权限和旧权限之间的 kubectl 差异。按 y 以修补角色。
5. 使用附加权限验证三个角色，如下所示：

```
kubectl describe role -n ${NAMESPACE}
```

6. 运行 Python 脚本：

```
python3 RBAC_Patch.py -n ${NAMESPACE}
```

7. 运行命令后，将显示新旧权限之间的 kubectl 差异。按 y 以修补角色。
8. 使用附加权限验证三个角色：

```
kubectl describe role -n ${NAMESPACE}
```

9. 再次提交作业。

手动修补

如果您的应用程序所需的权限适用于 PVC 规则以外的其他内容，您可以根据需要为 Amazon EMR 虚拟集群手动添加 Kubernetes 权限。

> **Note**
>
> 角色 emr-containers 是主要角色。这意味着它必须提供所有必要的权限，然后您才能更改底层驱动程序或客户端角色。

1. 通过运行以下命令将当前权限下载到 yaml 文件：

```
kubectl get role -n ${NAMESPACE} emr-containers -o yaml >> emr-containers-role-patch.yaml
kubectl get role -n ${NAMESPACE} emr-containers-role-spark-driver -o yaml >> driver-role-patch.yaml
kubectl get role -n ${NAMESPACE} emr-containers-role-spark-client -o yaml >> client-role-patch.yaml
```

2. 根据应用程序所需的权限，编辑每个文件并添加其他规则，如下所示：

   - emr-containers-role-patch.yaml

```
- apiGroups:
  - ""
  resources:
  - persistentvolumeclaims
  verbs:
  - list
  - create
  - delete
  - patch
```

   - driver-role-patch.yaml

```
- apiGroups:
  - ""
  resources:
  - persistentvolumeclaims
  verbs:
  - list
  - create
  - delete
  - patch
  - deletecollection
- apiGroups:
  - ""
  resources:
  - services
  verbs:
  - get
  - list
  - describe
  - create
```

```
    - delete
    - watch
    - deletecollection
  - apiGroups:
    - ""
    resources:
    - configmaps
    - pods
    verbs:
    - deletecollection
```

- client-role-patch.yaml

```
  - apiGroups:
    - ""
    resources:
    - persistentvolumeclaims
    verbs:
    - list
    - create
    - delete
    - patch
```

3. 删除以下属性及其值。这是应用更新所必需的。
   - creationTimestamp
   - resourceVersion
   - uid
4. 最后，运行补丁：

```
kubectl apply -f emr-containers-role-patch.yaml
kubectl apply -f driver-role-patch.yaml
kubectl apply -f client-role-patch.yaml
```

对 Amazon EMR on EKS 垂直自动扩展进行问题排查

如果您在使用 Operator Lifecycle Manager 在 Amazon EKS 集群上设置 Amazon EMR on EKS 垂直自动扩展运算符时遇到问题，请参阅以下部分。有关更多信息（包括完成安装的步骤），请参阅 使用垂直自动扩展功能处理 Amazon EMR Spark 任务。

403 禁止错误

如果您按照 `olm status` 中的步骤运行了 在 Amazon EKS 集群上安装 Operator Lifecycle Manager（OLM） 命令，并且返回了如下所示的 `403 Forbidden` 错误，则说明您可能尚未获取运算符的 Amazon ECR 存储库身份验证令牌。

要解决此问题，请重复 安装 Amazon EMR on EKS 垂直自动扩展 Operator 中的步骤以获取令牌。然后，再次尝试安装。

```
Error: FATA[0002] Failed to run bundle: pull bundle image: error pulling image IMAGE.
error resolving name : unexpected status code [manifests latest]: 403 Forbidden
```

未找到 Kubernetes 命名空间

在 Amazon EKS 集群上设置 Amazon EMR on EKS 垂直自动扩展运算符时，可能会出现 `namespaces not found` 错误，如下所示：

```
FATA[0020] Failed to run bundle: create catalog: error creating catalog source:
 namespaces "NAME" not found.
```

如果您指定的命名空间不存在，OLM 将不会安装垂直自动扩展运算符。要解决此问题，请使用以下命令创建命名空间。然后，再次尝试安装。

```
kubectl create namespace NAME
```

保存 Docker 凭证时出错

要设置垂直自动扩展，您必须进行身份验证并获取与 Amazon EMR on EKS 垂直自动扩展相关的 Docker 映像。执行此操作时，如果 Docker 没有运行，您可能会收到类似以下的错误：

```
aws ecr get-login-password \
 --region $REGION | docker login \
 --username AWS \
 --password-stdin $ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com

Error saving credentials: error storing credentials - err: exit status 1
out: 'Post "http://ipc/registry/credstore-updated": dial unix backend.sock: connect: no
 such file or directory'
```

要解决此问题，请确认 Docker 正在运行或打开 Docker 桌面。然后，尝试再次保存凭证。

对 Amazon EMR on EKS Spark 运算符进行故障排除

如果您在使用 Amazon EMR on EKS Spark 运算符时遇到问题，请参阅以下部分。有关更多信息（包括完成安装的步骤），请参阅 使用 Spark Operator 运行 Spark 任务。

Helm 图表安装出错

尝试安装或验证 Helm 图表时，如果您按照 安装 Spark Operator 中的步骤执行操作，返回了如下所示的 `INSTALLATION FAILED` 错误，则说明您可能尚未获取运算符的 Amazon ECR 存储库身份验证令牌。

要解决此问题，请重复 安装 Spark Operator 中的步骤，向 Amazon ECR 注册表验证 Helm 客户端。然后，再次尝试安装步骤。

```
Error: INSTALLATION FAILED: Kubernetes cluster unreachable: the server has asked for
 the client to provide credentials
```

UnsupportedFileSystemException: 方案 “s3” 不 FileSystem 是

您可能会在线程“main”中遇到以下异常情况：

```
org.apache.hadoop.fs.UnsupportedFileSystemException: No FileSystem for scheme "s3"
```

如果发生这种情况，请在 `SparkApplication` 规范中添加以下例外：

```
 hadoopConf:
   # EMRFS filesystem
   fs.s3.customAWSCredentialsProvider:
 com.amazonaws.auth.WebIdentityTokenCredentialsProvider
   fs.s3.impl: com.amazon.ws.emr.hadoop.fs.EmrFileSystem
   fs.AbstractFileSystem.s3.impl: org.apache.hadoop.fs.s3.EMRFSDelegate
   fs.s3.buffer.dir: /mnt/s3
   fs.s3.getObject.initialSocketTimeoutMilliseconds: "2000"
   mapreduce.fileoutputcommitter.algorithm.version.emr_internal_use_only.EmrFileSystem:
 "2"
   mapreduce.fileoutputcommitter.cleanup-
failures.ignored.emr_internal_use_only.EmrFileSystem: "true"
 sparkConf:
```

```
    # Required for EMR Runtime
    spark.driver.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/hadoop-
aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/share/aws/
emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/security/conf:/
usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-glue-datacatalog-
spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-serde.jar:/usr/share/aws/
sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/hadoop/extrajars/*
    spark.driver.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/lib/
native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/native
    spark.executor.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/hadoop-
aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/share/aws/
emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/security/conf:/
usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-glue-datacatalog-
spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-serde.jar:/usr/share/aws/
sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/hadoop/extrajars/*
    spark.executor.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/lib/
native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/native
```

Amazon EMR on EKS 服务端点和限额

以下是 Amazon EMR on EKS 的服务终端节点和服务配额。要以编程方式连接到 Amazon 服务，请使用终端节点。除标准 Amazon 终端节点外，某些 Amazon 服务还在选定区域提供 FIPS 终端节点。有关更多信息，请参阅Amazon 服务终端节点。服务限额（也称为限制）是 Amazon 账户使用的服务资源或操作的最大数量。有关更多信息，请参阅Amazon 服务配额。

服务端点

Amazon Web Services 区域 名字	代码	终端节点	协议
美国东部（弗吉尼亚州北部）	us-east-1	`emr-containers.us-east-1.amazonaws.com`	HTTPS
美国东部（俄亥俄州）	us-east-2	`emr-containers.us-east-2.amazonaws.com`	HTTPS
美国西部（加利福尼亚北部）	us-west-1	`emr-containers.us-west-1.amazonaws.com`	HTTPS
美国西部（俄勒冈）	us-west-2	`emr-containers.us-west-2.amazonaws.com`	HTTPS
亚太地区（东京）	ap-northeast-1	`emr-containers.ap-northeast-1.amazonaws.com`	HTTPS
亚太地区 (首尔)	ap-northeast-2	`emr-containers.ap-northeast-2.amazonaws.com`	HTTPS
亚太地区（大阪）	ap-northeast-3	`emr-containers.ap-northeast-3.amazonaws.com`	HTTPS
亚太地区（孟买）	ap-south-1	`emr-containers.ap-south-1.amazonaws.com`	HTTPS
亚太地区（海得拉巴）	ap-south-2	`emr-containers.ap-south-2.amazonaws.com`	HTTPS

Amazon Web Services 区域 名字	代码	终端节点	协议
亚太地区（新加坡）	ap-southeast-1	`emr-containers.ap-southeast-1.amazonaws.com`	HTTPS
亚太地区（悉尼）	ap-southeast-2	`emr-containers.ap-southeast-2.amazonaws.com`	HTTPS
亚太地区（雅加达）	ap-southeast-3	`emr-containers.ap-southeast-3.amazonaws.com`	HTTPS
亚太地区（香港）	ap-east-1	`emr-containers.ap-east-1.amazonaws.com`	HTTPS
非洲（开普敦）	af-south-1	`emr-containers.af-south-1.amazonaws.com`	HTTPS
加拿大（中部）	ca-central-1	`emr-containers.ca-central-1.amazonaws.com`	HTTPS
中国（宁夏）	cn-northwest-1	`emr-containers.cn-northwest-1.amazonaws.com.cn`	HTTPS
中国（北京）	cn-north-1	`emr-containers.cn-north-1.amazonaws.com.cn`	HTTPS
欧洲地区（法兰克福）	eu-central-1	`emr-containers.eu-central-1.amazonaws.com`	HTTPS
欧洲（苏黎世）	eu-central-2	`emr-containers.eu-central-2.amazonaws.com`	HTTPS
欧洲地区（爱尔兰）	eu-west-1	`emr-containers.eu-west-1.amazonaws.com`	HTTPS
欧洲地区（伦敦）	eu-west-2	`emr-containers.eu-west-2.amazonaws.com`	HTTPS

Amazon Web Services 区域 名字	代码	终端节点	协议
欧洲地区（巴黎）	eu-west-3	emr-containers.eu-west-3.amazonaws.com	HTTPS
欧洲地区（斯德哥尔摩）	eu-north-1	emr-containers.eu-north-1.amazonaws.com	HTTPS
欧洲地区（米兰）	eu-south-1	emr-containers.eu-south-1.amazonaws.com	HTTPS
欧洲（西班牙）	eu-south-2	emr-containers.eu-south-2.amazonaws.com	HTTPS
以色列（特拉维夫）	il-central-1	emr-containers.il-central-1.amazonaws.com	HTTPS
南美洲（圣保罗）	sa-east-1	emr-containers.sa-east-1.amazonaws.com	HTTPS
中东（阿联酋）	me-central-1	emr-containers.me-central-1.amazonaws.com	HTTPS
中东（巴林）	me-south-1	emr-containers.me-south-1.amazonaws.com	HTTPS
Amazon GovCloud（美国东部）	us-gov-east-1	emr-containers.us-gov-east-1.amazonaws.com	HTTPS
Amazon GovCloud（美国西部）	us-gov-west-1	emr-containers.us-gov-west-1.amazonaws.com	HTTPS

服务配额

EKS 上的 Amazon EMR 会按区域限制每个 Amazon 账户的以下 API 请求。有关如何应用限制的更多信息，请参阅 Amazon API 参考中的 API 请求限制。 EC2 您可以按照以下指南申请增加 Amazon 账户的 API 限制配额。

API 操作	存储桶最大容量	存储桶重填速率（每秒）
CancelJobRun	25	1
CreateJobTemplate	25	1
CreateManagedEndpoint	25	1
CreateSecurityConfiguration	25	1
CreateVirtualCluster	25	1
DeleteJobTemplate	25	1
DeleteManagedEndpoint	25	1
DeleteVirtualCluster	25	1
DescribeJobRun	100	20
DescribeJobTemplate	25	1
DescribeManagedEndpoint	100	5
DescribeSecurityConfiguration	25	1
DescribeVirtualCluster	100	5
GetManagedEndpoint SessionCredentials	25	1
ListJobRuns	100	5
ListJobTemplates	25	1
ListManagedEndpoints	25	1
ListSecurityConfigurations	25	1
ListVirtualClusters	100	5
StartJobRun	25	1

API 操作	存储桶最大容量	存储桶重填速率（每秒）
Throttle quota for all EMR on EKS API requests	200	20

在您创建 Amazon 账户时，我们会根据每个区域为您的 Amazon 资源设置默认配额（也称为限制）。如果超出资源的限额，请求会失败。如果发生这种情况，可以减少资源使用量或请求增加限额。

S ervice Q uotas 控制台是一个中心位置，您可以在其中查看和管理 Amazon 服务配额，并请求增加您使用的许多资源的配额。使用此处提供的配额信息来管理您的 Amazon 基础架构。请根据需要请提前计划以请求提高限额。

查看配额并申请增加配额

您可以使用 Service Quotas 控制台查看每个区域的当前服务配额。有关详细说明，请参阅 Service Quotas 用户指南中的查看服务配额。

您可以通过 Amazon 管理控制台或使用 Amazon CLI 申请增加配额。这些步骤详见申请增加配额。

Amazon EMR on EKS 版本

Amazon EMR 版本是一组来自大数据生态系统的开源应用程序。每个发行版由您在运行任务时选择用于部署和配置 Amazon EMR on EKS 不同的大数据应用程序、组件和功能组成。

从 Amazon EMR 版本 5.32.0 和 6.2.0 开始，您可以部署 Amazon EMR on EKS。此部署选项不适用于早期 Amazon EMR 发行版。提交任务时，必须指定受支持的发行版。

Amazon EMR on EKS 使用以下发行版标注形式：`emr-x.x.x-latest` 或者 `emr-x.x.x-yyyymmdd`，带具体的发布日期。例如，`emr-7.8.0-latest` 或 `emr-7.8.0-20210129`。使用 `-latest` 后缀时，确保您的 Amazon EMR 版本始终包含最新的安全更新。

> **Note**
>
> 有关 EKS 上的 Amazon EMR 和运行的亚马逊 EMR 的比较， EC2请参阅网站上的亚马逊 EMR。 FAQs Amazon

主题

- EKS 7.9.0 版本上的 Amazon EMR
- EKS 7.8.0 版本上的 Amazon EMR
- EKS 7.7.0 版本上的 Amazon EMR
- EKS 7.6.0 版本上的 Amazon EMR
- EKS 7.5.0 版本上的 Amazon EMR
- EKS 7.4.0 版本上的 Amazon EMR
- Amazon EMR on EKS 7.3.0 发行版
- Amazon EMR on EKS 7.2.0 发行版
- Amazon EMR on EKS 7.1.0 发行版
- EKS 上的 Amazon EMR 7.0.0 发行版
- EKS 上的 Amazon EMR 6.15.0 发行版
- Amazon EMR on EKS 6.14.0 发行版
- Amazon EMR on EKS 6.13.0 版本
- Amazon EMR on EKS 6.12.0 版本

- Amazon EMR on EKS 6.11.0 版本
- Amazon EMR on EKS 6.10.0 版本
- Amazon EMR on EKS 6.9.0 版本
- Amazon EMR on EKS 6.8.0 版本
- Amazon EMR on EKS 6.7.0 版本
- Amazon EMR on EKS 6.6.0 版本
- Amazon EMR on EKS 6.5.0 版本
- Amazon EMR on EKS 6.4.0 版本
- Amazon EMR on EKS 6.3.0 版本
- Amazon EMR on EKS 6.2.0 版本
- Amazon EMR on EKS 5.36.0 版本
- Amazon EMR on EKS 5.35.0 版本
- Amazon EMR on EKS 5.34.0 版本
- Amazon EMR on EKS 5.33.0 版本
- Amazon EMR on EKS 5.32.0 版本

EKS 7.9.0 版本上的 Amazon EMR

本页介绍了 Amazon EMR 的新增和更新的功能，这些功能特定于 Amazon EMR on EKS 部署。有关在亚马逊上运行的亚马逊 EMR EC2 以及亚马逊 EMR 7.9.0 版本的总体详情，请参阅《亚马逊 EMR 发布指南》中的 Amazon EM R 7.9. 0。

EKS 7.9 版本上的 Amazon EMR

以下亚马逊 EMR 7.9.0 版本可用于 EKS 上的亚马逊 EMR。选择特定的 EMR-7.9.0-xxxx 版本以查看更多详细信息，例如相关的容器映像标签。

Flink releases

当您运行 Flink 应用程序时，以下亚马逊 EMR 7.9.0 版本可用于 EKS 上的亚马逊 EMR。

- emr-7.9.0-flink-latest
- emr-7.9.0-flink-20250425

Spark releases

当您运行 Spark 应用程序时，以下亚马逊 EMR 7.9.0 版本可用于 EKS 上的亚马逊 EMR。

- emr-7.9.0-最新
- emr-7.9.0-20250425
- emr-7.9.0-spark-rapids-latest
- emr-7.9.0-spark-rapids-20250425
- emr-7.9.0-java11-latest
- emr-7.9.0-java11-20250425
- emr-7.9.0-java8-latest
- emr-7.9.0-java8-20250425
- emr-7.9.0-spark-rapids-java8-latest
- emr-7.9.0-spark-rapids-java8-20250425
- notebook-spark/emr-7.9.0-latest
- notebook-spark/emr-7.9.0-20250425
- notebook-spark/emr-7.9.0-spark-rapids-latest
- notebook-spark/emr-7.9.0-spark-rapids-20250425
- notebook-spark/emr-7.9.0-java11-latest
- notebook-spark/emr-7.9.0-java11-20250425
- notebook-spark/emr-7.9.0-java8-latest
- notebook-spark/emr-7.9.0-java8-20250425
- notebook-spark/emr-7.9.0-spark-rapids-java8-latest
- notebook-spark/emr-7.9.0-spark-rapids-java8-20250425
- notebook-python/emr-7.9.0-latest
- notebook-python/emr-7.9.0-20250425
- notebook-python/emr-7.9.0-spark-rapids-latest
- notebook-python/emr-7.9.0-spark-rapids-20250425
- notebook-python/emr-7.9.0-java11-latest
- notebook-python/emr-7.9.0-java11-20250425
- notebook-python/emr-7.9.0-java8-latest
- notebook-python/emr-7.9.0-java8-20250425

- notebook-python/emr-7.9.0-spark-rapids-java8-latest
- notebook-python/emr-7.9.0-spark-rapids-java8-20250425
- livy/emr-7.9.0-latest
- livy/emr-7.9.0-20250425
- livy/emr-7.9.0-java11-latest
- livy/emr-7.9.0-java11-20250425
- livy/emr-7.9.0-java8-latest
- livy/emr-7.9.0-java8-20250425

发行说明

EKS 7.9.0 上亚马逊 EMR 的发布说明

- 支持的应用程序 - 适用于 Java 的 Amazon SDK 2.31.16 and 1.12.782, Apache Spark 3.5.5, Apache Hudi 0.15.0-amzn-6, Apache Iceberg 1.7.1-amzn-2, Delta 3.3.0-amzn-1, Apache Spark RAPIDS 25.02.1-amzn-0, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.20.0-amzn-3, Flink Operator 1.10.0-amzn-3
- 支持的组件-emr-ddb `emr-goodies`、`emr-s3-select`、、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。
- 支持的配置分类

  与[StartJobRun](#)和一起使用 [CreateManagedEndpoint](#) APIs：

分类	描述
`core-site`	更改 `core-site.xml` Hadoop 文件中的值。
`emrfs-site`	更改 EMRFS 设置。
`spark-metrics`	更改 `metrics.properties` Spark 文件中的值。
`spark-defaults`	更改 `spark-defaults.conf` Spark 文件中的值。

分类	描述
spark-env	更改 Spark 环境中的值。
spark-hive-site	更改 hive-site.xml Spark 文件中的值。
spark-log4j2	更改 log4j2.properties Spark 文件中的值。
emr-job-submitter	任务提交者 Pod 的配置。

专门用于以下用途 CreateManagedEndpoint APIs：

分类	描述
jeg-config	更改 Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 文件中的值。
jupyter-kernel-overrides	更改 Jupyter 内核规范文件中内核映像的值。

配置分类允许您自定义应用程序。这些通常与应用程序的配置 XML 文件（例如 spark-hive-site.xml）相对应。有关更多信息，请参阅配置应用程序。

更改

EKS 上的 Amazon EMR 7.9.0 版本中包含以下更改：

- 该版本没有变化。

emr-7.9.0-最新

发布说明：emr-7.9.0-latest 当前指向 emr-7.9.0-20250425。

区域：emr-7.9.0-latest 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.9.0:latest`

emr-7.9.0-20250425

版本说明：`emr-7.9.0-20250425`已于 2025 年 2 月发布。这是亚马逊 EMR 7.9.0 (Spark) 的初始版本。

区域：`emr-emr-7.9.0-20250425` 已在支持 Amazon EMR on EKS 的所有区域开放。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.9.0-20250425`

emr-7.9.0-flink-latest

发行说明：`emr-7.9.0-flink-latest`目前指向 `emr-7.9.0-flink-20250425`

区域：`emr-7.9.0-flink-latest` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.9.0-flink:latest`

emr-7.9.0-flink-20250425

版本说明：`7.9.0-flink-20250425`已于 2025 年 2 月发布。这是亚马逊 EMR 7.9.0 (Flink) 的初始版本。

区域：`emr-7.9.0-flink-20250425` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.9.0-flink:20250425`

EKS 7.8.0 版本上的 Amazon EMR

本页介绍了 Amazon EMR 的新增和更新的功能，这些功能特定于 Amazon EMR on EKS 部署。有关在亚马逊上运行的亚马逊 EMR EC2 以及亚马逊 EMR 7.8.0 版本的总体详情，请参阅亚马逊 EMR 发布指南中的亚马逊 EM R 7.8. 0。

EKS 7.8 版本上的 Amazon EMR

以下亚马逊 EMR 7.8.0 版本可用于 EKS 上的亚马逊 EMR。选择特定的 EMR-7.8.0-xxxx 版本以查看更多详细信息，例如相关的容器映像标签。

Flink releases

当您运行 Flink 应用程序时，以下亚马逊 EMR 7.8.0 版本可用于 EKS 上的亚马逊 EMR。

- emr-7.8.0-flink-latest
- emr-7.8.0-flink-20250228

Spark releases

当您运行 Spark 应用程序时，以下亚马逊 EMR 7.8.0 版本可用于 EKS 上的亚马逊 EMR。

- emr-7.8.0-最新
- emr-7.8.0-20250228
- emr-7.8.0-spark-rapids-latest
- emr-7.8.0-spark-rapids-20250228
- emr-7.8.0-java11-latest
- emr-7.8.0-java11-20250228
- emr-7.8.0-java8-latest
- emr-7.8.0-java8-20250228
- emr-7.8.0-spark-rapids-java8-latest
- emr-7.8.0-spark-rapids-java8-20250228
- notebook-spark/emr-7.8.0-latest
- notebook-spark/emr-7.8.0-20250228
- notebook-spark/emr-7.8.0-spark-rapids-latest
- notebook-spark/emr-7.8.0-spark-rapids-20250228
- notebook-spark/emr-7.8.0-java11-latest
- notebook-spark/emr-7.8.0-java11-20250228
- notebook-spark/emr-7.8.0-java8-latest
- notebook-spark/emr-7.8.0-java8-20250228
- notebook-spark/emr-7.8.0-spark-rapids-java8-latest
- notebook-spark/emr-7.8.0-spark-rapids-java8-20250228
- notebook-python/emr-7.8.0-latest
- notebook-python/emr-7.8.0-20250228

- notebook-python/emr-7.8.0-spark-rapids-latest
- notebook-python/emr-7.8.0-spark-rapids-20250228
- notebook-python/emr-7.8.0-java11-latest
- notebook-python/emr-7.8.0-java11-20250228
- notebook-python/emr-7.8.0-java8-latest
- notebook-python/emr-7.8.0-java8-20250228
- notebook-python/emr-7.8.0-spark-rapids-java8-latest
- notebook-python/emr-7.8.0-spark-rapids-java8-20250228
- livy/emr-7.8.0-latest
- livy/emr-7.8.0-20250228
- livy/emr-7.8.0-java11-latest
- livy/emr-7.8.0-java11-20250228
- livy/emr-7.8.0-java8-latest
- livy/emr-7.8.0-java8-20250228

发行说明

EKS 7.8.0 上亚马逊 EMR 的发布说明

- 支持的应用程序 - 适用于 Java 的 Amazon SDK 2.29.52 and 1.12.780, Apache Spark 3.5.4, Apache Hudi 0.15.0-amzn-5, Apache Iceberg 1.7.1-amzn-1, Delta 3.3.0-amzn-0, Apache Spark RAPIDS 24.12.0-amzn-0, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.20.0-amzn-2, Flink Operator 1.10.0-amzn-2
- 支持的组件-`emr-ddb` `emr-goodies`、`emr-s3-select`、、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。
- 支持的配置分类

  与StartJobRun和一起使用 CreateManagedEndpoint APIs：

分类	描述
`core-site`	更改 `core-site.xml` Hadoop 文件中的值。

分类	描述
emrfs-site	更改 EMRFS 设置。
spark-metrics	更改 metrics.properties　Spark 文件中的值。
spark-defaults	更改 spark-defaults.conf　Spark 文件中的值。
spark-env	更改 Spark 环境中的值。
spark-hive-site	更改 hive-site.xml　Spark 文件中的值。
spark-log4j2	更改 log4j2.properties　Spark 文件中的值。
emr-job-submitter	任务提交者 Pod 的配置。

专门用于以下用途 CreateManagedEndpoint APIs：

分类	描述
jeg-config	更改 Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py　文件中的值。
jupyter-kernel-overrides	更改 Jupyter 内核规范文件中内核映像的值。

配置分类允许您自定义应用程序。这些通常与应用程序的配置 XML 文件（例如 spark-hive-site.xml）相对应。有关更多信息，请参阅配置应用程序。

更改

EKS 上的 Amazon EMR 7.8.0 版本中包含以下更改：

- 原生 FGAC 功能，包括：

  - Iceberg 支持在精细访问控制 (FGAC) 虚拟集群中运行对非湖形成表执行操作的作业。（IAM 有一个后备方案。）
  - S3 表格支持
- 火花连接

emr-7.8.0-最新

发布说明：`emr-7.8.0-latest` 当前指向 `emr-7.8.0-20250228`。

区域：`emr-7.8.0-latest` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.8.0:latest`

emr-7.8.0-20250228

版本说明：`emr-7.8.0-20250228`已于 2025 年 2 月发布。这是亚马逊 EMR 7.8.0 (Spark) 的初始版本。

区域：`emr-emr-7.8.0-20250228` 已在支持 Amazon EMR on EKS 的所有区域开放。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.8.0-20250228`

emr-7.8.0-flink-latest

发行说明：`emr-7.8.0-flink-latest`目前指向 `emr-7.8.0-flink-20250228`

区域：`emr-7.8.0-flink-latest` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.8.0-flink:latest`

emr-7.8.0-flink-20250228

版本说明：`7.8.0-flink-20250228`已于 2025 年 2 月发布。这是亚马逊 EMR 7.8.0 (Flink) 的初始版本。

区域：`emr-7.8.0-flink-20250228` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.8.0-flink:20250228`

EKS 7.7.0 版本上的 Amazon EMR

本页介绍了 Amazon EMR 的新增和更新的功能，这些功能特定于 Amazon EMR on EKS 部署。有关在亚马逊上运行的亚马逊 EMR EC2 以及亚马逊 EMR 7.7.0 版本的总体详情，请参阅《亚马逊 EMR 发布指南》中的亚马逊 EM R 7.7. 0。

EKS 7.7 版本上的 Amazon EMR

以下亚马逊 EMR 7.7.0 版本可用于 EKS 上的亚马逊 EMR。选择特定的 EMR-7.7.0-xxxx 版本以查看更多详细信息，例如相关的容器映像标签。

Flink releases

当您运行 Flink 应用程序时，以下亚马逊 EMR 7.7.0 版本可用于 EKS 上的亚马逊 EMR。

- emr-7.0-flink-latest
- emr-7.0-flink-20250131

Spark releases

当您运行 Spark 应用程序时，以下亚马逊 EMR 7.7.0 版本可用于 EKS 上的亚马逊 EMR。

- emr-7.7.0-最新
- emr-7.0-20250131
- emr-7.7.0-spark-rapids-latest
- emr-7.7.0-spark-rapids-20250131
- emr-7.7.0-java11-latest
- emr-7.7.0-java11-20250131
- emr-7.7.0-java8-latest
- emr-7.7.0-java8-20250131
- emr-7.7.0-spark-rapids-java8-latest
- emr-7.7.0-spark-rapids-java8-20250131
- notebook-spark/emr-7.7.0-latest
- notebook-spark/emr-7.7.0-20250131

- notebook-spark/emr-7.7.0-spark-rapids-latest
- notebook-spark/emr-7.7.0-spark-rapids-20250131
- notebook-spark/emr-7.7.0-java11-latest
- notebook-spark/emr-7.7.0-java11-20250131
- notebook-spark/emr-7.7.0-java8-latest
- notebook-spark/emr-7.7.0-java8-20250131
- notebook-spark/emr-7.7.0-spark-rapids-java8-latest
- notebook-spark/emr-7.7.0-spark-rapids-java8-20250131
- notebook-python/emr-7.7.0-latest
- notebook-python/emr-7.7.0-20250131
- notebook-python/emr-7.7.0-spark-rapids-latest
- notebook-python/emr-7.7.0-spark-rapids-20250131
- notebook-python/emr-7.7.0-java11-latest
- notebook-python/emr-7.7.0-java11-20250131
- notebook-python/emr-7.7.0-java8-latest
- notebook-python/emr-7.7.0-java8-20250131
- notebook-python/emr-7.7.0-spark-rapids-java8-latest
- notebook-python/emr-7.7.0-spark-rapids-java8-20250131
- livy/emr-7.7.0-latest
- livy/emr-7.7.0-20250131
- livy/emr-7.7.0-java11-latest
- livy/emr-7.7.0-java11-20250131
- livy/emr-7.7.0-java8-latest
- livy/emr-7.7.0-java8-20250131

发行说明

EKS 7.7.0 上亚马逊 EMR 的发布说明

- 支持的应用程序 - 适用于 Java 的 Amazon SDK 2.29.25 and 1.12.779, Apache Spark 3.5.3-amzn-0, Apache Hudi 0.15.0-amzn-3, Apache Iceberg 1.6.1-amzn-2, Delta 3.2.1-amzn-1, Apache Spark

RAPIDS 24.10.1-amzn-0, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.20.0-amzn-0, Flink Operator 1.10.0-amzn-0

- 支持的组件-emr-ddb emr-goodies、emr-s3-select、、emrfs、hadoop-client、hudi、hudi-spark、iceberg、spark-kubernetes。
- 支持的配置分类

与StartJobRun和一起使用 CreateManagedEndpoint APIs：

分类	描述
core-site	更改 core-site.xml Hadoop 文件中的值。
emrfs-site	更改 EMRFS 设置。
spark-metrics	更改 metrics.properties Spark 文件中的值。
spark-defaults	更改 spark-defaults.conf Spark 文件中的值。
spark-env	更改 Spark 环境中的值。
spark-hive-site	更改 hive-site.xml Spark 文件中的值。
spark-log4j2	更改 log4j2.properties Spark 文件中的值。
emr-job-submitter	任务提交者 Pod 的配置。

专门用于以下用途 CreateManagedEndpoint APIs：

分类	描述
jeg-config	更改 Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 文件中的值。

分类	描述
`jupyter-kernel-overrides`	更改 Jupyter 内核规范文件中内核映像的值。

配置分类允许您自定义应用程序。这些通常与应用程序的配置 XML 文件（例如 `spark-hive-site.xml`）相对应。有关更多信息，请参阅配置应用程序。

更改

EKS 上的 Amazon EMR 7.7.0 版本中包含以下更改：

- 自 EMR 7.7.0 起使用的 Iceberg 版本不再支持 Java 8。此外，以下 Java 8 图像中不包括 Iceberg：`emr-7.7.0-java8-latest`和。`emr-7.7.0-spark-rapids-java8-latest`

emr-7.7.0-最新

发布说明：`emr-7.7.0-latest` 当前指向 `emr-7.7.0-20250131`。

区域：`emr-7.7.0-latest` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.7.0:latest`

emr-7.0-20250131

版本说明：`emr-7.7.0-20250131`已于 2025 年 2 月发布。这是亚马逊 EMR 7.7.0 (Spark) 的初始版本。

区域：`emr-emr-7.7.0-20250131` 已在支持 Amazon EMR on EKS 的所有区域开放。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.7.0-20250131`

emr-7.0-flink-latest

发行说明：`emr-7.7.0-flink-latest`目前指向 `emr-7.7.0-flink-20250131`

区域：`emr-7.7.0-flink-latest` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.7.0-flink:latest`

emr-7.0-flink-20250131

版本说明：`7.7.0-flink-20250131`已于 2025 年 2 月发布。这是亚马逊 EMR 7.7.0 (Flink) 的初始版本。

区域：`emr-7.7.0-flink-20250131` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.7.0-flink:20250131`

EKS 7.6.0 版本上的 Amazon EMR

本页介绍了 Amazon EMR 的新增和更新的功能，这些功能特定于 Amazon EMR on EKS 部署。有关在亚马逊上运行的亚马逊 EMR EC2 以及亚马逊 EMR 7.6.0 版本的总体详情，请参阅《亚马逊 EMR 发布指南》中的 Amazon EM R 7.6. 0。

EKS 7.6 版本上的 Amazon EMR

以下亚马逊 EMR 7.6.0 版本可用于 EKS 上的亚马逊 EMR。选择特定的 EMR-7.6.0-xxxx 版本以查看更多详细信息，例如相关的容器映像标签。

Flink releases

当您运行 Flink 应用程序时，以下亚马逊 EMR 7.6.0 版本可用于 EKS 上的亚马逊 EMR。

- emr-7.6.0-flink-latest
- emr-7.6.0-flink-20241213

Spark releases

当您运行 Spark 应用程序时，以下亚马逊 EMR 7.6.0 版本可用于 EKS 上的亚马逊 EMR。

- emr-7.6.0-最新
- emr-7.6.0-20241213
- emr-7.6.0-spark-rapids-latest
- emr-7.6.0-spark-rapids-20241213

- emr-7.6.0-java11-latest
- emr-7.6.0-java11-20241213
- emr-7.6.0-java8-latest
- emr-7.6.0-java8-20241213
- emr-7.6.0-spark-rapids-java8-latest
- emr-7.6.0-spark-rapids-java8-20241213
- notebook-spark/emr-7.6.0-latest
- notebook-spark/emr-7.6.0-20241213
- notebook-spark/emr-7.6.0-spark-rapids-latest
- notebook-spark/emr-7.6.0-spark-rapids-20241213
- notebook-spark/emr-7.6.0-java11-latest
- notebook-spark/emr-7.6.0-java11-20241213
- notebook-spark/emr-7.6.0-java8-latest
- notebook-spark/emr-7.6.0-java8-20241213
- notebook-spark/emr-7.6.0-spark-rapids-java8-latest
- notebook-spark/emr-7.6.0-spark-rapids-java8-20241213
- notebook-python/emr-7.6.0-latest
- notebook-python/emr-7.6.0-20241213
- notebook-python/emr-7.6.0-spark-rapids-latest
- notebook-python/emr-7.6.0-spark-rapids-20241213
- notebook-python/emr-7.6.0-java11-latest
- notebook-python/emr-7.6.0-java11-20241213
- notebook-python/emr-7.6.0-java8-latest
- notebook-python/emr-7.6.0-java8-20241213
- notebook-python/emr-7.6.0-spark-rapids-java8-latest
- notebook-python/emr-7.6.0-spark-rapids-java8-20241213
- livy/emr-7.6.0-latest
- livy/emr-7.6.0-20241213
- livy/emr-7.6.0-java11-latest
- livy/emr-7.6.0-java11-20241213

- livy/emr-7.6.0-java8-latest
- livy/emr-7.6.0-java8-20241213

发行说明

EKS 7.6.0 上亚马逊 EMR 的发布说明

- 支持的应用程序 - 适用于 Java 的 Amazon SDK 2.29.25 and 1.12.779, Apache Spark 3.5.3-amzn-0, Apache Hudi 0.15.0-amzn-3, Apache Iceberg 1.6.1-amzn-2, Delta 3.2.1-amzn-1, Apache Spark RAPIDS 24.10.1-amzn-0, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.20.0-amzn-0, Flink Operator 1.10.0-amzn-0
- 支持的组件 - `aws-sagemaker-spark-sdk`、`emr-ddb`、`emr-goodies`、`emr-s3-select`、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。
- 支持的配置分类

  与StartJobRun和一起使用 CreateManagedEndpoint APIs：

分类	描述
`core-site`	更改 `core-site.xml` Hadoop 文件中的值。
`emrfs-site`	更改 EMRFS 设置。
`spark-metrics`	更改 `metrics.properties` Spark 文件中的值。
`spark-defaults`	更改 `spark-defaults.conf` Spark 文件中的值。
`spark-env`	更改 Spark 环境中的值。
`spark-hive-site`	更改 `hive-site.xml` Spark 文件中的值。
`spark-log4j2`	更改 `log4j2.properties` Spark 文件中的值。
`emr-job-submitter`	任务提交者 Pod 的配置。

专门用于以下用途 CreateManagedEndpoint APIs：

分类	描述
jeg-config	更改 Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 文件中的值。
jupyter-kernel-overrides	更改 Jupyter 内核规范文件中内核映像的值。

配置分类允许您自定义应用程序。这些通常与应用程序的配置 XML 文件（例如 spark-hive-site.xml）相对应。有关更多信息，请参阅配置应用程序。

显著功能

EKS 上的 Amazon EMR 7.6.0 版本包含以下功能：

- Apache Spark 操作员的监控配置支持 — 监控配置可让您轻松设置 Spark 应用程序和操作员日志的日志存档到亚马逊 S3 或亚马逊 CloudWatch。您可以选择一个或两个。这样做会将日志代理 sidecar 添加到 Spark 操作员窗格、驱动程序和执行器窗格，然后将这些组件的日志转发到您配置的接收器。有关更多信息，请参阅使用监控配置监控 Spark Kubernetes 操作员和 Spark 作业。

更改

EKS 上的 Amazon EMR 7.6.0 版本中包含以下更改：

- 该版本没有变化。

emr-7.6.0-最新

发布说明：emr-7.6.0-latest 当前指向 emr-7.6.0-20241213。

区域：emr-7.6.0-latest 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：emr-7.6.0:latest

emr-7.6.0-20241213

发行说明：`7.6.0-20241213`已于 2024 年 1 月发布。这是亚马逊 EMR 7.6.0 (Spark) 的初始版本。

区域：`emr-7.6.0-20241213` 已在支持 Amazon EMR on EKS 的所有区域开放。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.6.0:20241213`

emr-7.6.0-flink-latest

发行说明：`emr-7.6.0-flink-latest`目前指向 `emr-7.6.0-flink-20241213`

区域：`emr-7.6.0-flink-latest` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.6.0-flink:latest`

emr-7.6.0-flink-20241213

发行说明：`7.6.0-flink-20241213`已于 2024 年 1 月发布。这是亚马逊 EMR 7.6.0 (Flink) 的初始版本。

区域：`emr-7.6.0-flink-20241213` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.6.0-flink:20241213`

EKS 7.5.0 版本上的 Amazon EMR

本页介绍了 Amazon EMR 的新增和更新的功能，这些功能特定于 Amazon EMR on EKS 部署。有关在亚马逊上运行的亚马逊 EMR EC2 以及亚马逊 EMR 7.5.0 版本的总体详情，请参阅《亚马逊 EMR 发布指南》中的 Amazon EMR 7.5.0。

EKS 7.5 版本上的 Amazon EMR

以下亚马逊 EMR 7.5.0 版本可用于 EKS 上的亚马逊 EMR。选择特定的 EMR-7.5.0-xxxx 版本以查看更多详细信息，例如相关的容器映像标签。

发行说明

EKS 7.5.0 上亚马逊 EMR 的发行说明

- 支持的应用程序 - 适用于 Java 的 Amazon SDK 2.28.8 and 1.12.772, Apache Spark 3.5.2-amzn-1, Apache Hudi 0.15.0-amzn-1, Apache Iceberg 1.6.1-amzn-0, Delta 3.2.0-amzn-1, Apache Spark RAPIDS 24.08.1-amzn-1, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.19.1-amzn-1, Flink Operator 1.9.0-amzn-0
- 支持的组件 - `aws-sagemaker-spark-sdk`、`emr-ddb`、`emr-goodies`、`emr-s3-select`、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。

EKS 7.4.0 版本上的 Amazon EMR

本页介绍了 Amazon EMR 的新增和更新的功能，这些功能特定于 Amazon EMR on EKS 部署。有关在亚马逊上运行的亚马逊 EMR EC2 以及亚马逊 EMR 7.4.0 版本的总体详情，请参阅《亚马逊 EMR 发布指南》[中的亚马逊 EMR 7.4.0](https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-740-release.html)。

EKS 7.4 版本上的 Amazon EMR

以下亚马逊 EMR 7.4.0 版本可用于 EKS 上的亚马逊 EMR。选择特定的 EMR-7.4.0-xxxx 版本以查看更多详细信息，例如相关的容器映像标签。

发行说明

EKS 7.4.0 上亚马逊 EMR 的发布说明

- 支持的应用程序 - 适用于 Java 的 Amazon SDK 2.25.70 and 1.12.772, Apache Spark 3.5.2-amzn-0, Apache Hudi 0.15.0-amzn-1, Apache Iceberg 1.6.1-amzn-0, Delta 3.2.0-amzn-1, Apache Spark RAPIDS 24.08.1-amzn-0, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.19.1-amzn-0, Flink Operator 1.9.0-amzn-1
- 支持的组件 - `aws-sagemaker-spark-sdk`、`emr-ddb`、`emr-goodies`、`emr-s3-select`、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。

Amazon EMR on EKS 7.3.0 发行版

本页介绍了 Amazon EMR 的新增和更新的功能，这些功能特定于 Amazon EMR on EKS 部署。有关在亚马逊上运行的亚马逊 EMR EC2 以及亚马逊 EMR 7.3.0 版本的总体详情，请参阅《亚马逊 EMR 发布指南》中的亚马逊 EMR 7.3.0。

Amazon EMR on EKS 7.3 发行版

以下 Amazon EMR 7.3.0 发行版适用于 Amazon EMR on EKS。选择特定的 emr-7.3.0-XXXX 发行版以查看更多详细信息，例如相关的容器映像标签。

Flink releases

在运行 Flink 应用程序时，以下 Amazon EMR 7.3.0 发行版适用于 Amazon EMR on EKS。

- emr-7.3.0-flink-latest
- emr-7.3.0-flink-29240920

Spark releases

在运行 Spark 应用程序时，以下 Amazon EMR 7.3.0 发行版适用于 Amazon EMR on EKS。

- emr-7.3.0-latest
- emr-7.3.0-29240920
- emr-7.3.0-spark-rapids-latest
- emr-7.3.0-spark-rapids-29240920
- emr-7.3.0-java11-latest
- emr-7.3.0-java11-29240920
- emr-7.3.0-java8-latest
- emr-7.3.0-java8-29240920
- emr-7.3.0-spark-rapids-java8-latest
- emr-7.3.0-spark-rapids-java8-29240920
- notebook-spark/emr-7.3.0-latest
- notebook-spark/emr-7.3.0-29240920
- notebook-spark/emr-7.3.0-spark-rapids-latest
- notebook-spark/emr-7.3.0-spark-rapids-29240920
- notebook-spark/emr-7.3.0-java11-latest
- notebook-spark/emr-7.3.0-java11-29240920
- notebook-spark/emr-7.3.0-java8-latest
- notebook-spark/emr-7.3.0-java8-29240920

- notebook-spark/emr-7.3.0-spark-rapids-java8-latest
- notebook-spark/emr-7.3.0-spark-rapids-java8-29240920
- notebook-python/emr-7.3.0-latest
- notebook-python/emr-7.3.0-29240920
- notebook-python/emr-7.3.0-spark-rapids-latest
- notebook-python/emr-7.3.0-spark-rapids-29240920
- notebook-python/emr-7.3.0-java11-latest
- notebook-python/emr-7.3.0-java11-29240920
- notebook-python/emr-7.3.0-java8-latest
- notebook-python/emr-7.3.0-java8-29240920
- notebook-python/emr-7.3.0-spark-rapids-java8-latest
- notebook-python/emr-7.3.0-spark-rapids-java8-29240920
- livy/emr-7.3.0-latest
- livy/emr-7.3.0-29240920
- livy/emr-7.3.0-java11-latest
- livy/emr-7.3.0-java11-29240920
- livy/emr-7.3.0-java8-latest
- livy/emr-7.3.0-java8-29240920

发行说明

Amazon EMR on EKS 7.3.0 的发布说明

- 支持的应用程序 - 适用于 Java 的 Amazon SDK 2.25.70 and 1.12.747, Apache Spark 3.5.1-amzn-1, Apache Hudi 0.15.0-amzn-0, Apache Iceberg 1.5.2-amzn-0, Delta 3.2.0-amzn-0, Apache Spark RAPIDS 24.06.1-amzn-0, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.18.1-amzn-2, Flink Operator 1.9.0-amzn-0
- 支持的组件 - `aws-sagemaker-spark-sdk`、`emr-ddb`、`emr-goodies`、`emr-s3-select`、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。
- 支持的配置分类

  与StartJobRun和一起使用 CreateManagedEndpoint APIs：

分类	描述
core-site	更改 core-site.xml Hadoop 文件中的值。
emrfs-site	更改 EMRFS 设置。
spark-metrics	更改 metrics.properties Spark 文件中的值。
spark-defaults	更改 spark-defaults.conf Spark 文件中的值。
spark-env	更改 Spark 环境中的值。
spark-hive-site	更改 hive-site.xml Spark 文件中的值。
spark-log4j2	更改 log4j2.properties Spark 文件中的值。
emr-job-submitter	任务提交者 Pod 的配置。

专门用于以下用途 CreateManagedEndpoint APIs：

分类	描述
jeg-config	更改 Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 文件中的值。
jupyter-kernel-overrides	更改 Jupyter 内核规范文件中内核映像的值。

配置分类允许您自定义应用程序。这些通常与应用程序的配置 XML 文件（例如 spark-hive-site.xml）相对应。有关更多信息，请参阅配置应用程序。

显著功能

Amazon EMR on EKS 7.3.0 发行版包含以下功能。

- 应用程序升级：Amazon EMR on EKS 现在包含 Flink Operator 1.9.0。除了其他功能外，Flink Kubernetes 现在还允许为 Autoscaler 设置 CPU 和内存配额。
- Apache Iceberg 支持 Apache Flink：Apache Iceberg 是一种开源的高性能格式大型分析表。从 Amazon EMR 7.3.0 开始，在 Amazon EMR on EKS 上运行 Apache Flink 时，可使用 Apache Iceberg 表。有关更多信息，请参阅 Amazon EMR on EKS 将 Apache Iceberg 与 Amazon EMR on EKS 结合使用。
- Delta Lake 支持 Apache Flink：Delta Lake 是一种存储层框架，适用于通常基于 Amazon S3 构建的湖屋架构。对于 Amazon EMR 7.3.0 及更高版本，在 Amazon EMR on EKS 上运行 Apache Flink 时，可使用 Delta 表。有关更多信息，请参阅将 Delta Lake 与 Amazon EMR on EKS 结合使用。

更改

Amazon EMR on EKS 发行版 7.3.0 包含以下更改。

- 对于 Amazon EMR on EKS 7.3.0 及更高版本，Apache Flink 现在默认使用 Java 17 运行时。

emr-7.3.0-latest

发布说明：`emr-7.3.0-latest` 当前指向 `emr-7.3.0-29240920`。

区域：`emr-7.3.0-latest` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.3.0:latest`

emr-7.3.0-29240920

发布说明：`7.3.0-29240920` 已于 2023 年 12 月发布。这是 Amazon EMR 7.3.0（Spark）的初始发行版。

区域：`emr-7.3.0-29240920` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.3.0:29240920`

emr-7.3.0-flink-latest

发布说明：`emr-7.3.0-flink-latest` 当前指向 `emr-7.3.0-flink-29240920`。

区域：`emr-7.3.0-flink-latest` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.3.0-flink:latest`

emr-7.3.0-flink-29240920

发布说明：`7.3.0-flink-29240920` 已于 2023 年 12 月发布。这是 Amazon EMR 7.3.0（Flink）的初始发行版。

区域：`emr-7.3.0-flink-29240920` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.3.0-flink:29240920`

Amazon EMR on EKS 7.2.0 发行版

本页介绍了 Amazon EMR 的新增和更新的功能，这些功能特定于 Amazon EMR on EKS 部署。有关在亚马逊上运行的亚马逊 EMR EC2 以及亚马逊 EMR 7.2.0 版本的总体详情，请参阅亚马逊 EMR 发布指南中的亚马逊 EMR 7.2.0。

Amazon EMR on EKS 7.2 发行版

以下 Amazon EMR 7.2.0 发行版适用于 Amazon EMR on EKS。选择特定的 emr-7.2.0-XXXX 发行版以查看更多详细信息，例如相关的容器映像标签。

Flink releases

在运行 Flink 应用程序时，以下 Amazon EMR 7.2.0 发行版适用于 Amazon EMR on EKS。

- emr-7.2.0-flink-latest
- emr-7.2.0-flink-20240610

Spark releases

在运行 Spark 应用程序时，以下 Amazon EMR 7.2.0 发行版适用于 Amazon EMR on EKS。

- emr-7.2.0-latest
- emr-7.2.0-20240610
- emr-7.2.0-spark-rapids-latest
- emr-7.2.0-spark-rapids-20240610
- emr-7.2.0-java11-latest
- emr-7.2.0-java11-20240610
- emr-7.2.0-java8-latest
- emr-7.2.0-java8-20240610
- emr-7.2.0-spark-rapids-java8-latest
- emr-7.2.0-spark-rapids-java8-20240610
- notebook-spark/emr-7.2.0-latest
- notebook-spark/emr-7.2.0-20240610
- notebook-spark/emr-7.2.0-spark-rapids-latest
- notebook-spark/emr-7.2.0-spark-rapids-20240610
- notebook-spark/emr-7.2.0-java11-latest
- notebook-spark/emr-7.2.0-java11-20240610
- notebook-spark/emr-7.2.0-java8-latest
- notebook-spark/emr-7.2.0-java8-20240610
- notebook-spark/emr-7.2.0-spark-rapids-java8-latest
- notebook-spark/emr-7.2.0-spark-rapids-java8-20240610
- notebook-python/emr-7.2.0-latest
- notebook-python/emr-7.2.0-20240610
- notebook-python/emr-7.2.0-spark-rapids-latest
- notebook-python/emr-7.2.0-spark-rapids-20240610
- notebook-python/emr-7.2.0-java11-latest
- notebook-python/emr-7.2.0-java11-20240610
- notebook-python/emr-7.2.0-java8-latest
- notebook-python/emr-7.2.0-java8-20240610

- notebook-python/emr-7.2.0-spark-rapids-java8-latest
- notebook-python/emr-7.2.0-spark-rapids-java8-20240610
- livy/emr-7.2.0-latest
- livy/emr-7.2.0-20240610
- livy/emr-7.2.0-java11-latest
- livy/emr-7.2.0-java11-20240610
- livy/emr-7.2.0-java8-latest
- livy/emr-7.2.0-java8-20240610

发行说明

Amazon EMR on EKS 7.2.0 的发布说明

- 支持的应用程序 - 适用于 Java 的 Amazon SDK 2.23.18 and 1.12.705, Apache Spark 3.5.1-amzn-1, Apache Hudi 0.14.1-amzn-0, Apache Iceberg 1.5.0-amzn-0, Delta 3.1.0, Apache Spark RAPIDS 24.02.0-amzn-1, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.18.1-amzn-0, Flink Operator 1.8.0-amzn-1
- 支持的组件 - `aws-sagemaker-spark-sdk`、`emr-ddb`、`emr-goodies`、`emr-s3-select`、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。
- 支持的配置分类

  与StartJobRun和一起使用 CreateManagedEndpoint APIs：

分类	描述
`core-site`	更改 `core-site.xml` Hadoop 文件中的值。
`emrfs-site`	更改 EMRFS 设置。
`spark-metrics`	更改 `metrics.properties` Spark 文件中的值。
`spark-defaults`	更改 `spark-defaults.conf` Spark 文件中的值。

分类	描述
spark-env	更改 Spark 环境中的值。
spark-hive-site	更改 hive-site.xml Spark 文件中的值。
spark-log4j2	更改 log4j2.properties Spark 文件中的值。
emr-job-submitter	任务提交者 Pod 的配置。

专门用于以下用途 CreateManagedEndpoint APIs：

分类	描述
jeg-config	更改 Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 文件中的值。
jupyter-kernel-overrides	更改 Jupyter 内核规范文件中内核映像的值。

配置分类允许您自定义应用程序。这些通常与应用程序的配置 XML 文件（例如 spark-hive-site.xml）相对应。有关更多信息，请参阅配置应用程序。

显著功能

Amazon EMR on EKS 7.2.0 发行版包含以下功能。

- 应用程序升级：Amazon EMR on EKS 7.2.0 应用程序升级包括 Spark 3.5.1、Flink 1.18.1 和 Flink Operator 1.8.0。
- 适用于 Flink 的 Autoscaler 更新：7.2.0 发行版使用开源配置 job.autoscaler.restart.time-tracking.enabled 来启用重新缩放时间估计，因此无需再为重启时间手动分配经验值。如果运行的是 7.1.0 或更低版本，您仍然可以使用 Amazon EMR 自动扩缩。
- Apache Hudi 在 Amazon EMR on EKS 上集成 Apache Flink：此发行版增加了 Apache Hudi 和 Apache Flink 之间的集成，因此您可以使用 Flink Kubernetes 运算符来运行 Hudi 作业。Hudi 允许您使用记录级操作，从而简化数据管理和数据管道开发。

- Amazon S3 Express One Zone 存储类与 Amazon EMR on EKS 集成：对于 7.2.0 及更高版本，可使用 Amazon EMR on EKS 将数据上传到 S3 Express One Zone。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。
- 支持 Spark 运算符的默认配置：现在，对于 7.2.0 及更高版本的 Amazon EMR on EKS 上的启动作业运行模型，Amazon EKS 上的 Spark 运算符支持与之相同的默认配置。这意味着 Amazon S3 和 EMRFS 等功能不再需要在 yaml 文件中手动配置。

emr-7.2.0-latest

发布说明：`emr-7.2.0-latest` 当前指向 `emr-7.2.0-20240610`。

区域：`emr-7.2.0-latest` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.2.0:latest`

emr-7.2.0-20240610

发布说明：`7.2.0-20240610` 已于 2023 年 12 月发布。这是 Amazon EMR 7.2.0（Spark）的初始发行版。

区域：`emr-7.2.0-20240610` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.2.0:20240610`

emr-7.2.0-flink-latest

发布说明：`emr-7.2.0-flink-latest` 当前指向 `emr-7.2.0-flink-20240610`。

区域：`emr-7.2.0-flink-latest` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.2.0-flink:latest`

emr-7.2.0-flink-20240610

发布说明：`7.2.0-flink-20240610` 已于 2023 年 12 月发布。这是 Amazon EMR 7.2.0（Flink）的初始发行版。

区域：`emr-7.2.0-flink-20240610` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.2.0-flink:20240610`

Amazon EMR on EKS 7.1.0 发行版

本页介绍了 Amazon EMR 的新增和更新的功能，这些功能特定于 Amazon EMR on EKS 部署。有关在亚马逊上运行的亚马逊 EMR EC2 以及亚马逊 EMR 7.1.0 版本的总体详情，请参阅《亚马逊 EMR 发布指南》中的亚马逊 EMR 7.1.0。

Amazon EMR on EKS 7.1 发行版

以下 Amazon EMR 7.1.0 发行版适用于 Amazon EMR on EKS。选择特定的 emr-7.1.0-XXXX 发行版以查看更多详细信息，例如相关的容器映像标签。

Flink releases

在运行 Flink 应用程序时，以下 Amazon EMR 7.1.0 发行版适用于 Amazon EMR on EKS。

- emr-7.1.0-flink-latest
- emr-7.1.0-flink-20240321

Spark releases

在运行 Spark 应用程序时，以下 Amazon EMR 7.1.0 发行版适用于 Amazon EMR on EKS。

- emr-7.1.0-latest
- emr-7.1.0-20240321
- emr-7.1.0-spark-rapids-latest
- emr-7.1.0-spark-rapids-20240321
- emr-7.1.0-java11-latest
- emr-7.1.0-java11-20240321
- emr-7.1.0-java8-latest
- emr-7.1.0-java8-20240321
- emr-7.1.0-spark-rapids-java8-latest

- emr-7.1.0-spark-rapids-java8-20240321
- notebook-spark/emr-7.1.0-latest
- notebook-spark/emr-7.1.0-20240321
- notebook-spark/emr-7.1.0-spark-rapids-latest
- notebook-spark/emr-7.1.0-spark-rapids-20240321
- notebook-spark/emr-7.1.0-java11-latest
- notebook-spark/emr-7.1.0-java11-20240321
- notebook-spark/emr-7.1.0-java8-latest
- notebook-spark/emr-7.1.0-java8-20240321
- notebook-spark/emr-7.1.0-spark-rapids-java8-latest
- notebook-spark/emr-7.1.0-spark-rapids-java8-20240321
- notebook-python/emr-7.1.0-latest
- notebook-python/emr-7.1.0-20240321
- notebook-python/emr-7.1.0-spark-rapids-latest
- notebook-python/emr-7.1.0-spark-rapids-20240321
- notebook-python/emr-7.1.0-java11-latest
- notebook-python/emr-7.1.0-java11-20240321
- notebook-python/emr-7.1.0-java8-latest
- notebook-python/emr-7.1.0-java8-20240321
- notebook-python/emr-7.1.0-spark-rapids-java8-latest
- notebook-python/emr-7.1.0-spark-rapids-java8-20240321
- livy/emr-7.1.0-latest
- livy/emr-7.1.0-20240321
- livy/emr-7.1.0-java11-latest
- livy/emr-7.1.0-java11-20240321
- livy/emr-7.1.0-java8-latest
- livy/emr-7.1.0-java8-20240321

发行说明

Amazon EMR on EKS 7.1.0 的发布说明

- 支持的应用程序 - 适用于 Java 的 Amazon SDK 2.23.18 and 1.12.656, Apache Spark 3.5.0-amzn-1, Apache Hudi 0.14.1-amzn-0, Apache Iceberg 1.4.3-amzn-0, Delta 3.0.0, Apache Spark RAPIDS 23.10.0-amzn-1, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.18.1-amzn-0, Flink Operator 1.6.1-amzn-1
- 支持的组件 - `aws-sagemaker-spark-sdk`、`emr-ddb`、`emr-goodies`、`emr-s3-select`、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。
- 支持的配置分类

与StartJobRun和一起使用 CreateManagedEndpoint APIs：

分类	描述
`core-site`	更改 `core-site.xml` Hadoop 文件中的值。
`emrfs-site`	更改 EMRFS 设置。
`spark-metrics`	更改 `metrics.properties` Spark 文件中的值。
`spark-defaults`	更改 `spark-defaults.conf` Spark 文件中的值。
`spark-env`	更改 Spark 环境中的值。
`spark-hive-site`	更改 `hive-site.xml` Spark 文件中的值。
`spark-log4j2`	更改 `log4j2.properties` Spark 文件中的值。
`emr-job-submitter`	任务提交者 Pod 的配置。

专门用于以下用途 CreateManagedEndpoint APIs：

分类	描述
jeg-config	更改 Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 文件中的值。
jupyter-kernel-overrides	更改 Jupyter 内核规范文件中内核映像的值。

配置分类允许您自定义应用程序。这些通常与应用程序的配置 XML 文件（例如 spark-hive-site.xml）相对应。有关更多信息，请参阅配置应用程序。

显著功能

Amazon EMR on EKS 7.1.0 发行版包含以下功能。

- Apache Livy 支持 Amazon EMR on EKS：对于 Amazon EMR on EKS 7.1.0 及更高版本，可在 Amazon EKS 集群上使用 Apache Livy 创建 Apache Livy REST 接口，来提交 Spark 作业或 Spark 代码片段。这样做可以同步和异步检索结果，同时利用 Amazon EMR on EKS 的优势，例如 Amazon EMR 优化的 Spark 运行时、启用 SSL 的 Livy 端点和编程设置体验。

emr-7.1.0-latest

发布说明：emr-7.1.0-latest 当前指向 emr-7.1.0-20240321。

区域：emr-7.1.0-latest 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：emr-7.1.0:latest

emr-7.1.0-20240321

发布说明：7.1.0-20240321 已于 2023 年 12 月发布。这是 Amazon EMR 7.1.0（Spark）的初始发行版。

区域：emr-7.1.0-20240321 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.1.0:20240321`

emr-7.1.0-flink-latest

发布说明：`emr-7.1.0-flink-latest` 当前指向 `emr-7.1.0-flink-20240321`。

区域：`emr-7.1.0-flink-latest` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 [Amazon EMR on EKS 服务终端节点]。

容器镜像标签：`emr-7.1.0-flink:latest`

emr-7.1.0-flink-20240321

发布说明：`7.1.0-flink-20240321` 已于 2023 年 12 月发布。这是 Amazon EMR 7.1.0（Flink）的初始发行版。

区域：`emr-7.1.0-flink-20240321` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 [Amazon EMR on EKS 服务终端节点]。

容器镜像标签：`emr-7.1.0-flink:20240321`

EKS 上的 Amazon EMR 7.0.0 发行版

本页介绍了 Amazon EMR 的新增和更新的功能，这些功能特定于 Amazon EMR on EKS 部署。有关在亚马逊上运行的亚马逊 EMR EC2 以及亚马逊 EMR 7.0.0 版本的总体详情，请参阅亚马逊 EMR 发布指南中的亚马逊 EM R 7.0. 0。

EKS 上的 Amazon EMR 7.0 发行版

以下 Amazon EMR 7.0.0 发行版适用于 EKS 上的 Amazon EMR。选择特定的 emr-7.0.0-XXXX 发行版以查看更多详细信息，例如相关的容器映像标签。

Flink releases

在您运行 Flink 应用程序时，以下 Amazon EMR 7.0.0 发行版适用于 EKS 上的 Amazon EMR。

- emr-7.0.0-flink-latest
- emr-7.0.0-flink-2024321

- emr-7.0.0-flink-20231211

Spark releases

在您运行 Spark 应用程序时，以下 Amazon EMR 7.0.0 发行版适用于 EKS 上的 Amazon EMR。

- emr-7.0.0-latest
- emr-7.0.0-20231211
- emr-7.0.0-spark-rapids-latest
- emr-7.0.0-spark-rapids-20231211
- emr-7.0.0-java11-latest
- emr-7.0.0-java11-20231211
- emr-7.0.0-java8-latest
- emr-7.0.0-java8-20231211
- emr-7.0.0-spark-rapids-java8-latest
- emr-7.0.0-spark-rapids-java8-20231211
- notebook-spark/emr-7.0.0-latest
- notebook-spark/emr-7.0.0-20231211
- notebook-spark/emr-7.0.0-spark-rapids-latest
- notebook-spark/emr-7.0.0-spark-rapids-20231211
- notebook-spark/emr-7.0.0-java11-latest
- notebook-spark/emr-7.0.0-java11-20231211
- notebook-spark/emr-7.0.0-java8-latest
- notebook-spark/emr-7.0.0-java8-20231211
- notebook-spark/emr-7.0.0-spark-rapids-java8-latest
- notebook-spark/emr-7.0.0-spark-rapids-java8-20231211
- notebook-python/emr-7.0.0-latest
- notebook-python/emr-7.0.0-20231211
- notebook-python/emr-7.0.0-spark-rapids-latest
- notebook-python/emr-7.0.0-spark-rapids-20231211

- notebook-python/emr-7.0.0-java11-latest
- notebook-python/emr-7.0.0-java11-20231211
- notebook-python/emr-7.0.0-java8-latest
- notebook-python/emr-7.0.0-java8-20231211
- notebook-python/emr-7.0.0-spark-rapids-java8-latest
- notebook-python/emr-7.0.0-spark-rapids-java8-20231211

发行说明

EKS 上的 Amazon EMR 7.0.0 的发布说明

- 支持的应用程序 - 适用于 Java 的 Amazon SDK 2.20.160-amzn-0 and 1.12.595, Apache Spark 3.5.0-amzn-0, Apache Flink 1.18.0-amzn-0, Flink Operator 1.6.1, Apache Hudi 0.14.0-amzn-1, Apache Iceberg 1.4.2-amzn-0, Delta 3.0.0, Apache Spark RAPIDS 23.10.0-amzn-0, Jupyter Enterprise Gateway 2.6.0
- 支持的组件 - `aws-sagemaker-spark-sdk`、`emr-ddb`、`emr-goodies`、`emr-s3-select`、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。
- 支持的配置分类

  与StartJobRun和一起使用 CreateManagedEndpoint APIs：

分类	描述
`core-site`	更改 `core-site.xml` Hadoop 文件中的值。
`emrfs-site`	更改 EMRFS 设置。
`spark-metrics`	更改 `metrics.properties` Spark 文件中的值。
`spark-defaults`	更改 `spark-defaults.conf` Spark 文件中的值。
`spark-env`	更改 Spark 环境中的值。
`spark-hive-site`	更改 `hive-site.xml` Spark 文件中的值。

分类	描述
spark-log4j	更改 log4j2.properties Spark 文件中的值。
emr-job-submitter	任务提交者 Pod 的配置。

专门用于以下用途 CreateManagedEndpoint APIs：

分类	描述
jeg-config	更改 Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 文件中的值。
jupyter-kernel-overrides	更改 Jupyter 内核规范文件中内核映像的值。

配置分类允许您自定义应用程序。这些通常与应用程序的配置 XML 文件（例如 spark-hive-site.xml）相对应。有关更多信息，请参阅配置应用程序。

显著功能

EKS 上的 Amazon EMR 发行版 7.0 中包含以下功能。

- 应用程序升级 – EKS 上的 Amazon EMR 7.0.0 应用程序升级包括 Spark 3.5、Flink 1.18 和 Flink Operator 1.6.1。
- Flink Autoscaler 参数自动调整 – Flink Autoscaler 用于扩展计算的默认参数可能不是给定作业的最佳值。EKS 上的 Amazon EMR 7.0.0 使用捕获的特定指标的历史趋势来计算为作业量身定制的最佳参数。

更改

EKS 上的 Amazon EMR 发行版 7.0 中包含以下更改。

- Amazon Linux 2023 – 使用 EKS 上的 Amazon EMR 7.0.0 及更高版本，所有容器映像都基于 Amazon Linux 2023。

- Spark 使用 Java 17 作为默认运行时系统 – EKS 上的 Amazon EMR 7.0.0 Spark 使用 Java 17 作为默认运行时系统。如果需要，可以切换到使用具有 EKS 上的 Amazon EMR 7.0 发行版 列表中提供的相应版本标签的 Java 8 或 Java 11。

emr-7.0.0-latest

发布说明：`emr-7.0.0-latest` 当前指向 `emr-7.0.0-2024321`。

区域：`emr-7.0.0-latest` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.0.0:latest`

emr-7.0.0-2024321

发布说明：`7.0.0-2024321` 已于 2024 年 3 月 11 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：`emr-7.0.0-2024321` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.0.0:2024321`

emr-7.0.0-20231211

发布说明：`7.0.0-20231211` 已于 2023 年 12 月发布。这是 Amazon EMR 7.0.0（Spark）的初始发行版。

区域：`emr-7.0.0-20231211` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.0.0:20231211`

emr-7.0.0-flink-latest

发布说明：`emr-7.0.0-flink-latest` 当前指向 `emr-7.0.0-flink-2024321`。

区域：`emr-7.0.0-flink-latest` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.0.0-flink:latest`

emr-7.0.0-flink-2024321

发布说明：`7.0.0-flink-2024321` 已于 2024 年 3 月 11 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：`emr-7.0.0-flink-2024321` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.0.0-flink:2024321`

emr-7.0.0-flink-20231211

发布说明：`7.0.0-flink-20231211` 已于 2023 年 12 月发布。这是 Amazon EMR 7.0.0（Flink）的初始发行版。

区域：`emr-7.0.0-flink-20231211` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-7.0.0-flink:20231211`

EKS 上的 Amazon EMR 6.15.0 发行版

本页介绍了 Amazon EMR 的新增和更新的功能，这些功能特定于 Amazon EMR on EKS 部署。有关在亚马逊上运行的亚马逊 EMR EC2 以及亚马逊 EMR 6.15.0 版本的总体详情，请参阅《亚马逊 EMR 发布指南》中的 Amazon EM R 6.15. 0。

EKS 上的 Amazon EMR 6.15 发行版

以下 Amazon EMR 6.15.0 发行版适用于 EKS 上的 Amazon EMR。选择特定的 emr-6.15.0-XXXX 发行版以查看更多详细信息，例如相关的容器映像标签。

Flink releases

在您运行 Flink 应用程序时，以下 Amazon EMR 6.15.0 发行版适用于 EKS 上的 Amazon EMR。

- emr-6.15.0-flink-latest
- emr-6.15.0-flink-20240105

- emr-6.15.0-flink-20231109

Spark releases

在您运行 Spark 应用程序时，以下 Amazon EMR 6.15.0 发行版适用于 EKS 上的 Amazon EMR。

- emr-6.15.0-latest
- emr-6.15.0-20231109
- emr-6.15.0-spark-rapids-latest
- emr-6.15.0-spark-rapids-20231109
- emr-6.15.0-java11-latest
- emr-6.15.0-java11-20231109
- emr-6.15.0-java17-latest
- emr-6.15.0-java17-20231109
- emr-6.15.0-java17-al2023-latest
- emr-6.15.0-java17-al2023-20231109
- emr-6.15.0-spark-rapids-java17-latest
- emr-6.15.0-spark-rapids-java17-20231109
- emr-6.15.0-spark-rapids-java17-al2023-latest
- emr-6.15.0-spark-rapids-java17-al2023-20231109
- notebook-spark/emr-6.15.0-latest
- notebook-spark/emr-6.15.0-20231109
- notebook-spark/emr-6.15.0-spark-rapids-latest
- notebook-spark/emr-6.15.0-spark-rapids-20231109
- notebook-spark/emr-6.15.0-java11-latest
- notebook-spark/emr-6.15.0-java11-20231109
- notebook-spark/emr-6.15.0-java17-latest
- notebook-spark/emr-6.15.0-java17-20231109
- notebook-spark/emr-6.15.0-java17-al2023-latest
- notebook-spark/emr-6.15.0-java17-al2023-20231109
- notebook-python/emr-6.15.0-latest
- notebook-python/emr-6.15.0-20231109

- notebook-python/emr-6.15.0-spark-rapids-latest
- notebook-python/emr-6.15.0-spark-rapids-20231109
- notebook-python/emr-6.15.0-java11-latest
- notebook-python/emr-6.15.0-java11-20231109
- notebook-python/emr-6.15.0-java17-latest
- notebook-python/emr-6.15.0-java17-20231109
- notebook-python/emr-6.15.0-java17-al2023-latest
- notebook-python/emr-6.15.0-java17-al2023-20231109

发行说明

EKS 上的 Amazon EMR 6.15.0 发布说明

- 支持的应用程序 - 适用于 Java 的 Amazon SDK 1.12.569, Apache Spark 3.4.1-amzn-2, Apache Flink 1.17.1-amzn-1, Apache Hudi 0.14.0-amzn-0, Apache Iceberg 1.4.0-amzn-0, Delta 2.4.0, Apache Spark RAPIDS 23.08.01-amzn-0, Jupyter Enterprise Gateway 2.6.0
- 支持的组件 - `aws-sagemaker-spark-sdk`、`emr-ddb`、`emr-goodies`、`emr-s3-select`、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。
- 支持的配置分类

  与StartJobRun和一起使用 CreateManagedEndpoint APIs：

分类	描述
`core-site`	更改 `core-site.xml` Hadoop 文件中的值。
`emrfs-site`	更改 EMRFS 设置。
`spark-metrics`	更改 `metrics.properties` Spark 文件中的值。
`spark-defaults`	更改 `spark-defaults.conf` Spark 文件中的值。
`spark-env`	更改 Spark 环境中的值。

分类	描述
spark-hive-site	更改 hive-site.xml Spark 文件中的值。
spark-log4j	更改 log4j2.properties Spark 文件中的值。
emr-job-submitter	任务提交者 Pod 的配置。

专门用于以下用途 CreateManagedEndpoint APIs：

分类	描述
jeg-config	更改 Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 文件中的值。
jupyter-kernel-overrides	更改 Jupyter 内核规范文件中内核映像的值。

配置分类允许您自定义应用程序。这些通常与应用程序的配置 XML 文件（例如 spark-hive-site.xml）相对应。有关更多信息，请参阅配置应用程序。

显著功能

EKS 上的 Amazon EMR 发行版 6.15 中包含以下功能。

- 带有 Apache Flink 的 EKS 上的 Amazon EMR - 使用 EKS 上的 Amazon EMR 6.15.0，您可以运行基于 Apache Flink 的应用程序以及相同 Amazon EKS 集群上的其他类型的应用程序。这有助于提高资源利用率并简化基础架构管理。您可以在 Flink 应用程序中使用竞价型实例，并可以正常停用，并且可通过 Amazon EBS 实现精细恢复和任务本地恢复，从而缩短重启时间。可访问性和监控功能包括能够使用存储在 Amazon S3 中的 jar 启动 Flink 应用程序、访问 G Amazon lue 数据目录、监控与 Amazon S3 和亚马逊的集成以及容器日志轮换。 CloudWatch

emr-6.15.0-latest

发布说明：emr-6.15.0-latest 当前指向 emr-6.15.0-20240105。

区域：`emr-6.15.0-latest` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 [Amazon EMR on EKS 服务终端节点]。

容器镜像标签：`emr-6.15.0:latest`

emr-6.15.0-20240105

发布说明：`6.15.0-20240105` 已于 2024 年 1 月 17 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：`emr-6.15.0-20240105` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 [Amazon EMR on EKS 服务终端节点]。

容器镜像标签：`emr-6.15.0:20240105`

emr-6.15.0-20231109

发布说明：`6.15.0-20231109` 已于 2023 年 11 月 17 日发布。这是 Amazon EMR 6.15.0 的初始发行版。

区域：`emr-6.15.0-20231109` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 [Amazon EMR on EKS 服务终端节点]。

容器镜像标签：`emr-6.15.0:20231109`

emr-6.15.0-flink-latest

发布说明：`emr-6.15.0-flink-latest` 当前指向 `emr-6.15.0-flink-20240105`。

区域：`emr-6.15.0-flink-latest` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 [Amazon EMR on EKS 服务终端节点]。

容器镜像标签：`emr-6.15.0-flink:latest`

emr-6.15.0-flink-20240105

发布说明：`6.15.0-flink-20240105` 已于 2024 年 1 月 17 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：`emr-6.15.0-flink-20240105` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 [Amazon EMR on EKS 服务终端节点]。

容器镜像标签：`emr-6.15.0-flink:20240105`

emr-6.15.0-flink-20231109

发布说明：`6.15.0-flink-20231109` 已于 2023 年 11 月 17 日发布。这是 Amazon EMR 6.15.0 的初始发行版。

区域：`emr-6.15.0-flink-20231109` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.15.0-flink:20231109`

Amazon EMR on EKS 6.14.0 发行版

本页介绍了 Amazon EMR 的新增和更新的功能，这些功能特定于 Amazon EMR on EKS 部署。有关在亚马逊上运行的亚马逊 EMR EC2 以及亚马逊 EMR 6.14.0 版本的总体详情，请参阅亚马逊 EMR 发布指南中的亚马逊 EM R 6.14. 0。

Amazon EMR on EKS 6.14 发行版

以下 Amazon EMR 6.14.0 发行版适用于 Amazon EMR on EKS。选择特定的 emr-6.14.0-XXXX 发行版以查看更多详细信息，例如相关的容器映像标签。

- emr-6.14.0-latest
- emr-6.14.0-20231005
- emr-6.14.0-spark-rapids-latest
- emr-6.14.0-spark-rapids-20231005
- emr-6.14.0-java11-latest
- emr-6.14.0-java11-20231005
- emr-6.14.0-java17-latest
- emr-6.14.0-java17-20231005
- emr-6.14.0-java17-al2023-latest
- emr-6.14.0-java17-al2023-20231005
- emr-6.14.0-spark-rapids-java17-latest
- emr-6.14.0-spark-rapids-java17-20231005
- emr-6.14.0-spark-rapids-java17-al2023-latest
- emr-6.14.0-spark-rapids-java17-al2023-20231005

- notebook-spark/emr-6.14.0-latest
- notebook-spark/emr-6.14.0-20231005
- notebook-spark/emr-6.14.0-spark-rapids-latest
- notebook-spark/emr-6.14.0-spark-rapids-20231005
- notebook-spark/emr-6.14.0-java11-latest
- notebook-spark/emr-6.14.0-java11-20231005
- notebook-spark/emr-6.14.0-java17-latest
- notebook-spark/emr-6.14.0-java17-20231005
- notebook-spark/emr-6.14.0-java17-al2023-latest
- notebook-spark/emr-6.14.0-java17-al2023-20231005
- notebook-python/emr-6.14.0-latest
- notebook-python/emr-6.14.0-20231005
- notebook-python/emr-6.14.0-spark-rapids-latest
- notebook-python/emr-6.14.0-spark-rapids-20231005
- notebook-python/emr-6.14.0-java11-latest
- notebook-python/emr-6.14.0-java11-20231005
- notebook-python/emr-6.14.0-java17-latest
- notebook-python/emr-6.14.0-java17-20231005
- notebook-python/emr-6.14.0-java17-al2023-latest
- notebook-python/emr-6.14.0-java17-al2023-20231005

发行说明

Amazon EMR on EKS 6.14.0 发布说明

- 支持的应用程序- 适用于 Java 的 Amazon SDK 1.12.543、Apache Spark 3.4.1-amzn-1、Apache Hudi 0.13.1-amzn-2、Apache Iceberg 1.3.0-amzn-0、Delta 2.4.0、Apache Spark RAPIDS 23.06.0-amzn-2、Jupyter Enterprise Gateway 2.7.0
- 支持的组件 - `aws-sagemaker-spark-sdk`、`emr-ddb`、`emr-goodies`、`emr-s3-select`、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。
- 支持的配置分类

  与StartJobRun和一起使用 CreateManagedEndpoint APIs：

分类	描述
`core-site`	更改 `core-site.xml` Hadoop 文件中的值。
`emrfs-site`	更改 EMRFS 设置。
`spark-metrics`	更改 `metrics.properties` Spark 文件中的值。
`spark-defaults`	更改 `spark-defaults.conf` Spark 文件中的值。
`spark-env`	更改 Spark 环境中的值。
`spark-hive-site`	更改 `hive-site.xml` Spark 文件中的值。
`spark-log4j`	更改 `log4j2.properties` Spark 文件中的值。
`emr-job-submitter`	任务提交者 Pod 的配置。

专门用于以下用途 CreateManagedEndpoint APIs：

分类	描述
`jeg-config`	更改 Jupyter Enterprise Gateway `jupyter_enterprise_gateway_config.py` 文件中的值。
`jupyter-kernel-overrides`	更改 Jupyter 内核规范文件中内核映像的值。

配置分类允许您自定义应用程序。这些通常与应用程序的配置 XML 文件（例如 `spark-hive-site.xml`）相对应。有关更多信息，请参阅配置应用程序。

显著功能

Amazon EMR on EKS 发行版 6.14 中包含以下功能。

- Apache Livy 支持 – Amazon EMR on EKS 现在支持将 Apache Livy 与 `spark-submit` 结合使用。

emr-6.14.0-latest

发布说明：`emr-6.14.0-latest` 当前指向 `emr-6.14.0-20231005`。

区域：`emr-6.14.0-latest` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.14.0:latest`

emr-6.14.0-20231005

发布说明：`6.14.0-20231005` 已于 2023 年 10 月 17 日发布。这是 Amazon EMR 6.14.0 的初始发行版。

区域：`emr-6.14.0-20231005` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.14.0:20231005`

Amazon EMR on EKS 6.13.0 版本

本页介绍了 Amazon EMR 的新增和更新的功能，这些功能特定于 Amazon EMR on EKS 部署。有关在亚马逊上运行的亚马逊 EMR EC2 以及亚马逊 EMR 6.13.0 版本的总体详情，请参阅亚马逊 EMR 发布指南中的亚马逊 EM R 6.13. 0。

Amazon EMR on EKS 6.13 版本

以下 Amazon EMR 6.13.0 版本适用于 Amazon EMR on EKS。选择特定的 emr-6.13.0-XXXX 版本以查看更多详细信息，例如相关的容器映像标签。

- emr-6.13.0-latest
- emr-6.13.0-20230814

- emr-6.13.0-spark-rapids-latest
- emr-6.13.0-spark-rapids-20230814
- emr-6.13.0-java11-latest
- emr-6.13.0-java11-20230814
- emr-6.13.0-java17-latest
- emr-6.13.0-java17-20230814
- emr-6.13.0-java17-al2023-latest
- emr-6.13.0-java17-al2023-20230814
- emr-6.13.0-spark-rapids-java17-latest
- emr-6.13.0-spark-rapids-java17-20230814
- emr-6.13.0-spark-rapids-java17-al2023-latest
- emr-6.13.0-spark-rapids-java17-al2023-20230814
- notebook-spark/emr-6.13.0-latest
- notebook-spark/emr-6.13.0-20230814
- notebook-spark/emr-6.13.0-spark-rapids-latest
- notebook-spark/emr-6.13.0-spark-rapids-20230814
- notebook-spark/emr-6.13.0-java11-latest
- notebook-spark/emr-6.13.0-java11-20230814
- notebook-spark/emr-6.13.0-java17-latest
- notebook-spark/emr-6.13.0-java17-20230814
- notebook-spark/emr-6.13.0-java17-al2023-latest
- notebook-spark/emr-6.13.0-java17-al2023-20230814
- notebook-python/emr-6.13.0-latest
- notebook-python/emr-6.13.0-20230814
- notebook-python/emr-6.13.0-spark-rapids-latest
- notebook-python/emr-6.13.0-spark-rapids-20230814
- notebook-python/emr-6.13.0-java11-latest
- notebook-python/emr-6.13.0-java11-20230814
- notebook-python/emr-6.13.0-java17-latest

- notebook-python/emr-6.13.0-java17-20230814
- notebook-python/emr-6.13.0-java17-al2023-latest
- notebook-python/emr-6.13.0-java17-al2023-20230814

发行说明

Amazon EMR on EKS 6.13.0 的发布说明

- 支持的应用程序- 适用于 Java 的 Amazon SDK 1.12.513、Apache Spark 3.4.1-amzn-0、Apache Hudi 0.13.1-amzn-0、Apache Iceberg 1.3.0-amzn-0、Delta 2.4.0、Apache Spark RAPIDS 23.06.0-amzn-1、Jupyter Enterprise Gateway 2.6.0.amzn
- 支持的组件 - `aws-sagemaker-spark-sdk`、`emr-ddb`、`emr-goodies`、`emr-s3-select`、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。
- 支持的配置分类

  与StartJobRun和一起使用 CreateManagedEndpoint APIs：

分类	描述
`core-site`	更改 `core-site.xml` Hadoop 文件中的值。
`emrfs-site`	更改 EMRFS 设置。
`spark-metrics`	更改 `metrics.properties` Spark 文件中的值。
`spark-defaults`	更改 `spark-defaults.conf` Spark 文件中的值。
`spark-env`	更改 Spark 环境中的值。
`spark-hive-site`	更改 `hive-site.xml` Spark 文件中的值。
`spark-log4j`	更改 `log4j2.properties` Spark 文件中的值。
`emr-job-submitter`	任务提交者 Pod 的配置。

专门用于以下用途 CreateManagedEndpoint APIs：

分类	描述
jeg-config	更改 Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 文件中的值。
jupyter-kernel-overrides	更改 Jupyter 内核规范文件中内核映像的值。

配置分类允许您自定义应用程序。这些通常与应用程序的配置 XML 文件（例如 spark-hive-site.xml）相对应。有关更多信息，请参阅配置应用程序。

显著功能

Amazon EMR on EKS 的 6.13 版中包含以下功能。

- 亚马逊 Linux 202 3-在 EKS 6.13 及更高版本上使用亚马逊 EMR，你可以使用 AL2 023 作为操作系统和 Java 17 运行时一起启动 Spark。为此，请使用名称中带有 al2023 的发行版标签。例如：emr-6.13.0-java17-al2023-latest。我们建议您在将生产工作负载迁移到 AL2 023 和 Java 17 之前，先验证并运行性能测试。
- 带 Apache Flink 的 Amazon EMR on EKS（公开预览版）– Amazon EMR on EKS 版本 6.13 及更高版本支持 Apache Flink，现已提供公开预览版。发布后，您可以在同一 Amazon EKS 集群上运行基于 Apache Flink 的应用程序以及其他类型的应用程序。这有助于提高资源利用率并简化基础架构管理。如果您已在 Amazon EKS 上运行大数据框架，您现在可以让 Amazon EMR 进行自动预置和管理。

emr-6.13.0-latest

发布说明：emr-6.13.0-latest 当前指向 emr-6.13.0-20230814。

区域：emr-6.13.0-latest 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：emr-6.13.0:latest

emr-6.13.0-20230814

发布说明：`6.13.0-20230814` 已于 2023 年 9 月 7 日发布。这是 Amazon EMR 6.13.0 的初始版本。

区域：`emr-6.13.0-20230814` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.13.0:20230814`

Amazon EMR on EKS 6.12.0 版本

本页介绍了 Amazon EMR 的新增和更新的功能，这些功能特定于 Amazon EMR on EKS 部署。有关在亚马逊上运行的亚马逊 EMR EC2 以及亚马逊 EMR 6.12.0 版本的总体详情，请参阅亚马逊 EMR 发布指南中的亚马逊 EM R 6.12. 0。

Amazon EMR on EKS 6.12 版本

以下 Amazon EMR 6.12.0 版本适用于 Amazon EMR on EKS。选择特定的 emr-6.12.0-XXXX 版本以查看更多详细信息，例如相关的容器映像标签。

- emr-6.12.0-latest
- emr-6.12.0-20240321
- emr-6.12.0-20230701
- emr-6.12.0-spark-rapids-latest
- emr-6.12.0-spark-rapids-20230701
- emr-6.12.0-java11-latest
- emr-6.12.0-java11-20230701
- emr-6.12.0-java17-latest
- emr-6.12.0-java17-20230701
- emr-6.12.0-17-lest spark-rapids-java
- emr-6.12.0-17-20230701 spark-rapids-java
- notebook-spark/emr-6.12.0-latest
- notebook-spark/emr-6.12.0-20230701

- notebook-spark/emr-6.12.0-spark-rapids-latest
- notebook-spark/emr-6.12.0-spark-rapids-20230701
- notebook-python/emr-6.12.0-latest
- notebook-python/emr-6.12.0-20230701
- notebook-python/emr-6.12.0-spark-rapids-latest
- notebook-python/emr-6.12.0-spark-rapids-20230701

发行说明

Amazon EMR on EKS 6.12.0 的发布说明

- 支持的应用程序- 适用于 Java 的 Amazon SDK 1.12.490、Apache Spark 3.4.0-amzn-0、Apache Hudi 0.13.1-amzn-0、Apache Iceberg 1.3.0-amzn-0、Delta 2.4.0、Apache Spark RAPIDS 23.06.0-amzn-0、Jupyter 企业网关 2.6.0
- 支持的组件 - `aws-sagemaker-spark-sdk`、`emr-ddb`、`emr-goodies`、`emr-s3-select`、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。
- 支持的配置分类

  与StartJobRun和一起使用 CreateManagedEndpoint APIs：

分类	描述
`core-site`	更改 `core-site.xml` Hadoop 文件中的值。
`emrfs-site`	更改 EMRFS 设置。
`spark-metrics`	更改 `metrics.properties` Spark 文件中的值。
`spark-defaults`	更改 `spark-defaults.conf` Spark 文件中的值。
`spark-env`	更改 Spark 环境中的值。
`spark-hive-site`	更改 `hive-site.xml` Spark 文件中的值。

分类	描述
spark-log4j	更改 log4j2.properties Spark 文件中的值。
emr-job-submitter	任务提交者 Pod 的配置。

专门用于以下用途 CreateManagedEndpoint APIs：

分类	描述
jeg-config	更改 Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 文件中的值。
jupyter-kernel-overrides	更改 Jupyter 内核规范文件中内核映像的值。

配置分类允许您自定义应用程序。这些通常与应用程序的配置 XML 文件（例如 spark-hive-site.xml）相对应。有关更多信息，请参阅配置应用程序。

显著功能

Amazon EMR on EKS 的 6.12 版中包含以下功能。

- Java 17 – 借助 Amazon EMR on EKS 6.12 及更高版本，您可以使用 Java 17 运行时系统启动 Spark。为此，将 emr-6.12.0-java17-latest 作为发行版标签传递。我们建议您在将生产工作负载从更早版本的 Java 映像迁移到 Java 17 映像之前，先验证并运行性能测试。

emr-6.12.0-latest

发布说明：emr-6.12.0-latest 当前指向 emr-6.12.0-20240321。

区域：emr-6.12.0-latest 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：emr-6.12.0:latest

emr-6.12.0-20240321

发布说明：`6.12.0-20240321` 已于 2024 年 3 月 11 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：`emr-6.12.0-20240321` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.12.0:20240321`

emr-6.12.0-20230701

发布说明：`6.12.0-20230701` 已于 2023 年 7 月 1 日发布。这是 Amazon EMR 6.12.0 的初始版本。

区域：`emr-6.12.0-20230701` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.12.0:20230701`

Amazon EMR on EKS 6.11.0 版本

本页介绍了 Amazon EMR 的新增和更新的功能，这些功能特定于 Amazon EMR on EKS 部署。有关在亚马逊上运行的亚马逊 EMR EC2 以及亚马逊 EMR 6.11.0 版本的总体详情，请参阅亚马逊 EMR 发布指南中的亚马逊 EM R 6.11. 0。

Amazon EMR on EKS 6.11 版本

以下 Amazon EMR 6.11.0 版本适用于 Amazon EMR on EKS。选择特定的 emr-6.11.0-XXXX 版本以查看更多详细信息，例如相关的容器映像标签。

- emr-6.11.0-latest
- emr-6.11.0-20230905
- emr-6.11.0-20230509

- emr-6.11.0-spark-rapids-latest
- emr-6.11.0-spark-rapids-20230509
- emr-6.11.0-java11-latest
- emr-6.11.0-java11-20230509

- notebook-spark/emr-6.11.0-latest
- notebook-spark/emr-6.11.0-20230509
- notebook-python/emr-6.11.0-latest
- notebook-python/emr-6.11.0-20230509

发行说明

Amazon EMR on EKS 6.11.0 的发布说明

- 支持的应用程序- 适用于 Java 的 Amazon SDK 1.12.446、Apache Spark 3.3.2-amzn-0、Apache Hudi 0.13.0-amzn-0、Apache Iceberg 1.2.0-amzn-0、Delta 2.0、Apache Spark RAPIDS 23.02.0-amzn-0、Jupyter Enterprise Gateway 2.6.0
- 支持的组件 - `aws-sagemaker-spark-sdk`、`emr-ddb`、`emr-goodies`、`emr-s3-select`、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。
- 支持的配置分类

  与StartJobRun和一起使用 CreateManagedEndpoint APIs：

分类	描述
`core-site`	更改 `core-site.xml` Hadoop 文件中的值。
`emrfs-site`	更改 EMRFS 设置。
`spark-metrics`	更改 `metrics.properties` Spark 文件中的值。
`spark-defaults`	更改 `spark-defaults.conf` Spark 文件中的值。
`spark-env`	更改 Spark 环境中的值。
`spark-hive-site`	更改 `hive-site.xml` Spark 文件中的值。
`spark-log4j`	更改 `log4j.properties` Spark 文件中的值。

专门用于以下用途 CreateManagedEndpoint APIs：

分类	描述
`jeg-config`	更改 Jupyter Enterprise Gateway `jupyter_enterprise_gateway_config.py` 文件中的值。
`jupyter-kernel-overrides`	更改 Jupyter 内核规范文件中内核映像的值。

配置分类允许您自定义应用程序。这些通常与应用程序的配置 XML 文件（例如 `spark-hive-site.xml`）相对应。有关更多信息，请参阅配置应用程序。

显著功能

Amazon EMR on EKS 的 6.11 版中包含以下功能。

- Amazon ECR Public Gallery 中的 Amazon EMR on EKS 基础映像 – 如果您使用自定义映像功能，我们的基础映像将提供与 Amazon EMR on EKS 交互所需的必要 jar、配置和库。现在，您可以在 Amazon ECR Public Gallery 中找到基础映像。
- Spark 容器日志轮换 – Amazon EMR on EKS 6.11 支持 Spark 容器日志轮换。您可以在 `StartJobRun` API 的 `MonitoringConfiguration` 操作中使用 `containerLogRotationConfiguration` 启用该功能。您可以配置 `rotationSize` 和 `maxFilestoKeep` 来指定您希望 Amazon EMR on EKS 在 Spark 驱动程序和执行器 Pod 中保留的日志文件的数量和大小。有关更多信息，请参阅 使用 Spark 容器日志轮换。
- Spark 运算符和 Spark-submit 中的 Volcano 支持 – Amazon EMR on EKS 6.11 支持使用 Volcano 作为 Spark 运算符和 spark-submit 中的 Kubernetes 自定义调度程序来运行 Spark 任务。您可以使用分组调度、队列管理、抢占和公平分享调度等功能来实现高调度吞吐量和优化容量。有关更多信息，请参阅 在 Amazon EMR on EKS 上将 Volcano 用作 Apache Spark 自定义调度器。

emr-6.11.0-latest

发布说明：`emr-6.11.0-latest` 当前指向 `emr-20230905`。

区域：`emr-6.11.0-latest` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.11.0:latest`

emr-6.11.0-20230905

发布说明：`6.11.0-20230905` 已于 2023 年 9 月 29 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：`emr-6.11.0-20230509` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.11.0:20230509`

emr-6.11.0-20230509

发布说明：`6.11.0-20230509` 已于 2023 年 5 月 9 日发布。这是 Amazon EMR 6.11.0 的初始版本。

区域：`emr-6.11.0-20230509` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.11.0:20230509`

Amazon EMR on EKS 6.10.0 版本

以下 Amazon EMR 6.10.0 版本适用于 Amazon EMR on EKS。选择特定的 emr-6.10.0-XXXX 版本以查看更多详细信息，例如相关的容器映像标签。

- emr-6.10.0-latest
- emr-6.10.0-20230905
- emr-6.10.0-20230624
- emr-6.10.0-20230421
- emr-6.10.0-20230403
- emr-6.10.0-20230220
- emr-6.10.0-spark-rapids-latest
- emr-6.10.0-spark-rapids-20230624
- emr-6.10.0-spark-rapids-20230220

- emr-6.10.0-java11-latest
- emr-6.10.0-java11-20230624
- emr-6.10.0-java11-20230220
- notebook-spark/emr-6.10.0-latest
- notebook-spark/emr-6.10.0-20230624
- notebook-spark/emr-6.10.0-20230220
- notebook-python/emr-6.10.0-latest
- notebook-python/emr-6.10.0-20230624
- notebook-python/emr-6.10.0-20230220

Amazon EMR 6.10.0 的发布说明

- 支持的应用程序- 适用于 Java 的 Amazon SDK 1.12.397、Spark 3.3.1-amzn-0、Hudi 0.12.2-amzn-0、Iceberg 1.1.0-amzn-0、Delta 2.0。
- 支持的组件：`aws-sagemaker-spark-sdk`、`emr-ddb`、`emr-goodies`、`emr-s3-select`、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。
- 受支持的配置分类：

  与StartJobRun和一起使用 CreateManagedEndpoint APIs：

分类	描述
`core-site`	更改 Hadoop `core-site.xml` 文件中的值。
`emrfs-site`	更改 EMRFS 设置。
`spark-metrics`	更改 Spark `metrics.properties` 文件中的值。
`spark-defaults`	更改 Spark `spark-defaults.conf` 文件中的值。
`spark-env`	更改 Spark 环境中的值。
`spark-hive-site`	更改 Spark `hive-site.xml` 文件中的值。

分类	描述
spark-log4j	更改 Spark log4j.properties 文件中的值。

专门用于以下用途 CreateManagedEndpoint APIs：

分类	描述
jeg-config	更改 Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 文件中的值。
jupyter-kernel-overrides	更改 Jupyter 内核规范文件中内核映像的值。

配置分类允许您自定义应用程序。这些通常与应用程序的配置 XML 文件（例如 spark-hive-site.xml）相对应。有关更多信息，请参阅配置应用程序。

显著功能

- Spark 运算符 – 借助 Amazon EMR on EKS 6.10.0 及更高版本，您可以使用 Apache Spark 的 Kubernetes 运算符或 Spark 运算符，使用您自己的 Amazon EKS 集群上的 Amazon EMR 发行版运行时系统来部署和管理 Spark 应用程序。有关更多信息，请参阅 使用 Spark Operator 运行 Spark 任务。
- Java 11 – 借助 Amazon EMR on EKS 6.10 及更高版本，您可以使用 Java 11 运行时系统启动 Spark。为此，将 emr-6.10.0-java11-latest 作为发行版标签传递。我们建议您在将生产工作负载从 Java 8 映像迁移到 Java 11 映像之前，先验证并运行性能测试。
- 对于 Apache Spark 的 Amazon Redshift 集成，Amazon EMR on EKS 6.10.0 消除了对 minimal-json.jar 的依赖关系，并自动将所需的 spark-redshift 相关 jar 添加到 Spark 的执行器类路径：spark-redshift.jar、spark-avro.jar 和 RedshiftJDBC.jar。

更改

- 现在默认为 Parquet、ORC 和基于文本的格式（包括 CSV 和 JSON）启用 EMRFS S3 优化的提交程序。

emr-6.10.0-latest

发布说明：`emr-6.10.0-latest` 当前指向 `emr-6.10.0-20230905`。

区域：`emr-6.10.0-latest` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.10.0:latest`

emr-6.10.0-20230905

发布说明：`6.10.0-20230905` 已于 2023 年 9 月 29 日发布。与上一版本相比，此版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：`emr-6.10.0-20230905` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.10.0:20230905`

emr-6.10.0-20230624

发布说明：`6.10.0-20230624` 已于 2023 年 7 月 7 日发布。与上一版本相比，此版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：`emr-6.10.0-20230624` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.10.0:20230624`

emr-6.10.0-20230421

发布说明：`6.10.0-20230421` 已于 2023 年 4 月 28 日发布。与上一版本相比，此版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：`emr-6.10.0-20230421` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.10.0:20230421`

emr-6.10.0-20230403

发布说明：`6.10.0-20230403` 已于 2023 年 4 月 12 日发布。与上一版本相比，此版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：emr-6.10.0-20230403 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：emr-6.10.0:20230403

emr-6.10.0-20230220

发布说明：emr-6.10.0-20230220 已于 2023 年 2 月 20 日发布。这是 Amazon EMR 6.10.0 的初始版本。

区域：emr-6.10.0-20230220 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：emr-6.10.0:20230220

Amazon EMR on EKS 6.9.0 版本

以下 Amazon EMR 6.9.0 版本适用于 Amazon EMR on EKS。选择特定的 emr-6.9.0-XXXX 版本以查看更多详细信息，例如相关的容器映像标签。

- emr-6.9.0-latest
- emr-6.9.0-20230905
- emr-6.9.0-20230624
- emr-6.9.0-20221108
- emr-6.9.0-spark-rapids-latest
- emr-6.9.0-spark-rapids-20230624
- emr-6.9.0-spark-rapids-20221108
- notebook-spark/emr-6.9.0-latest
- notebook-spark/emr-6.9.0-20230624
- notebook-spark/emr-6.9.0-20221108
- notebook-python/emr-6.9.0-latest
- notebook-python/emr-6.9.0-20230624
- notebook-python/emr-6.9.0-20221108

Amazon EMR 6.9.0 的发布说明

- 支持的应用程序- 适用于 Java 的 Amazon SDK 1.12.331、Spark 3.3.0-amzn-1、Hudi 0.12.1-amzn-0、Iceberg 0.14.1-amzn-0、Delta 2.1.0。
- 支持的组件：aws-sagemaker-spark-sdk、emr-ddb、emr-goodies、emr-s3-select、emrfs、hadoop-client、hudi、hudi-spark、iceberg、spark-kubernetes。
- 受支持的配置分类：

  与StartJobRun和一起使用 CreateManagedEndpoint APIs：

分类	描述
core-site	更改 Hadoop 的 core-site.xml 文件中的值。
emrfs-site	更改 EMRFS 设置。
spark-metrics	更改 Spark 的 metrics.properties 文件中的值。
spark-defaults	更改 Spark 的 spark-defaults.conf 文件中的值。
spark-env	更改 Spark 环境中的值。
spark-hive-site	更改 Spark 的 hive-site.xml 文件中的值。
spark-log4j	更改 Spark 的 log4j.properties 文件中的值。

  专门用于以下用途 CreateManagedEndpoint APIs：

分类	描述
jeg-config	更改 Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 文件中的值。
jupyter-kernel-overrides	更改 Jupyter 内核规范文件中内核映像的值。

  配置分类允许您自定义应用程序。这些通常与应用程序的配置 XML 文件（例如 spark-hive-site.xml）相对应。有关更多信息，请参阅配置应用程序。

显著功能

- 适用于 Apache Spark 的 Nvidia RAPIDS 加速器 ——EKS 上的 Amazon EMR， EC2 使用图形处理单元 (GPU) 实例类型加速 Spark。要将 Spark 图像与 RAPIDS Accelerator 一起使用，请将发布标签指定为 emr-6.9.0-。spark-rapids-latest请访问文档页面以了解更多信息。
- Spark-Redshift 连接器：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 on EKS 上使用适用于 Apache Spark 的 Amazon Redshift 集成。
- Delta Lake：Delta Lake 是一种开源存储格式，可以构建具有事务一致性、对数据集进行一致定义、更改架构发展和数据的数据湖。请访问 使用 Delta Lake 以了解更多信息。
- 修改 PySpark 参数 ——交互式端点现在支持修改与 EMR Studio Jupyter 笔记本中的 PySpark 会话关联的 Spark 参数。要了解更多信息，请访问修改会 PySpark 话参数。

已解决的问题

- 在 Amazon EMR 版本 6.6.0、6.7.0 和 6.8.0 上将 DynamoDB 连接器与 Spark 结合使用时，即使输入拆分引用了非空数据，表中的所有读取都会返回空结果。Amazon EMR 发行版 6.9.0 修复了此问题。
- Amazon EMR on EKS 6.8.0 错误地填充了使用 Apache Spark 生成的 Parquet 文件元数据中的构建哈希。此问题可能会导致解析由 Amazon EMR on EKS 6.8.0 生成的 Parquet 文件中的元数据版本字符串的工具失败。

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-6.9.0-latest

发布说明：`emr-6.9.0-latest` 当前指向 `emr-6.9.0-20230905`。

区域：`emr-6.9.0-latest` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.9.0:latest`

emr-6.9.0-20230905

发布说明：`emr-6.9.0-20230905`。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：`emr-6.9.0-20230905` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.9.0:20230905`

emr-6.9.0-20230624

发布说明：`emr-6.9.0-20230624` 已于 2023 年 7 月 7 日发布。

区域：`emr-6.9.0-20230624` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.9.0:20230624`

emr-6.9.0-20221108

发布说明：`emr-6.9.0-20221108` 已于 2022 年 12 月 8 日发布。这是 Amazon EMR 6.9.0 的初始版本。

区域：`emr-6.9.0-20221108` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.9.0:20221108`

Amazon EMR on EKS 6.8.0 版本

以下 Amazon EMR 6.8.0 版本适用于 Amazon EMR on EKS。选择特定的 emr-6.8.0-XXXX 版本以查看更多详细信息，例如相关的容器映像标签。

- emr-6.8.0-latest
- emr-6.8.0-20230905
- emr-6.8.0-20230624

- emr-6.8.0-20221219
- emr-6.8.0-20220802

Amazon EMR 6.8.0 的发布说明

- 支持的应用程序- 适用于 Java 的 Amazon SDK 1.12.170、Spark 3.3.0-amzn-0、Hudi 0.11.1-amzn-0、Iceberg 0.14.0-amzn-0。
- 支持的组件：`aws-sagemaker-spark-sdk`、`emr-ddb`、`emr-goodies`、`emr-s3-select`、`emrfs`、`hadoop-client`、`hudi`、`hudi-spark`、`iceberg`、`spark-kubernetes`。
- 受支持的配置分类：

分类	描述
`core-site`	更改 Hadoop 的 core-site.xml 文件中的值。
`emrfs-site`	更改 EMRFS 设置。
`spark-metrics`	更改 Spark 的 metrics.properties 文件中的值。
`spark-defaults`	更改 Spark 的 spark-defaults.conf 文件中的值。
`spark-env`	更改 Spark 环境中的值。
`spark-hive-site`	更改 Spark 的 hive-site.xml 文件中的值。
`spark-log4j`	更改 Spark 的 log4j.properties 文件中的值。

配置分类允许您自定义应用程序。这些通常与应用程序的配置 XML 文件（例如 `spark-hive-site.xml`）相对应。有关更多信息，请参阅配置应用程序。

显著功能

- Spark3.3.0 - Amazon EMR EKS 6.8 包括 Spark 3.3.0，支持为 Spark 驱动程序和执行程序容器组（Pod）使用单独的节点选择器标签。这些新标签使您无需使用 pod 模板即可在 StartJobRun API 中分别定义驱动程序和执行程序 pod 的节点类型。
  - 驱动程序节点选择器属性：spark.kubernetes.driver.node.selector.[labelKey]

  - 执行程序节点选择器属性：spark.kubernetes.executor.node.selector.[labelKey]
- 增强型任务失败消息 - 此版本引入了 `spark.stage.extraDetailsOnFetchFailures.enabled` 和 `spark.stage.extraDetailsOnFetchFailures.maxFailuresToInclude` 配置来跟踪由于用户代码而导致的任务失败。当阶段由于随机获取失败而中止时，这些详细信息可用于增强驱动程序日志中显示的失败消息。

属性名称	默认值	含义	最低版本
`spark.stage.extraDetailsOnFetchFailures.enabled`	false	如果设置为 `true`，则当阶段由于随机获取失败而中止时，此属性可用于增强驱动程序日志中显示的任务失败消息。默认情况下，系统会跟踪由用户代码导致的最近 5 次任务失败，并将失败错误消息附加到驱动程序日志中。 若要增加要跟踪的用户异常任务失败次数，请参阅配置 `spark.stage.extraDetailsOnFetchFailures.maxFailuresToInclude` 。	emr-6.8
`spark.stage.extraDetailsOnFetchFailu`	5	每个阶段每次尝试跟踪的任务失败次数。当阶段由于随机获取失败而中止时，此属性可用于增强驱动程	emr-6.8

属性名称	默认值	含义	最低版本
`res.maxFailuresToInclude`		序日志中显示的任务失败消息，并显示用户异常。 仅当 Config spark.stage 时，此属性才起作用。 extraDetailsOnFetchFailures.enabled 设置为 true。	

有关更多信息，请参阅 Apache Spark 配置文档。

已知问题

- Amazon EMR on EKS 6.8.0 错误地填充了使用 Apache Spark 生成的 Parquet 文件元数据中的构建哈希。此问题可能会导致解析由 Amazon EMR on EKS 6.8.0 生成的 Parquet 文件中的元数据版本字符串的工具失败。解析 Parquet 元数据中的版本字符串并依赖构建哈希的客户，应切换到其他 Amazon EMR 版本并重写该文件。

已解决的问题

- PySpark 内核的中断内核功能 - 可使用 `Interrupt Kernel` 功能，停止由在 Notebook 中执行单元格触发的进行中交互式工作负载。已引入修复程序，以便此功能适用于 PySpark 内核。这也可以在开源版本的变更中找到，用于处理 PySpark Kubernetes 内核 #1115 的中断。

emr-6.8.0-latest

发布说明：`emr-6.8.0-latest` 当前指向 `emr-6.8.0-20230624`。

区域：`emr-6.8.0-latest` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.8.0:latest`

emr-6.8.0-20230905

发布说明：`emr-6.8.0-20230905` 已于 2023 年 9 月 29 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：`emr-6.8.0-20230905` 已在支持 Amazon EMR on EKS 的所有区域开放。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.8.0:20230905`

emr-6.8.0-20230624

发布说明：`emr-6.8.0-20230624` 已于 2023 年 7 月 7 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：`emr-6.8.0-20230624` 已在支持 Amazon EMR on EKS 的所有区域开放。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.8.0:20230624`

emr-6.8.0-20221219

发布说明：`emr-6.8.0-20221219` 已于 2023 年 1 月 19 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：`emr-6.8.0-20221219` 已在支持 Amazon EMR on EKS 的所有区域开放。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.8.0:20221219`

emr-6.8.0-20220802

发布说明：`emr-6.8.0-20220802` 已于 2022 年 9 月 27 日发布。这是 Amazon EMR 6.8.0 的初始版本。

区域：`emr-6.8.0-20220802` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.8.0:20220802`

Amazon EMR on EKS 6.7.0 版本

以下 Amazon EMR 6.7.0 版本适用于 Amazon EMR on EKS。选择特定的 emr-6.7.0-XXXX 版本以查看更多详细信息，例如相关的容器映像标签。

- emr-6.7.0-latest
- emr-6.7.0-20240321
- emr-6.7.0-20230624
- emr-6.7.0-20221219
- emr-6.7.0-20220630

Amazon EMR 6.7.0 的发布说明

- 受支持的应用程序 - Spark 3.2.1-amzn-0、Jupyter Enterprise Gateway 2.6、Hudi 0.11-amzn-0、Iceberg 0.13.1。
- 受支持的组件 – `aws-hm-client`（Glue 连接器）、`aws-sagemaker-spark-sdk`、`emr-s3-select`、`emrfs`、`emr-ddb`、`hudi-spark`。
- 升级到 JEG 2.6 后，内核管理现在是异步的，这意味着 JEG 不会在内核启动过程中阻塞事务。这通过提供以下功能极大地改善了用户体验：
  - 在其他内核启动过程中在当前运行的 Notebook 中执行命令的功能
  - 同时启动多个内核而不会影响已在运行的内核的功能
- 受支持的配置分类：

分类	描述
`core-site`	更改 Hadoop `core-site.xml` 文件中的值。
`emrfs-site`	更改 EMRFS 设置。
`spark-metrics`	更改 Spark `metrics.properties` 文件中的值。
`spark-defaults`	更改 Spark `spark-defaults.conf` 文件中的值。

分类	描述
spark-env	更改 Spark 环境中的值。
spark-hive-site	更改 Spark hive-site.xml 文件中的值。
spark-log4j	更改 Spark log4j.properties 文件中的值。

配置分类允许您自定义应用程序。这些通常与应用程序的配置 XML 文件（例如 spark-hive-site.xml）相对应。有关更多信息，请参阅配置应用程序。

已解决的问题

- Amazon EMR on EKS 6.7 修复了 6.6 中将 Apache Spark 的 Pod 模板功能与交互式端点结合使用时的问题。Amazon EMR on EKS 版本 6.4、6.5 和 6.6 中存在此问题。现在，您可以使用 Pod 模板来定义使用交互式端点来运行交互式分析时，Spark 驱动程序和执行程序 Pod 的启动方式。
- 在之前的 Amazon EMR on EKS 版本中，Jupyter Enterprise Gateway 会在内核启动过程中阻止事务，这阻碍了当前正在运行的 Notebook 会话的执行。您现在可以在其他内核启动过程中在当前运行的 Notebook 中执行命令。您还可以同时启动多个内核，而不会丢失与已在运行的内核的连接。

emr-6.7.0-latest

发布说明：emr-6.7.0-latest 当前指向 emr-6.7.0-20240321。

区域：emr-6.7.0-latest 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：emr-6.7.0:latest

emr-6.7.0-20240321

发布说明：emr-6.7.0-20240321 已于 2024 年 3 月 11 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：emr-6.7.0-20240321 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.7.0:20240321`

emr-6.7.0-20230624

发布说明：`emr-6.7.0-20230624` 已于 2023 年 7 月 7 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：`emr-6.7.0-20230624` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.7.0:20230624`

emr-6.7.0-20221219

发布说明：`emr-6.7.0-20221219` 已于 2023 年 1 月 19 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：`emr-6.7.0-20221219` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.7.0:20221219`

emr-6.7.0-20220630

发布说明：`emr-6.7.0-20220630` 已于 2022 年 7 月 12 日发布。这是 Amazon EMR 6.7.0 的初始版本。

区域：`emr-6.7.0-20220630` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.7.0:20220630`

Amazon EMR on EKS 6.6.0 版本

以下 Amazon EMR 6.6.0 版本适用于 Amazon EMR on EKS。选择特定的 emr-6.6.0-XXXX 版本以查看更多详细信息，例如相关的容器映像标签。

- emr-6.6.0-最新版本
- emr-6.6.0-20240321
- emr-6.6.0-20230624

- emr-6.6.0-20221219
- emr-6.6.0-20220411

Amazon EMR 6.6.0 的版本注释

- 受支持的应用程序 - Spark 3.2.0-amzn-0、Jupyter Enterprise Gateway（端点、公共预览版）、Hudi 0.10.1-amzn-0、Iceberg 0.13.1。
- 受支持的组件 – `aws-hm-client`（Glue 连接器）、`aws-sagemaker-spark-sdk`、`emr-s3-select`、`emrfs`、`emr-ddb`、`hudi-spark`。
- 受支持的配置分类：

分类	描述
`core-site`	更改 Hadoop 的 core-site.xml 文件中的值。
`emrfs-site`	更改 EMRFS 设置。
`spark-metrics`	更改 Spark 的 metrics.properties 文件中的值。
`spark-defaults`	更改 Spark 的 spark-defaults.conf 文件中的值。
`spark-env`	更改 Spark 环境中的值。
`spark-hive-site`	更改 Spark 的 hive-site.xml 文件中的值。
`spark-log4j`	更改 Spark 的 log4j.properties 文件中的值。

配置分类允许您自定义应用程序。它们通常对应于应用程序的配置 XML 文件，例如 spark-hive-site .xml。有关更多信息，请参阅配置应用程序。

已知问题

- 在 Amazon EMR on EKS 版本 6.4、6.5 和 6.6 中，具有交互式端点的 Spark Pod 模板功能不起作用。

已解决的问题

- 交互式端点日志将上传到 Cloudwatch 和 S3。

emr-6.6.0-最新版本

发布说明：`emr-6.6.0-latest` 当前指向 `emr-6.6.0-20240321`。

区域：`emr-6.6.0-latest` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.6.0:latest`

emr-6.6.0-20240321

发布说明：`emr-6.6.0-20240321` 已于 2024 年 3 月 11 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：`emr-6.6.0-20240321` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.6.0:20240321`

emr-6.6.0-20230624

发布说明：`emr-6.6.0-20230624` 已于 2023 年 1 月 27 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：`emr-6.6.0-20230624` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.6.0:20230624`

emr-6.6.0-20221219

发布说明：`emr-6.6.0-20221219` 已于 2023 年 1 月 27 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：`emr-6.6.0-20221219` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.6.0:20221219`

emr-6.6.0-20220411

发布说明：`emr-6.6.0-20220411` 已于 2022 年 5 月 20 日发布。这是 Amazon EMR 6.6.0 的初始版本。

区域：`emr-6.6.0-20220411` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.6.0:20220411`

Amazon EMR on EKS 6.5.0 版本

以下 Amazon EMR 6.5.0 版本适用于 Amazon EMR on EKS。选择特定的 emr-6.5.0-XXXX 版本以查看更多详细信息，例如相关的容器映像标签。

- emr-6.5.0-最新版本
- emr-6.5.0-20240321
- emr-6.5.0-20221219
- emr-6.5.0-20220802
- emr-6.5.0-20211119

Amazon EMR 6.5.0 的发布说明

- 受支持的应用程序 - Spark 3.1.2-amzn-1、Jupyter Enterprise Gateway（终端节点，公共预览版）。
- 受支持的组件 – `aws-hm-client`（Glue 连接器）、`aws-sagemaker-spark-sdk`、`emr-s3-select`、`emrfs`、`emr-ddb`、`hudi-spark`。
- 受支持的配置分类：

分类	描述
`core-site`	更改 Hadoop 的 core-site.xml 文件中的值。
`emrfs-site`	更改 EMRFS 设置。
`spark-metrics`	更改 Spark 的 metrics.properties 文件中的值。

分类	描述
`spark-defaults`	更改 Spark 的 spark-defaults.conf 文件中的值。
`spark-env`	更改 Spark 环境中的值。
`spark-hive-site`	更改 Spark 的 hive-site.xml 文件中的值。
`spark-log4j`	更改 Spark 的 log4j.properties 文件中的值。

配置分类允许您自定义应用程序。它们通常对应于应用程序的配置 XML 文件，例如 spark-hive-site .xml。有关更多信息，请参阅配置应用程序。

已知问题

- 在 Amazon EMR on EKS 版本 6.4 和 6.5 中，具有交互式端点的 Spark Pod 模板功能不起作用。

emr-6.5.0-最新版本

发布说明：`emr-6.5.0-latest` 当前指向 `emr-6.5.0-20240321`。

区域：`emr-6.5.0-latest` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.5.0:latest`

emr-6.5.0-20240321

发布说明：`emr-6.5.0-20240321` 已于 2024 年 3 月 11 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：`emr-6.5.0-20240321` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.5.0:20240321`

emr-6.5.0-20221219

发布说明：`emr-6.5.0-20221219` 已于 2023 年 1 月 19 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：`emr-6.5.0-20221219` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.5.0:20221219`

emr-6.5.0-20220802

发布说明：`emr-6.5.0-20220802` 已于 2022 年 8 月 24 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包刷新。

区域：`emr-6.5.0-20220802` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.5.0:20220802`

emr-6.5.0-20211119

发布说明：`emr-6.5.0-20211119` 已于 2022 年 1 月 20 日发布。这是 Amazon EMR 6.5.0 的初始版本。

区域：`emr-6.5.0-20211119` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.5.0:20211119`

Amazon EMR on EKS 6.4.0 版本

以下 Amazon EMR 6.4.0 版本适用于 Amazon EMR on EKS。选择特定的 emr-6.4.0-XXXX 版本以查看更多详细信息，例如相关的容器映像标签。

- emr-6.4.0-latest
- emr-6.4.0-20240321
- emr-6.4.0-20221219
- emr-6.4.0-20210830

Amazon EMR 6.4.0 的发布说明

- 受支持的应用程序 - Spark 3.1.2-amzn-0、Jupyter Enterprise Gateway（终端节点，公共预览版）。
- 受支持的组件 – `aws-hm-client`（Glue 连接器）、`aws-sagemaker-spark-sdk`、`emr-s3-select`、`emrfs`、`emr-ddb`、`hudi-spark`。
- 受支持的配置分类：

分类	描述
`core-site`	更改 Hadoop 的 core-site.xml 文件中的值。
`emrfs-site`	更改 EMRFS 设置。
`spark-metrics`	更改 Spark 的 metrics.properties 文件中的值。
`spark-defaults`	更改 Spark 的 spark-defaults.conf 文件中的值。
`spark-env`	更改 Spark 环境中的值。
`spark-hive-site`	更改 Spark 的 hive-site.xml 文件中的值。
`spark-log4j`	更改 Spark 的 log4j.properties 文件中的值。

配置分类允许您自定义应用程序。它们通常对应于应用程序的配置 XML 文件，例如 spark-hive-site .xml。有关更多信息，请参阅配置应用程序。

已知问题

- 在 Amazon EMR on EKS 版本 6.4 中，具有交互式端点的 Spark Pod 模板功能不起作用。

emr-6.4.0-latest

发布说明：`emr-6.4.0-latest` 当前指向 `emr-6.4.0-20240321`。

区域：`emr-6.4.0-latest` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.4.0:latest`

emr-6.4.0-20240321

发布说明：`emr-6.4.0-20240321` 已于 2024 年 3 月 11 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：`emr-6.4.0-20240321` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.4.0:20240321`

emr-6.4.0-20221219

发布说明：`emr-6.4.0-20221219` 已于 2023 年 1 月 27 日发布。与上一版本相比，本版本已使用最近添加的 Amazon Linux 软件包刷新。

区域：`emr-6.4.0-20221219` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.4.0:20221219`

emr-6.4.0-20210830

发布说明：`emr-6.4.0-20210830` 已于 2021 年 12 月 9 日发布。这是 Amazon EMR 6.4.0 的初始版本。

区域：`emr-6.4.0-20210830` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.4.0:20210830`

Amazon EMR on EKS 6.3.0 版本

以下 Amazon EMR 6.3.0 版本适用于 Amazon EMR on EKS。选择特定的 emr-6.3.0-XXXX 版本以查看更多详细信息，例如相关的容器映像标签。

- emr-6.3.0-latest
- emr-6.3.0-20240321
- emr-6.3.0-20220802
- emr-6.3.0-20211008

- emr-6.3.0-20210802
- emr-6.3.0-20210429

Amazon EMR 6.3.0 的版本注释

- 新功能 - 从 6.x 版本系列中的 Amazon EMR 6.3.0 开始，Amazon EMR on EKS 支持 Spark 的 Pod 模板功能。您还可以为 Amazon EMR on EKS 启用 Spark 事件日志转动功能。有关更多信息，请参阅使用 Pod 模板和使用 Spark 事件日志轮替。
- 受支持的应用程序 - Spark 3.1.1-amzn-0、Jupyter Enterprise Gateway（终端节点，公共预览版）。
- 受支持的组件 – `aws-hm-client`（Glue 连接器）、`aws-sagemaker-spark-sdk`、`emr-s3-select`、`emrfs`、`emr-ddb`、`hudi-spark`。
- 受支持的配置分类：

分类	描述
`core-site`	更改 Hadoop 的 core-site.xml 文件中的值。
`emrfs-site`	更改 EMRFS 设置。
`spark-metrics`	更改 Spark 的 metrics.properties 文件中的值。
`spark-defaults`	更改 Spark 的 spark-defaults.conf 文件中的值。
`spark-env`	更改 Spark 环境中的值。
`spark-hive-site`	更改 Spark 的 hive-site.xml 文件中的值。
`spark-log4j`	更改 Spark 的 log4j.properties 文件中的值。

配置分类允许您自定义应用程序。它们通常对应于应用程序的配置 XML 文件，例如 spark-hive-site .xml。有关更多信息，请参阅配置应用程序。

emr-6.3.0-latest

发布说明：`emr-6.3.0-latest` 当前指向 `emr-6.3.0-20240321`。

区域：`emr-6.3.0-latest` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.3.0:latest`

emr-6.3.0-20240321

发布说明：`emr-6.3.0-20240321` 已于 2024 年 3 月 11 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：`emr-6.3.0-20240321` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.3.0:20240321`

emr-6.3.0-20220802

发布说明：`emr-6.3.0-20220802` 已于 2022 年 9 月 27 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包刷新。

区域：`emr-6.3.0-20220802` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.3.0:20220802`

emr-6.3.0-20211008

发布说明：`emr-6.3.0-20211008` 已于 2021 年 12 月 9 日发布。与以前的版本相比，此版本包含问题修复和安全更新。

区域：`emr-6.3.0-20211008` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.3.0:20211008`

emr-6.3.0-20210802

版本注释：`emr-6.3.0-20210802` 已于 2021 年 8 月 2 日发布。与以前的版本相比，此版本包含问题修复和安全更新。

区域：emr-6.3.0-20210802 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：emr-6.3.0:20210802

emr-6.3.0-20210429

发布说明：emr-6.3.0-20210429 已于 2021 年 4 月 29 日发布。这是 Amazon EMR 6.3.0 的初始版本。

区域：emr-6.3.0-20210429 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：emr-6.3.0:20210429

Amazon EMR on EKS 6.2.0 版本

以下 Amazon EMR 6.2.0 版本适用于 Amazon EMR on EKS。选择特定的 emr-6.2.0-XXXX 版本以查看更多详细信息，例如相关的容器映像标签。

- emr-6.2.0-latest
- emr-6.2.0-20240321
- emr-6.2.0-20220802
- emr-6.2.0-20211008
- emr-6.2.0-20210802
- emr-6.2.0-20210615
- emr-6.2.0-20210129
- emr-6.2.0-20201218
- emr-6.2.0-20201201

Amazon EMR 6.2.0 的版本注释

- 受支持的应用程序 - Spark 3.0.1-amzn-0、Jupyter Enterprise Gateway（终端节点，公共预览版）。
- 受支持的组件 – aws-hm-client（Glue 连接器）、aws-sagemaker-spark-sdk、emr-s3-select、emrfs、emr-ddb、hudi-spark。
- 受支持的配置分类：

分类	描述
core-site	更改 Hadoop 的 core-site.xml 文件中的值。
emrfs-site	更改 EMRFS 设置。
spark-metrics	更改 Spark 的 metrics.properties 文件中的值。
spark-defaults	更改 Spark 的 spark-defaults.conf 文件中的值。
spark-env	更改 Spark 环境中的值。
spark-hive-site	更改 Spark 的 hive-site.xml 文件中的值。
spark-log4j	更改 Spark 的 log4j.properties 文件中的值。

配置分类允许您自定义应用程序。它们通常对应于应用程序的配置 XML 文件，例如 spark-hive-site .xml。有关更多信息，请参阅配置应用程序。

emr-6.2.0-latest

发布说明：`emr-6.2.0-latest` 当前指向 `emr-6.2.0-20240321`。

区域：`emr-6.2.0-latest` 已在支持 Amazon EMR on EKS 的所有区域开放。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.2.0:20240321`

emr-6.2.0-20240321

发布说明：`emr-6.2.0-20240321` 已于 2024 年 3 月 11 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：`emr-6.2.0-20240321` 已在支持 Amazon EMR on EKS 的所有区域开放。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-6.2.0:20240321`

emr-6.2.0-20220802

发布说明：emr-6.2.0-20220802 已于 2022 年 9 月 27 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包刷新。

区域：emr-6.2.0-20220802 已在支持 Amazon EMR on EKS 的所有区域开放。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：emr-6.2.0:20220802

emr-6.2.0-20211008

发布说明：emr-6.2.0-20211008 已于 2021 年 12 月 9 日发布。与以前的版本相比，此版本包含问题修复和安全更新。

区域：emr-6.2.0-20211008 已在以下区域推出：美国东部（弗吉尼亚北部）、美国西部（俄勒冈）、亚太地区（东京）、欧洲（爱尔兰）、南美洲（爱尔兰）、南美洲（圣保罗）。

容器镜像标签：emr-6.2.0:20211008

emr-6.2.0-20210802

版本注释：emr-6.2.0-20210802 已于 2021 年 8 月 2 日发布。与以前的版本相比，此版本包含问题修复和安全更新。

区域：emr-6.2.0-20210802 已在以下区域推出：美国东部（弗吉尼亚北部）、美国西部（俄勒冈）、亚太地区（东京）、欧洲（爱尔兰）、南美洲（爱尔兰）、南美洲（圣保罗）。

容器镜像标签：emr-6.2.0:20210802

emr-6.2.0-20210615

版本注释：emr-6.2.0-20210615 已于 2021 年 6 月 15 日发布。与以前的版本相比，此版本包含问题修复和安全更新。

区域：emr-6.2.0-20210615 已在以下区域推出：美国东部（弗吉尼亚北部）、美国西部（俄勒冈）、亚太地区（东京）、欧洲（爱尔兰）、南美洲（爱尔兰）、南美洲（圣保罗）。

容器镜像标签：emr-6.2.0:20210615

emr-6.2.0-20210129

发布说明：emr-6.2.0-20210129 已于 2021 年 1 月 29 日发布。比较 emr-6.2.0-20201218，此版本包含问题修复和安全更新。

区域：emr-6.2.0-20210129 已在以下区域推出：美国东部（弗吉尼亚北部）、美国西部（俄勒冈）、亚太地区（东京）、欧洲（爱尔兰）、南美洲（爱尔兰）、南美洲（圣保罗）。

容器镜像标签：emr-6.2.0-20210129

emr-6.2.0-20201218

发布说明：emr-6.2.0-20201218 已于 2020 年 12 月 18 日发布。比较 emr-6.2.0-20201201，此版本包含问题修复和安全更新。

区域：emr-6.2.0-20201218 已在以下区域推出：美国东部（弗吉尼亚北部）、美国西部（俄勒冈）、亚太地区（东京）、欧洲（爱尔兰）、南美洲（爱尔兰）、南美洲（圣保罗）。

容器镜像标签：emr-6.2.0-20201218

emr-6.2.0-20201201

发布说明：emr-6.2.0-20201201 已于 2020 年 12 月 1 日发布。这是 Amazon EMR 6.2.0 的初始版本。

区域：emr-6.2.0-20201201 已在以下区域推出：美国东部（弗吉尼亚北部）、美国西部（俄勒冈）、亚太地区（东京）、欧洲（爱尔兰）、南美洲（爱尔兰）、南美洲（圣保罗）。

容器镜像标签：emr-6.2.0-20201201

Amazon EMR on EKS 5.36.0 版本

以下 Amazon EMR 5.36.0 版本适用于 Amazon EMR on EKS。选择特定的 emr-5.36.0-XXXX 版本以查看更多详细信息，例如相关的容器映像标签。

- emr-5.36.0-latest
- emr-5.36.0-20240321
- emr-5.36.0-20221219
- emr-5.36.0-20220620

- emr-5.36.0-20220525

Amazon EMR 5.36.0 的发行说明

- 已修复 log4j2 的安全问题。
- 支持的应用程序 – Spark 2.4.8-amzn-2、Jupyter Enterprise Gateway（端点，公开预览版；不支持 Scala 内核）、livy-0.7.1、fluentd-4.0.0。
- 支持的组件- aws-hm-client、、emr-ddb aws-sagemaker-spark-sdk、emr-goodies、emr-kinesis、kerberos-server。
- 受支持的配置分类：

分类	描述
`core-site`	更改 Hadoop 的 core-site.xml 文件中的值。
`emrfs-site`	更改 EMRFS 设置。
`spark-metrics`	更改 Spark 的 metrics.properties 文件中的值。
`spark-defaults`	更改 Spark 的 spark-defaults.conf 文件中的值。
`spark-env`	更改 Spark 环境中的值。
`spark-hive-site`	更改 Spark 的 hive-site.xml 文件中的值。
`spark-log4j`	更改 Spark 的 log4j.properties 文件中的值。

配置分类允许您自定义应用程序。它们通常对应于应用程序的配置 XML 文件，例如 spark-hive-site .xml。有关更多信息，请参阅配置应用程序。

emr-5.36.0-latest

发布说明：`emr-5.36.0-latest` 当前指向 `emr-5.36.0-20240321`。

区域：`emr-5.36.0-latest` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-5.36.0:latest`

emr-5.36.0-20240321

发布说明：`emr-5.36.0-20240321` 已于 2024 年 3 月 11 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：`emr-5.36.0-20240321` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 [Amazon EMR on EKS 服务终端节点]。

容器镜像标签：`emr-5.36.0:20240321`

emr-5.36.0-20221219

发布说明：`emr-5.36.0-20221219` 已于 2023 年 1 月 27 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包刷新。

区域：`emr-5.36.0-20221219` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 [Amazon EMR on EKS 服务终端节点]。

容器镜像标签：`emr-5.36.0:20221219`

emr-5.36.0-20220620

发布说明：`emr-5.36.0-20220620` 于 2022 年 7 月 27 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包刷新。

区域：`emr-5.36.0-20220620` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 [Amazon EMR on EKS 服务终端节点]。

容器镜像标签：`emr-5.36.0:20220620`

emr-5.36.0-20220525

发行说明：`emr-5.36.0-20220525` 于 2022 年 6 月 16 日发行。这是 Amazon EMR 5.36.0 的初始版本。

区域：`emr-5.36.0-20220525` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 [Amazon EMR on EKS 服务终端节点]。

容器镜像标签：`emr-5.36.0:20220525`

Amazon EMR on EKS 5.35.0 版本

以下 Amazon EMR 5.35.0 版本适用于 Amazon EMR on EKS。选择特定的 emr-5.35.0-XXXX 版本以查看更多详细信息，例如相关的容器映像标签。

- emr-5.35.0-latest
- emr-5.35.0-20240321
- emr-5.35.0-20221219
- emr-5.35.0-20220802
- emr-5.35.0-20220307

Amazon EMR 5.35.0 的版本注释

- 已修复 log4j2 的安全问题。
- 支持的应用程序 – Spark 2.4.8-amzn-1、Hudi 0.9.0-amzn-2、Jupyter Enterprise Gateway（端点，公开预览版；不支持 Scala 内核）。
- 支持的组件- aws-hm-client（Glue 连接器）、emr-s3-select aws-sagemaker-spark-sdk、emrfs、emr-ddb、hudi-spark。
- 受支持的配置分类：

分类	描述
`core-site`	更改 Hadoop 的 core-site.xml 文件中的值。
`emrfs-site`	更改 EMRFS 设置。
`spark-metrics`	更改 Spark 的 metrics.properties 文件中的值。
`spark-defaults`	更改 Spark 的 spark-defaults.conf 文件中的值。
`spark-env`	更改 Spark 环境中的值。
`spark-hive-site`	更改 Spark 的 hive-site.xml 文件中的值。

分类	描述
`spark-log4j`	更改 Spark 的 log4j.properties 文件中的值。

配置分类允许您自定义应用程序。它们通常对应于应用程序的配置 XML 文件，例如 spark-hive-site .xml。有关更多信息，请参阅配置应用程序。

emr-5.35.0-latest

发布说明：`emr-5.35.0-latest` 当前指向 `emr-5.35.0-20240321`。

区域：`emr-5.35.0-latest` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-5.35.0:latest`

emr-5.35.0-20240321

发布说明：`emr-5.35.0-20240321` 已于 2024 年 3 月 11 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：`emr-5.35.0-20240321` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-5.35.0:20240321`

emr-5.35.0-20221219

发布说明：`emr-5.35.0-20221219` 已于 2023 年 1 月 27 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包刷新。

区域：`emr-5.35.0-20221219` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-5.35.0:20221219`

emr-5.35.0-20220802

发布说明：`emr-5.35.0-20220802` 已于 2022 年 9 月 27 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包刷新。

区域：`emr-5.35.0-20220802` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 [Amazon EMR on EKS 服务终端节点](#)。

容器镜像标签：`emr-5.35.0:20220802`

emr-5.35.0-20220307

版本注释: `emr-5.35.0-20220307` 已于 2022 年 3 月 30 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包刷新。

区域：`emr-5.35.0-20220307` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 [Amazon EMR on EKS 服务终端节点](#)。

容器镜像标签：`emr-5.35.0:20220307`

Amazon EMR on EKS 5.34.0 版本

以下 Amazon EMR 5.34.0 版本适用于 Amazon EMR on EKS。选择特定的 emr-5.34.0-XXXX 版本以查看更多详细信息，例如相关的容器映像标签。

- emr-5.34.0-最新版本
- emr-5.34.0-20240321
- emr-5.34.0-20220802

Amazon EMR 5.34.0 的发布说明

- 支持的应用程序 – Spark 2.4.8-amzn-0、Jupyter Enterprise Gateway（端点，公开预览版；不支持 Scala 内核）。
- 受支持的组件 – `aws-hm-client`（Glue 连接器）、`aws-sagemaker-spark-sdk`、`emr-s3-select`、`emrfs`、`emr-ddb`、`hudi-spark`。
- 受支持的配置分类：

分类	描述
`core-site`	更改 Hadoop 的 core-site.xml 文件中的值。
`emrfs-site`	更改 EMRFS 设置。

分类	描述
spark-metrics	更改 Spark 的 metrics.properties 文件中的值。
spark-defaults	更改 Spark 的 spark-defaults.conf 文件中的值。
spark-env	更改 Spark 环境中的值。
spark-hive-site	更改 Spark 的 hive-site.xml 文件中的值。
spark-log4j	更改 Spark 的 log4j.properties 文件中的值。

配置分类允许您自定义应用程序。它们通常对应于应用程序的配置 XML 文件，例如 spark-hive-site .xml。有关更多信息，请参阅配置应用程序。

emr-5.34.0-最新版本

发布说明：emr-5.34.0-latest 当前指向 emr-5.34.0-20220802。

区域：emr-5.34.0-latest 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：emr-5.34.0:latest

emr-5.34.0-20240321

发布说明：emr-5.34.0-20240321 已于 2024 年 3 月 11 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：emr-5.34.0-20240321 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：emr-5.34.0:20240321

emr-5.34.0-20220802

发布说明：emr-5.34.0-20220802 已于 2022 年 8 月 24 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包刷新。

区域：emr-5.34.0-20220802 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：emr-5.34.0:20220802

emr-5.34.0-20211208

发布说明：emr-5.34.0-20211208 已于 2022 年 1 月 20 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包刷新。

区域：emr-5.34.0-20211208 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：emr-5.34.0:20211208

Amazon EMR on EKS 5.33.0 版本

以下 Amazon EMR 5.33.0 版本适用于 Amazon EMR on EKS。选择特定的 emr-5.33.0-XXXX 版本以查看更多详细信息，例如相关的容器映像标签。

- emr-5.33.0-latest
- emr-5.33.0-20240321
- emr-5.33.0-20221219
- emr-5.33.0-20220802
- emr-5.33.0-20211008
- emr-5.33.0-20210802
- emr-5.33.0-20210615
- emr-5.33.0-20210323

Amazon EMR 5.33.0 的版本注释

- 新功能 - 从 5.x 版本系列中的 Amazon EMR 5.33.0 开始，Amazon EMR on EKS 支持 Spark 的 Pod 模板功能。有关更多信息，请参阅 使用 Pod 模板。
- 支持的应用程序 – Spark 2.4.7-amzn-1、Jupyter Enterprise Gateway（端点，公开预览版；不支持 Scala 内核）。
- 受支持的组件 – aws-hm-client（Glue 连接器）、aws-sagemaker-spark-sdk、emr-s3-select、emrfs、emr-ddb、hudi-spark。

- 受支持的配置分类：

分类	描述
`core-site`	更改 Hadoop 的 core-site.xml 文件中的值。
`emrfs-site`	更改 EMRFS 设置。
`spark-metrics`	更改 Spark 的 metrics.properties 文件中的值。
`spark-defaults`	更改 Spark 的 spark-defaults.conf 文件中的值。
`spark-env`	更改 Spark 环境中的值。
`spark-hive-site`	更改 Spark 的 hive-site.xml 文件中的值。
`spark-log4j`	更改 Spark 的 log4j.properties 文件中的值。

配置分类允许您自定义应用程序。它们通常对应于应用程序的配置 XML 文件，例如 spark-hive-site .xml。有关更多信息，请参阅配置应用程序。

emr-5.33.0-latest

发布说明：`emr-5.33.0-latest` 当前指向 `emr-5.33.0-20240321`。

区域：`emr-5.33.0-latest` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-5.33.0:latest`

emr-5.33.0-20240321

发布说明：`emr-5.33.0-20240321` 已于 2024 年 3 月 11 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：`emr-5.33.0-20240321` 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-5.33.0:20240321`

emr-5.33.0-20221219

发布说明：emr-5.33.0-20221219 已于 2023 年 1 月 19 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：emr-5.33.0-20221219 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：emr-5.33.0:20221219

emr-5.33.0-20220802

发布说明：emr-5.33.0-20220802 已于 2022 年 8 月 24 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包刷新。

区域：emr-5.33.0-20220802 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：emr-5.33.0:20220802

emr-5.33.0-20211008

发布说明：emr-5.33.0-20211008 已于 2021 年 12 月 9 日发布。与以前的版本相比，此版本包含问题修复和安全更新。

区域：emr-5.33.0-20211008 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：emr-5.33.0:20211008

emr-5.33.0-20210802

版本注释：emr-5.33.0-20210802 已于 2021 年 8 月 2 日发布。与以前的版本相比，此版本包含问题修复和安全更新。

区域：emr-5.33.0-20210802 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：emr-5.33.0:20210802

emr-5.33.0-20210615

版本注释：emr-5.33.0-20210615 已于 2021 年 6 月 15 日发布。与以前的版本相比，此版本包含问题修复和安全更新。

区域：emr-5.33.0-20210615 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：emr-5.33.0:20210615

emr-5.33.0-20210323

发布说明：emr-5.33.0-20210323 已于 2021 年 3 月 23 日发布。这是 Amazon EMR 5.33.0 的初始版本。

区域：emr-5.33.0-20210323 适用于 Amazon EMR on EKS 支持的所有区域。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：emr-5.33.0-20210323

Amazon EMR on EKS 5.32.0 版本

以下 Amazon EMR 5.32.0 版本适用于 Amazon EMR on EKS。选择特定的 emr-5.32.0-XXXX 版本以查看更多详细信息，例如相关的容器映像标签。

- emr-5.32.0-latest
- emr-5.32.0-20240321
- emr-5.32.0-20220802
- emr-5.32.0-20211008
- emr-5.32.0-20210802
- emr-5.32.0-20210615
- emr-5.32.0-20210129
- emr-5.32.0-20201218
- emr-5.32.0-20201201

Amazon EMR 5.32.0 的版本注释

- 支持的应用程序 – Spark 2.4.7-amzn-0、Jupyter Enterprise Gateway（端点，公开预览版；不支持 Scala 内核）。
- 受支持的组件 – `aws-hm-client`（Glue 连接器）、`aws-sagemaker-spark-sdk`、`emr-s3-select`、`emrfs`、`emr-ddb`、`hudi-spark`。
- 受支持的配置分类：

分类	描述
`core-site`	更改 Hadoop 的 core-site.xml 文件中的值。
`emrfs-site`	更改 EMRFS 设置。
`spark-metrics`	更改 Spark 的 metrics.properties 文件中的值。
`spark-defaults`	更改 Spark 的 spark-defaults.conf 文件中的值。
`spark-env`	更改 Spark 环境中的值。
`spark-hive-site`	更改 Spark 的 hive-site.xml 文件中的值。
`spark-log4j`	更改 Spark 的 log4j.properties 文件中的值。

配置分类允许您自定义应用程序。它们通常对应于应用程序的配置 XML 文件，例如 spark-hive-site .xml。有关更多信息，请参阅配置应用程序。

emr-5.32.0-latest

发布说明：`emr-5.32.0-latest` 当前指向 `emr-5.32.0-20240321`。

区域：`emr-5.32.0-latest` 已在支持 Amazon EMR on EKS 的所有区域开放。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：`emr-5.32.0:latest`

emr-5.32.0-20240321

发布说明：emr-5.32.0-20240321 已于 2024 年 3 月 11 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包和关键修复刷新。

区域：emr-5.32.0-20240321 已在支持 Amazon EMR on EKS 的所有区域开放。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：emr-5.32.0:20240321

emr-5.32.0-20220802

发布说明：emr-5.32.0-20220802 已于 2022 年 8 月 24 日发布。与上一版本相比，本版本已使用最近更新的 Amazon Linux 软件包刷新。

区域：emr-5.32.0-20220802 已在支持 Amazon EMR on EKS 的所有区域开放。有关更多信息，请参阅 Amazon EMR on EKS 服务终端节点。

容器镜像标签：emr-5.32.0:20220802

emr-5.32.0-20211008

发布说明：emr-5.32.0-20211008 已于 2021 年 12 月 9 日发布。与以前的版本相比，此版本包含问题修复和安全更新。

区域：emr-5.32.0-20211008 已在以下区域推出：美国东部（弗吉尼亚北部）、美国西部（俄勒冈）、亚太地区（东京）、欧洲（爱尔兰）、南美洲（爱尔兰）、南美洲（圣保罗）。

容器镜像标签：emr-5.32.0:20211008

emr-5.32.0-20210802

版本注释：emr-5.32.0-20210802 已于 2021 年 8 月 2 日发布。与以前的版本相比，此版本包含问题修复和安全更新。

区域：emr-5.32.0-20210802 已在以下区域推出：美国东部（弗吉尼亚北部）、美国西部（俄勒冈）、亚太地区（东京）、欧洲（爱尔兰）、南美洲（爱尔兰）、南美洲（圣保罗）。

容器镜像标签：emr-5.32.0:20210802

emr-5.32.0-20210615

版本注释：emr-5.32.0-20210615 已于 2021 年 6 月 15 日发布。与以前的版本相比，此版本包含问题修复和安全更新。

区域：emr-5.32.0-20210615 已在以下区域推出：美国东部（弗吉尼亚北部）、美国西部（俄勒冈）、亚太地区（东京）、欧洲（爱尔兰）、南美洲（爱尔兰）、南美洲（圣保罗）。

容器镜像标签：emr-5.32.0:20210615

emr-5.32.0-20210129

发布说明：emr-5.32.0-20210129 已于 2021 年 1 月 29 日发布。比较 emr-5.32.0-20201218，此版本包含问题修复和安全更新。

区域：emr-5.32.0-20210129 已在以下区域推出：美国东部（弗吉尼亚北部）、美国西部（俄勒冈）、亚太地区（东京）、欧洲（爱尔兰）、南美洲（爱尔兰）、南美洲（圣保罗）。

容器镜像标签：emr-5.32.0-20210129

emr-5.32.0-20201218

发布说明：5.32.0-20201218 已于 2020 年 12 月 18 日发布。比较 5.32.0-20201201，此版本包含问题修复和安全更新。

区域：emr-5.32.0-20201218 已在以下区域推出：美国东部（弗吉尼亚北部）、美国西部（俄勒冈）、亚太地区（东京）、欧洲（爱尔兰）、南美洲（爱尔兰）、南美洲（圣保罗）。

容器镜像标签：emr-5.32.0-20201218

emr-5.32.0-20201201

发布说明：5.32.0-20201201 已于 2020 年 12 月 1 日发布。这是 Amazon EMR 5.32.0 的初始版本。

区域：5.32.0-20201201 已在以下区域推出：美国东部（弗吉尼亚北部）、美国西部（俄勒冈）、亚太地区（东京）、欧洲（爱尔兰）、南美洲（爱尔兰）、南美洲（圣保罗）。

容器镜像标签：emr-5.32.0-20201201

文档历史记录

下表介绍了自 Amazon EMR on EKS 上一次发布以来对文档所做的重要更改。如需对此文档更新的通知，您可以订阅 RSS 源。

更改	描述	日期
更新内容	Amazon EMR on EKS 的托管式策略— 的附加权限`AmazonEMRContainersServiceRolePolicy` 。	2025 年 2 月 3 日
新版本	EKS 7.6.0 版本上的 Amazon EMR	2025 年 1 月 10 日
新版本	EKS 7.5.0 版本上的 Amazon EMR	2024 年 11 月 21 日
新版本	EKS 7.4.0 版本上的 Amazon EMR	2024 年 11 月 13 日
新版本	Amazon EMR on EKS 7.3.0 发行版	2024 年 10 月 16 日
新版本	Amazon EMR on EKS 7.2.0 发行版	2024 年 7 月 25 日
新版本	Amazon EMR on EKS 7.1.0 发行版	2024 年 4 月 17 日
新版本	EKS 上的 Amazon EMR 7.0.0 发行版	2023 年 12 月 22 日
新版本	EKS 上的 Amazon EMR 6.15.0 发行版	2023 年 11 月 17 日
新版本	Amazon EMR on EKS 6.14.0 发行版	2023 年 10 月 17 日
更新内容	将“托管式端点”重命名为交互式端点；交互式端点正式发布	2023 年 9 月 29 日
新版本	Amazon EMR on EKS 6.13.0 版本，以及 使用 Amazon EMR on EKS 运行 Flink 任务 的公共预览文档	2023 年 9 月 12 日
新版本	Amazon EMR on EKS 6.12.0 版本	2023 年 7 月 21 日

更改	描述	日期
新增内容	新增了 在 Amazon EMR on EKS 上将 Volcano 用作 Apache Spark 自定义调度器	2023 年 6 月 13 日
新增内容	新增了 在 Amazon EMR on EKS 上将 Volcano 用作 Apache Spark 自定义调度器	2023 年 6 月 13 日
新增内容	新增了 使用 Spark 容器日志轮换	2023 年 6 月 12 日
更新内容	Amazon ECR Public Gallery 中用于查找基础映像信息的自定义映像文档。	2023 年 6 月 8 日
新版本	Amazon EMR on EKS 6.11.0 版本	2023 年 6 月 8 日
新增内容	添加 使用 Spark Operator 运行 Spark 任务 并重新组织了 使用 Amazon EMR on EKS 运行 Spark 作业 下的“任务运行”部分。	2023 年 6 月 5 日
新增内容	添加了两个部分：使用垂直自动扩展功能处理 Amazon EMR Spark 任务 和 使用自托管式 Jupyter notebook	2023 年 5 月 4 日
文档历史记录页面	为 Amazon EMR on EKS 创建文档历史记录页面。	2023 年 3 月 13 日
托管式策略页面	为 Amazon EMR on EKS 创建托管式策略页面。	2023 年 3 月 13 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。