ALIBABA CLOUD

阿里云

E-MapReduce

E-MapReduce公共云合集

文档版本：20220713

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.动态与公告
1.1. 产品公告
1.1.1. EMR元数据迁移公告

本文为您介绍迁移E-MapReduce（简称EMR）元数据至数据湖元数据DLF（Data Lake Formation）中的原因及迁移流程。

迁移原因

2020年阿里云EMR推出全新的数据湖构建和统一元数据服务，为EMR用户提供了更好的统一元数据服务方案。阿里云EMR团队发现部分用户在EMR集群上，仍然使用本地MySQL和统一meta数据库（旧版功能）作为生产环境的Hive元数据存储。我们强烈建议您尽快迁移到数据湖构建DLF中，原因如下：

- 本地MySQL是单机部署，无法保证服务高可用，容易造成服务中断。
- 旧版功能的统一元数据，后续将逐步下线，需要迁移到新版统一元数据DLF中。

> **说明** 如果您的集群使用自建RDS存储元数据，也可以迁移到统一元数据DLF中，以便于为您提供更好的性能和可扩展性。

数据湖元数据DLF是阿里云提供的统一元数据服务，具有高可用、免运维和高性能等优点，兼容Hive Metastore，无缝对接EMR上开源计算引擎，并支持多版本管理和Data Profile功能。另外，DLF还支持数据探索、湖管理和数据权限控制等功能，并与阿里云其他计算产品（例如MaxCompute、Databricks和Hologres等）无缝对接，可以扩展更丰富的计算场景。DLF详细介绍，请参见产品简介。

迁移流程

阿里云EMR和DLF团队提供了完善的迁移流程及技术工具支持，同时会在整个迁移过程中做好支持和保障工作，以确保快速平滑迁移。

步骤	具体描述	参与方	预计耗时
一、准备阶段	1. 填写元数据迁移登记表。 2. 请使用钉钉搜索钉钉群号33719678或扫描联系我们中的二维码加入阿里云EMR元数据迁移群。 阿里云EMR团队会分派工程师对接。 3. 阿里云EMR团队对客户集群和实际使用情况进行摸底，确定迁移的可行性及排期。	阿里云EMR团队+客户	2小时
二、迁移阶段	1. 暂停集群上运行的任务和停止元数据服务。 2. 备份用户现有元数据内容。 3. 在DLF上通过元数据迁移功能进行迁移和验证。 4. 切换集群元数据配置至DLF元数据。 5. 恢复业务任务。	阿里云EMR团队+客户	30分钟
三、验证阶段	观察作业运行一周或者更长时间，查看运行结果。 • 如果正常运行，则迁移成功。 • 如果遇到问题，则需要定位分析，可以根据实际情况确定是在线解决还是进入回滚阶段。 回滚详情，请参见步骤四。	阿里云EMR团队+客户	1周
（可选）四、回滚阶段	1. 暂停业务任务。 2. 对比DLF元数据和HMS元数据，回放增量部分。 3. 切换集群元数据配置至HMS元数据。 4. 启动HMS服务。 5. 恢复业务任务并观察运行结果。	阿里云EMR团队+客户	30分钟

联系我们

阿里云EMR和DLF团队，针对此次迁移提供了完善的迁移方案和技术工具支持，如有需要，请先填写元数据迁移登记表，然后使用钉钉搜索钉钉群号33719678或扫描下方二维码加入钉钉迁移群，我们会安排工程师与您对接具体方案。

1.1.2. EMR数据开发停止更新公告

2022年2月21日21点起，E-MapReduce（简称EMR）数据开发功能将停止更新，进入维护状态，会影响创建项目、新建和运行作业、工作流、数据开发运维等功能。如果您还在使用数据开发功能，请尽快迁移到EMR Studio构建数据开发平台。本文为您介绍数据开发模块停止更新的情况，以及迁移至EMR Studio的流程。

停止更新时间

2022年2月21日21点

影响

您在停更时间点前创建的数据开发项目不会受到影响，可正常使用EMR数据开发模块，包括运行作业和工作流调度。如果您未来需要更丰富的数据开发功能，推荐迁移至EMR Studio构建数据开发平台。EMR Studio是EMR提供的开源大数据开发套件，包含Airflow、Zeppelin和Jupyter等开源组件。能够无缝关联EMR集群（EMR on ECS和EMR on ACK）开源计算引擎提交任务，并给用户提供交互式作业开发、任务调度和任务监控等开源大数据开发服务。

迁移流程

阿里云EMR Studio团队提供了完善的迁移流程及技术工具支持，同时会在整个迁移过程中做好支持和保障工作，以确保快速平滑迁移。

步骤	具体描述	参与方	预计耗时
一、准备阶段	1. 填写阿里云EMR Studio迁移登记表。 2. 联系阿里云EMR团队确认待创建EMR Studio的规格，然后创建EMR Studio，详情请参见创建EMR Studio集群。 您可以查看EMR Studio概述文档了解EMR Studio。	阿里云EMR团队+客户	1天
二、迁移阶段	联系EMR团队，EMR团队会协助您完成将EMR数据开发自动化迁移至EMR Studio构建数据开发平台。	阿里云EMR团队+客户	2天
三、验证阶段	修改作业配置（例如数据路径等），验证运行作业是否符合预期。	阿里云EMR团队+客户	2~4周
四、完成迁移	关闭EMR数据开发里的工作流，全面切换到EMR Studio。	阿里云EMR团队+客户	1周

联系我们

如果您需要迁移至EMR Studio，请先填写阿里云EMR Studio迁移登记表，我们会安排工程师与您对接具体方案。如有任何问题，请提交提交工单联系售后服务。

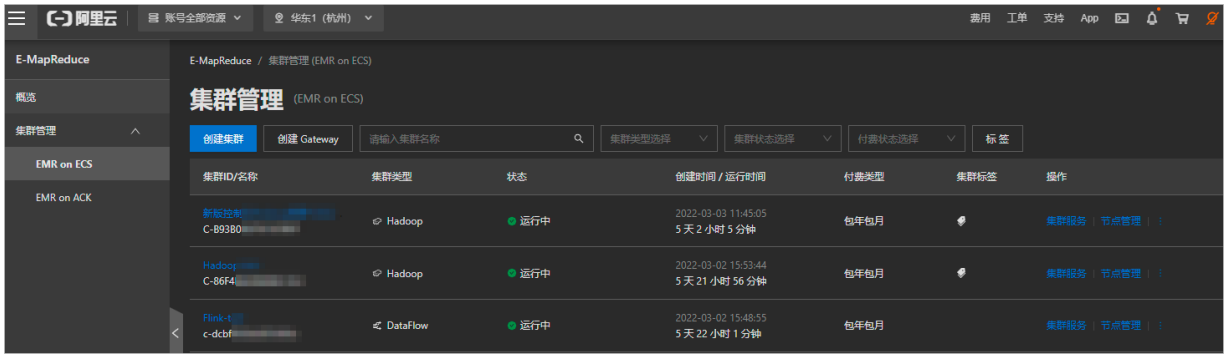
1.1.3. 新版控制台上线公告

E-MapReduce新版控制台为您提供更高性能，更佳的用户体验。本文为您介绍新版控制台的功能特点。

运维更便捷

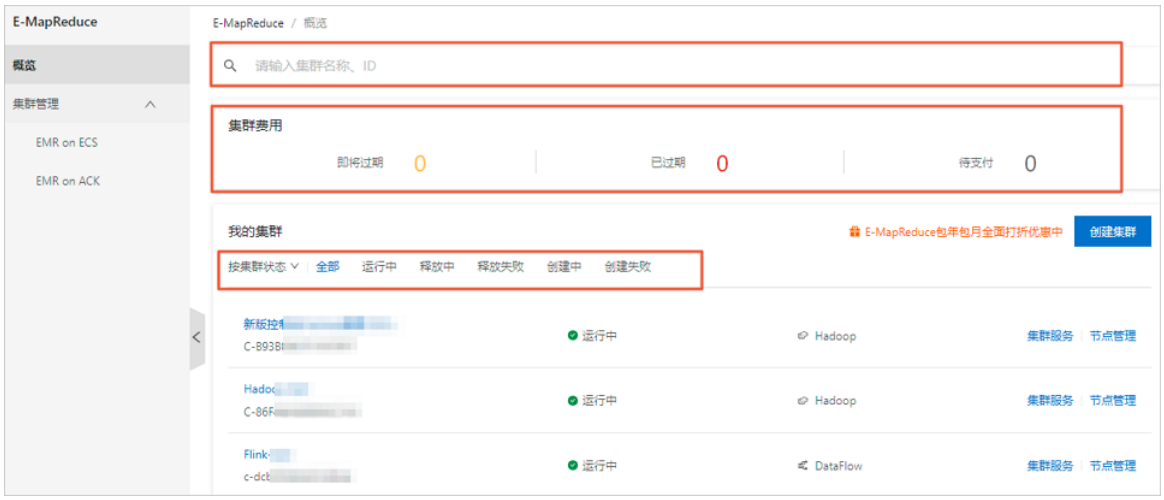
- 新版控制台增加了智能刷新功能，集群状态实时更新，页面切换更流畅。
- 新版实时数据流（Dataflow）集群，扩容速度提升了3倍以上，可以及时响应业务诉求。
- 新增深色模式开关，减轻视觉疲劳。

  您可以单击顶部的按钮，切换为暗色主题。



操作更精简

- 新增全局搜索栏，在任意页面3步内可快速管理目标集群。
- 新增集群费用展示模块，清晰掌握产品订单状态。
- 新增自定义切换集群列表，可按照集群状态或集群类型多角度了解集群现状。



搭建更高效

- 支持按照业务场景灵活搭配选所需服务组件。
- 新增可修改的配置清单，灵活跳转更改集群配置项。
- 新增数据分析（OLAP）和实时数据流（Dataflow）业务场景，创建集群速度提升了2倍以上。

联系我们

如果您有业务问题或者产品疑难问题，建议您在阿里云官网提交工单。

您可以扫描下方二维码进入E-MapReduce交流群。在E-MapReduce交流群中，您可以和其他用户进行交流。

1.1.4. YARN缺陷修复公告

本文为您介绍YARN-4946引入缺陷的影响范围以及相应的修复方案。

背景信息

YARN-4946引入缺陷，YARN ResourceManager删除历史应用作业时增加了日志收集已完成的必要条件，但日志收集的状态没有保留到ResourceManager State Store中，所以重启后ResourceManager加载的历史应用都是未完成状态，无法被ResourceManager自动删除掉，造成application在ResourceManager中堆积。这部分应用达到上限（相关配置为${yarn.resourcemanager.state-store.max-completed-applications} 或${yarn.resourcemanager.max-completed-applications}，默认值为10000）之后，会影响ResourceManager的调度。

具体缺陷引入的Issue为YARN-4946，详情请参见YARN-4946。

通过revert方式修复该缺陷的Issue为YARN-9571，详情请参见YARN-9571。

缺陷影响

- 缺陷影响的组件：Hadoop YARN（开启了服务高可用，并且添加了Zookeeper服务）。
- 缺陷级别：严重，建议修复，集群长时间运行重启后会导致集群不可用。
- 缺陷发生现象：ResourceManager日志一直打印“Application should be expired, max number of completed apps kept in memory met: maxCompletedAppsInMemory = 10000, but not removing app XXX from memory as log aggregation have not finished yet.”，导致YARN ResourceManager不可用，或者出现ResourceManager重启之后长时间不可用的情况。

缺陷修复方案

您需要将E-MapReduce集群中包含缺陷的Hadoop YARN ResourceManager的JAR包替换掉，然后重启ResourceManager服务，重启服务时先处理Standby ResourceManager，再处理Active ResourceManager。

该修复方案：

- 适用于EMR的4.6.0、4.7.0、4.8.0、4.9.0、5.1.0、5.2.0和5.2.1版本。

  > **说明** 对应Hadoop的3.2.1版本。

- 执行该修复方案后，需要重启对应的组件。重启组件可能会导致作业失败，所以建议业务低峰期时执行。

修复流程

按照先处理Standby ResourceManager，再处理Active ResourceManager的顺序，通过JAR包替换的方式修复ResourceManager组件缺陷。

> **注意**
> - 替换过程中，重启一个ResourceManager待正常运行之后再重启另外一个ResoureManager。
> - 如果集群没有开启高可用，则无需执行该修复流程。

1. 单击hadoop-yarn-server-resourcemanager-3.2.1.jar下载YARN ResourceManager JAR包。
2. 登录EMR集群的Master节点，将下载好的JAR包放在Hadoop的软件安装目录下。

   本示例是在*/usr/lib/hadoop-current/share/hadoop/yarn/*目录。
3. 备份旧的JAR包，并将新包拷贝到相应位置。

   ```
   mv $HADOOP_HOME/share/hadoop/yarn/hadoop-yarn-server-resourcemanager-3.2.1.jar /tmp/
   cp hadoop-yarn-server-resourcemanager-3.2.1.jar $HADOOP_HOME/share/hadoop/yarn/
   ```

   命令中的*$HADOOP_HOME*为Hadoop的安装目录。本文示例中Hadoop的安装目录为*/usr/lib/hadoop-current*。
4. 重启YARN ResourceManager服务。

   观察ResourceManager重启情况，如果重启之后ResourceManager日志不再提示“but not removing app XXX from memory as log aggregation have not finished yet.”的问题，并且作业可以正常提交，则说明修复成功。
5. 在控制台查找Active ResourceManager，重启YARN ResourceManager服务。

回滚流程

如果修复过程中遇到问题需要回滚，则可以执行以下命令，使用备份路径下的旧包替换掉补丁包，再重启该节点上的YARN ResourceManager。

```
cp /tmp/hadoop-yarn-server-resourcemanager-3.2.1.jar $HADOOP_HOME/share/hadoop/yarn/
```

1.1.5. SmartData数据读取异常修复公告

历史版本的SmartData（3.0.x～3.5.x）服务存在已知缺陷可能会造成缓存数据出现损坏，导致读取数据内容发生异常。本文为您介绍缺陷影响，缺陷方案以及缺陷修复流程。

缺陷影响

- 缺陷影响的组件：打开SmartData数据缓存功能的所有组件。

  > **注意** 如果集群部署了SmartData，但确定不会使用缓存可以忽略本缺陷。

  SmartData支持JindoFS Cache模式和JindoFS Block模式两种缓存模式。
- 缺陷影响的版本：
  - EMR版本：3.30.x、4.5.x、3.32.x、4.6.x、3.33.x、4.7.x、3.34.x、4.8.x、3.35.x、4.9.x。
  - SmartData版本：3.0.x、3.1.x、3.2.x、3.3.x、3.4.x、3.5.x。
- 缺陷级别：严重，建议修复，概率性发生时会出现数据正确性问题。
- 缺陷发生现象：如果集群启用JindoFS Cache模式（即设置数据缓存参数jfs.cache.data-cache.enable为true）或者使用了JindoFS Block模式（Block模式默认启用缓存），则数据缓存会出现小概率数据污染的情况，从而导致作业读取数据时报错。例如，作业对源数据读取报数据内容不正确的错误（ORC或Parquet文件格式无法解析）或HBase报HFile格式错误等。

缺陷修复方案

由于历史版本缓存损坏问题是由于Storage Service的小文件合并（compaction）流程的缺陷导致，通过修改compaction配置关闭该优化路径并重启SmartData服务，即可避免该问题的产生。如果已发生该问题，优先关闭缓存开关及时止损，以消除缓存数据的影响，尽快恢复线上业务；如果您仅启用了Cache模式，没用使用Block模式，则可以使用工具对集群全部缓存进行全量清理，彻底格式化缓存系统，从而清除集群中所有可能损坏的缓存块，清理完成后可以重新启用缓存。

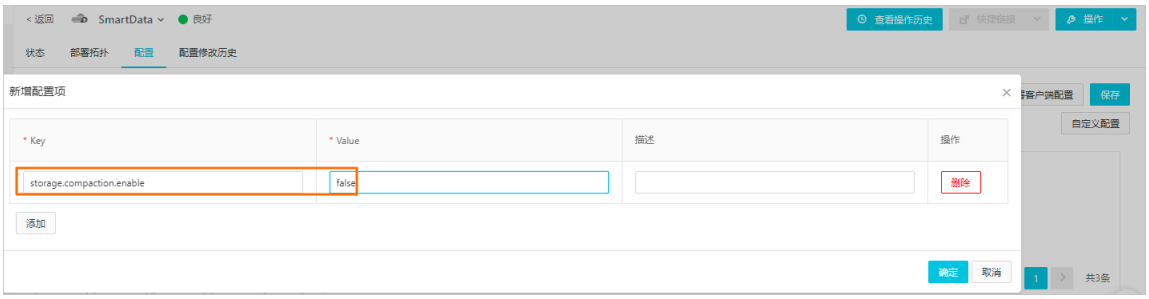
修复流程

常规修复

如果所在集群尚未发生该问题，则可以通过关闭小文件合并的优化路径，彻底避免该问题。

1. 在EMR控制台SmartData服务页面，添加自定义配置。
   i. 在SmartData服务的**storage**配置页，单击**自定义配置**。

   ii. 在**新增配置项**对话框中，添加Key为storage.compaction.enable，Value为false的配置项。



   iii. 单击**确定**。
2. 重启Jindo Storage Service。
   i. 在SmartData服务页面，选择**操作 > 重启Jindo Storage Service**。

   ii. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

紧急修复

对于已发生该问题的情况，请按以下步骤及时恢复业务，并进行缓存修复：

1. 该缺陷是缓存数据损坏导致的，通过关闭缓存可以消除缓存数据的影响。在SmartData服务的**client**配置页面通过以下配置关闭缓存。
   - 使用Block模式：添加自定义配置项。添加Key为jfs.data-cache.enable，Vlaue为false的配置项。
   - 使用Cache模式：修改参数jfs.cache.data-cache.enable的参数值为false。
2. 重跑相关作业。

   重跑作业后，作业能够恢复正常，如果仍有问题，则可能不是该文档所描述的错误原因，请另行排查问题原因或者提交工单处理。

   Presto、Impala和HBase等常驻服务的计算组件需重启各自的服务组件使上述配置生效。Hive和Spark等On YARN的计算组件直接重跑作业即可生效。
3. 缓存修复流程。
   - 使用Block模式：提交工单进行组件升级处理。
   - 使用Cache模式：

     > **说明** 因为已经关闭了缓存，所以以下操作不会影响业务。

     a. 在EMR控制台停止SmartData服务。

b. 将脚本format_cache.sh上传到集群的Master节点，并使用hadoop用户执行以下命令。

```
sh format_cache.sh
```

c. 在EMR控制台SmartData服务的**storage**配置页，添加Key为storage.compaction.enable，Value为false的配置项。

d. 在EMR控制台启动SmartData服务。

e. 重新打开缓存开关，修改参数jfs.cache.data-cache.enable的参数值为true。

1.2. 安全公告

1.2.1. 漏洞公告 | Apache Log4j2远程代码执行漏洞

近日，阿里云计算有限公司发现Apache Log4j2组件存在远程代码执行漏洞，并将漏洞情况告知Apache软件基金会。本文为您介绍该漏洞的影响范围及相应的修复方案。

漏洞影响

具体漏洞详情，请参见【漏洞通告】Apache Log4j2 远程代码执行漏洞（CVE-2021-44228/CVE-2021-45046）。

E-MapReduce（简称EMR）集群内受到影响的组件包括：Hive、Presto、Impala、Druid、Flink、Solr、Ranger、Storm、Oozie、Spark和Zeppelin，其中Spark和Zeppelin是由于依赖Hive组件而受到影响。

漏洞修复方案

您需要将EMR集群内包含漏洞的Log4j2 JAR包整体替换为2.17.0正式版本，同时修改Hive和Spark Log4j配置，禁用JNDI Lookup功能（`log4j2.formatMsgNoLookups=true`）。具体操作流程，请参见修复流程。

该修复方案：

- 适用于EMR-3.38.2及之前版本，EMR-5.4.2及之前版本，EMR-4.x版本。EMR-3.38.3及之后版和EMR-5.4.3及之后版本已修复该问题，无需进行修复操作。
- 执行该修复方案后，需要重启对应的组件。
- 该漏洞修复脚本不会对线上业务造成影响，但由于必须重启对应的组件才能生效，所以建议业务低峰期时执行。

修复流程

1. 单击patches-log4j.tar.gz下载patch包。
2. 登录EMR集群的Master节点，并将步骤一中下载的patch包放在hadoop用户的HOME目录下。
3. 将patch包解压缩后，使用hadoop用户执行操作。

   ```
   su hadoop
   tar zxf patches-log4j.tar.gz
   ```

4. 编辑patch包下的hosts文件，添加集群所有节点的host name，如emr-header-1或emr-worker-1，文件内容以行分割。

   ```
   cd patches
   vim hosts
   ```

   hosts文件内容示例如下：

   ```
   emr-header-1
   emr-worker-1
   emr-worker-2
   ```

5. 通过*fix.sh*脚本执行修复操作。

   ```
   ./fix.sh
   ```

   脚本执行完成后，返回如下提示信息。

   ```
   ### NOTICE: YOU CAN RESTORE THIS PATCH BY RUN RESTORE SCRIPT ABOVE
   $> sh ./restore.sh 20211213001755
   ### DONE
   ```

   如果您需要将上述修复过程进行回滚，可以执行以下命令。

   ```
   ./restore.sh 20211213001755
   ```

   > **说明** 对于已经在运行的YARN作业（Application，例如，Spark Streaming或Flink作业），需要停止作业后，批量滚动重启YARN NodeManager。

6. 重启服务。

Hive、Presto、Impala、Druid、Flink、Solr、Ranger、Storm、Oozie、Spark和Zeppelin等组件需要重启之后才能完全修复漏洞。

以Hive组件为例，在EMR集群的Hive服务页面，选择右上角的**操作 > 重启All Components**。

由于该方案依赖于SSH免密登录，对于Gateway集群，您需要手动将patch包上传到Gateway集群的每一个节点，并按EMR集群的修复流程分别执行修复操作。

> 注意 patch包中的hosts文件只需要填写当前执行节点的hostname。

新建EMR集群时在EMR控制台添加引导操作，或扩容已有集群时可以自动修复漏洞。具体操作步骤如下：

1. 单击patches-log4j.tar.gz和bootstrap_log4j.sh，下载patch包和bootstrap脚本，并且上传到OSS上。

   例如，上传到OSS的路径为*oss://<bucket-name>/path/to/patches-log4j.tar.gz*和*oss://<bucket-name>/path/to/bootstrap_log4j.sh*。
2. 在EMR控制台添加引导操作，详细信息请参见管理引导操作。

   在**添加引导操作**对话框中，填写配置项。

参数	描述
名称	引导操作的名称。例如，log4j漏洞修复。
脚本位置	选择脚本所在OSS的位置。 脚本路径格式必须为*oss://**/*.sh*格式。本文示例为*oss://<bucket-name>/path/to/bootstrap_log4j.sh*。
参数	引导操作脚本的参数，指定脚本中所引用的变量的值。 本文示例为*oss://<bucket-name>/path/to/patches-log4j.tar.gz* 。
执行范围	选择**集群**。
执行时间	选择**组件启动后**。
执行失败策略	选择**继续执行**。

3. 如果是新建集群，则需要重启Hive、Presto、Impala、Druid、Flink、Solr、Ranger、Storm、Oozie、Spark和Zeppelin等组件。

Gateway集群

新建集群和扩容已有集群

1.3. 文档修订记录

本文为您介绍E-MapReduce文档的最新动态，包括新增功能及功能变更。

E-MapReduce的版本更新和重要功能发布记录请参见版本概述。

2021年12月更新记录

特性	类别	描述	产品文档
SmartData	新增功能	介绍SmartData（3.8.x）版本更新的内容。	SmartData 3.8.x版本简介
SmartData	更新功能	介绍如何使用Raft-RocksDB-OTS作为Jindo元数据服务（Namespace Service）的存储后端。	使用Raft-RocksDB-Tablestore作为存储后端
SmartData	新增功能	介绍如何使用Credential Provider。	JindoFS OSS Credential Provider使用说明
Ranger	新增功能	介绍如何将OSS集成到Ranger，以及如何配置权限。	JindoFS OSS配置
Presto	新增功能	介绍创建独立的Presto集群后，如何配置连接器和数据湖元数据，以使用独立的Presto集群。	使用独立的Presto集群
EMR on ACK	新增功能	介绍其他阿里云账号（主账号）或RAM用户（子账号）如何访问当前集群的Web UI。	访问Web UI

2021年11月更新记录

时间	特性	类别	描述	产品文档
2021.11.08	技术支持	更新功能	介绍EMR产品技术支持服务的范围、支持方式和服务SLA。	技术支持
2021.11.17	DLF-Auth	新增功能	介绍如何开启DLF-Auth权限。	DLF-Auth
2021.11.25	产品简介	更新功能	介绍EMR-5.4.1和EMR-5.4.0发行版本的发布日期和更新内容等信息。	EMR-5.4.x版本说明
2021.11.25	产品简介	更新功能	介绍EMR-3.38.1和EMR-3.38.0发行版本的发布日期和更新内容等信息。	EMR-3.38.x版本说明

2021年10月更新记录

时间	特性	类别	描述	产品文档
2021.10.20	产品简介	新增功能	介绍EMR-5.4.0发行版本的发布日期和更新内容等信息。	EMR-5.4.x版本说明
2021.10.20	产品简介	新增功能	介绍EMR-3.38.0发行版本的发布日期和更新内容等信息。	EMR-3.38.x版本说明
2021.10.22	Presto	新增功能	新增Iceberg连接器，使用Iceberg连接器可以用来查询Iceberg格式的数据文件。	Iceberg连接器
2021.10.27	ClickHouse	新增功能	新增从OSS导入数据至ClickHouse。	从OSS导入数据至ClickHouse

2021年9月更新记录

时间	特性	类别	描述	产品文档
2021.09.02	Delta Lake	新增功能	介绍G-SCD的具体解决方案及如何通过G-SCD处理维度的数据。	场景四：Slowly Changing Dimension
2021.09.15	产品简介	新增功能	介绍EMR-5.3.x发行版本的发布日期和更新内容等信息。	EMR-5.3.x版本说明
2021.09.15	产品简介	新增功能	介绍EMR-3.37.x发行版本的发布日期和更新内容等信息。	EMR-3.37.x版本说明

2021年8月更新记录

时间	特性	类别	描述	产品文档
2021.08.12	Flink（VVR）	新增功能	介绍如何在E-MapReduce上提交Flink作业以及查看作业。	基础使用
2021.08.16	Kerberos	新增功能	介绍如何在EMR上对接外部的MIT Kerberos。	EMR对接外部的MIT Kerberos
2021.08.25	EMR Studio	新增功能	介绍EMR Studio的特性。	EMR Studio概述
2021.08.31	EMR Studio	新增功能	介绍如何快速创建EMR Studio集群并开展交互式开发和工作流调度工作。	快速入门

2021年7月更新记录

时间	特性	类别	描述	产品文档
2021.07.15	HDFS	新增功能	介绍HDFS的高阶使用和开发指南。	• 开启权限认证 • 开发指南
2021.07.22	Flume	新增功能	介绍Flume的高阶使用和开发指南。	• 高阶使用 • 安装第三方插件

2021年6月更新记录

时间	特性	类别	描述	产品文档
2021.06.21	SmartData	新增功能	介绍SmartData 3.6.x版本的更新内容。	SmartData 3.6.x版本简介
2021.06.21	产品简介	新增功能	介绍EMR-3.36.x发行版本的发布日期和更新内容等信息。	EMR-3.36.x版本说明
2021.06.21	产品简介	新增功能	介绍EMR-5.2.x发行版本的发布日期和更新内容等信息。	EMR-5.2.x版本说明

2021年4月更新记录

时间	特性	类别	描述	产品文档
2021.04.20	SmartData	新增功能	介绍SmartData 3.5.x版本的更新内容。	SmartData 3.5.x版本简介
2021.04.20	产品简介	新增功能	介绍EMR-3.35.x发行版本的发布日期和更新内容等信息。	EMR-3.35.x版本说明
2021.04.20	产品简介	新增功能	介绍EMR-4.9.x发行版本的发布日期和更新内容等信息。	EMR-4.9.x版本说明

2021年3月更新记录

时间	特性	类别	描述	产品文档
2021.03.15	SmartData	新增功能	介绍SmartData 3.4.x版本的更新内容。	SmartData 3.4.x版本简介
2021.03.15	产品简介	新增功能	介绍EMR-3.34.x发行版本的发布日期和更新内容等信息。	EMR-3.34.x版本说明
2021.03.15	产品简介	新增功能	介绍EMR-4.8.x发行版本的发布日期和更新内容等信息。	EMR-4.8.x版本说明

2021年1月更新记录

时间	特性	类别	描述	产品文档
2021.01.15	SmartData	新增功能	介绍SmartData 3.2.x版本的更新内容。	SmartData 3.2.x版本简介
2021.01.15	产品简介	新增功能	介绍EMR-3.33.x发行版本的发布日期和更新内容等信息。	EMR-3.33.x版本说明
2021.01.15	产品简介	新增功能	介绍EMR-4.6.x发行版本的发布日期和更新内容等信息。	EMR-4.6.x版本说明

2020年12月更新记录

时间	特性	类别	描述	产品文档
2020.12.02	集群类型	新增功能	Alluxio是一个面向基于云的数据分析和人工智能的开源的数据编排技术。	概述
2020.12.02	集群类型	新增功能	Hudi是一种数据湖的存储格式，在Hadoop文件系统之上提供了更新数据和删除数据的能力以及流式消费变化数据的能力。	Hudi概述
2020.12.14	集群类型	新增功能	ESS（EMR Remote Shuffle Service）是EMR在优化计算引擎的Shuffle操作上，推出的扩展组件。	ESS
2020.12.14	新增功能	新增功能	介绍EMR-4.5.0发行版本的发布日期和更新内容等信息。	EMR-4.5.x版本说明

2020年11月更新记录

时间	特性	类别	描述	产品文档
2020.11.25	产品简介	新增功能	介绍EMR-3.32.x发行版本的发布日期和更新内容等信息。	EMR-3.32.x版本说明
2020.11.25	SmartData	新增功能	介绍SmartData 3.1.x版本的更新内容。	SmartData 3.1.x版本简介
2020.11.25	SmartData	新增功能	介绍如何改写Jindo HDFS客户端级别的路径。	改写Jindo HDFS客户端路径
2020.11.25	SmartData	新增功能	介绍JindoFS块存储模式的数据管理策略。	数据管理策略
2020.11.25	SmartData	新增功能	介绍如何使用JindoTable表或分区的访问热度收集功能。	JindoTable表或分区的访问热度收集
2020.11.25	SmartData	更新功能	介绍SmartData 3.1.0版本如何使用Credential Provider。	JindoFS OSS Credential Provider使用说明
2020.11.25	SmartData	更新功能	SmartData 3.1.0版本支持Flink可恢复性写入JindoFS或OSS。	支持Flink可恢复性写入JindoFS或OSS

2020年10月更新记录

时间	特性	类别	描述	产品文档
2020.10.28	SmartData	新增功能	介绍SmartData 3.0.0版本的更新内容。	SmartData 3.0.x版本简介
2020.10.28	SmartData	新增功能	介绍使用SmartData时的常见问题。	SmartData常见问题

2020年9月更新记录

时间	特性	类别	描述	产品文档
2020.09.1	新增功能	最佳实践	介绍如何使用阿里云E-MapReduce创建的Hadoop和Kafka集群，运行PyFlink作业以消费Kafka数据。	通过PyFlink作业处理Kafka数据
2020.09.4	集群类型	新增功能	介绍如何使用Alink调度作业。	Alink调度
2020.09.7	JindoFS	新增功能	介绍如何使用Jindo AuditLog。	Jindo AuditLog使用说明

2020年8月更新记录

时间	特性	类别	描述	产品文档
2020.08.11	集群类型	新增功能	介绍如何使用AutoML。	AutoML

2020年7月更新记录

时间	特性	类别	描述	产品文档
2020.07.24	新增功能	新增功能	介绍如何配置Impala SQL作业。	Impala SQL作业配置
2020.07.13	数据开发	优化	介绍如何配置Flink（VVR）类型的作业。	Flink（VVR）作业配置
2020.07.09	开发指南	体验优化	更新spark.hadoop.fs.oss.impl参数的默认值。	• 参数说明 • 参数说明
2020.07.02	OpenAPI	新接口	介绍如何使用API进行EMR集群节点缩容。	集群缩容

2020年6月更新记录

时间	特性	类别	描述	产品文档
2020.06.17	新增功能	新增功能	介绍如何在E-MapReduce控制台上配置及使用Alink、Faiss-Server和TensorFlow。	• 概述 • PAI-Alink • Faiss-Server • TensorFlow

2.快速入门

本文为您介绍如何通过阿里云账号登录E-MapReduce控制台，使用一键购买，快速创建一个Hadoop集群并执行作业。

前提条件

- 注册阿里云账号，并完成实名认证。具体操作请参见阿里云账号注册流程。

  > **说明** 根据阿里云ECS的规则，您在购买按量付费实例时，需要保证阿里云账户中可用余额（含现金、代金券、优惠券等）不得少于100元人民币。充值方式请参见前往充值。

- 完成对E-MapReduce的服务账号授予默认的EMR和ECS角色权限，详情请参见角色授权。

操作流程

1. 步骤一：创建集群

   通过一键购买，快速创建一个Hadoop集群。

2. 步骤二：创建并执行作业

   Hadoop集群创建成功后，您可以通过控制台方式或者命令行方式创建并执行Spark类型作业。

3. 步骤三：查看作业运行记录

   提交作业后，您可以通过数据开发控制台方式和YARN UI方式查看作业运行记录。

4. （可选）步骤四：释放集群

   如果不再使用该集群，可以释放集群以节约成本。

步骤一：创建集群

1. 进入集群管理页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域（Region）和资源组。
   iii. 单击上方的**集群管理**页签。
2. 在**集群管理**页面，单击右上角的**一键购买**。
3. 在创建集群页面，完成集群相关配置。

   本文使用的配置如下表。通过自定义购买方式创建集群的配置信息，请参见创建集群。

配置区域	配置项	示例	说明
基础信息	**付费类型**	按量付费	在测试场景下，建议使用**按量付费**，测试正常后可以释放该集群，再新建一个**包年包月**的生产集群正式使用。
	集群类型	Hadoop	默认集群类型。
	产品版本	EMR-5.3.1	当前最新的软件版本。
	集群名称	Emr-Hadoop	集群的名字，长度限制为1~64个字符，仅可使用中文、字母、数字、短划线（-）和下划线（_）。
	登录密码	自定义密码	请记录该配置，登录集群时您需要输入该密码。 **说明** 使用自定义购买方式创建集群时，建议选择密钥对方式。该方式具有安全性和便捷性优势，密钥对使用方式，请参见SSH密钥对方式。
网络配置	**可用区**	华东1（杭州） 可用区 I	集群创建后，无法直接更改地域和可用区，请谨慎选择。
	网络类型	专有网络	默认为专有网络，不可更改。
	VPC	VPC_Hangzhou(192.168.xx.xx/16)(ID：vpc-bp1f4epmkvncimpgs****)	选择对应区域下的VPC。如果没有，单击**创建 VPC / 子网（交换机）**前往新建。创建专有网络完成后，单击**刷新**，可以选择刚创建好的VPC。 **说明** 创建VPC/子网（交换机）的步骤，请参见创建专有网络和交换机。请确保交换机可用区信息与集群可用区信息一致。

配置区域	配置项	示例	说明
	交换机	vsw_i(192.168.xx.xx/24)(ID：vsw-bp1e2f5fhaplp0g6p****)	选择在对应VPC下可用区的交换机，如果在这个可用区没有可用的交换机，则需要新创建一个。
	安全组名称	sg-bp1ddw7sm2risw****(ID:sg-bp1ddw7sm2riswb1****)	注意 禁止使用ECS上创建的企业安全组。 您可直接输入安全组名称来新建一个安全组。如果已有在使用的安全组，则可以直接选择使用。
高可用	**高可用**	不开启。	默认不开启。打开**高可用**开关后，Hadoop集群会有两个或三个Master节点来支持ResourceManager和NameNode的高可用。
实例	**选型配置**	使用默认值。	根据业务诉求，配置**Master实例**、**Core实例**和**Task实例**信息。详情请参见选型配置说明。

4. 选中**E-MapReduce服务条款**，单击**创建**。

   集群创建后，您看到的集群**状态**为**初始化中**，单击**初始化中**右侧的图标，可以查看操作流程。通过刷新页面可以查看当前集群**状态**，当集群**状态**显示为**空闲**时，集群创建成功。

步骤二：创建并执行作业

集群创建成功后，您可以在该集群创建并执行作业。本文为您提供两种方式创建并执行作业，分别是控制台方式和命令行方式。您可以根据业务诉求选择以下任意一种方式完成作业的创建和执行。

1. 创建项目。
   i. 单击上方的**数据开发**页签。
   ii. 在**项目列表**页面，单击**新建项目**。
   iii. 在**新建项目**对话框中，配置相关信息。

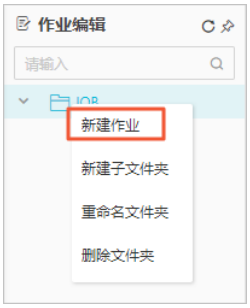
配置项	示例	描述
项目名称	project-dev	创建项目的名称。
项目描述	测试项目	创建项目的描述信息。
资源组选择	default resource group	从**资源组选择**列表中，选择已有的资源组。

   iv. 单击**创建**。

   在**项目列表**页面，可查看或者操作新增的项目。

2. 在目标项目创建作业。
   i. 在**项目列表**页面，单击目标项目所在行的**作业编辑**。

   ii. 在**作业编辑**区域，在需要操作的文件夹上单击右键，选择**新建作业**。



   说明 您还可以通过在文件夹上单击右键，进行创建子文件夹、重命名文件夹和删除文件夹操作。

iii. 在**新建作业**对话框中，配置相关信息。

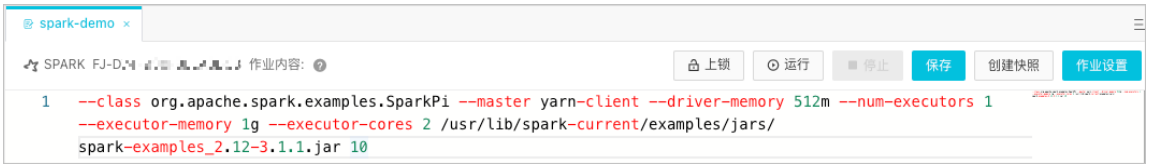
配置项	示例	描述
作业名称	spark-demo	建议使用有意义的名称，便于后续管理作业。
作业描述	spark demo作业	建议使用有意义的描述，便于后续管理作业。
作业类型	Spark	支持多种作业类型，本文以Spark为例。

3. 编辑作业内容。

i. 在**作业内容**区域，填写提交Spark作业需要提供的命令行信息。

本文以Spark 3.1.1版本为例，输入的作业内容如下。

```
--class org.apache.spark.examples.SparkPi --master yarn-client --driver-memory 512m --num-executors 1 --executor-me
mory 1g --executor-cores 2 /usr/lib/spark-current/examples/jars/spark-examples_2.12-3.1.1.jar 10
```



> **说明** *spark-examples_2.12-3.1.1.jar*为您集群中对应的JAR包名称，您可以登录集群，在*/usr/lib/spark-current/examples/jars*路径下获取。

ii. 单击右上角**保存**。

4. 运行作业。

i. 在新建的作业页面，单击右上方的**运行**来执行作业。

ii. 在**运行作业**对话框中，选择资源组和执行集群。

iii. 单击**确定**。

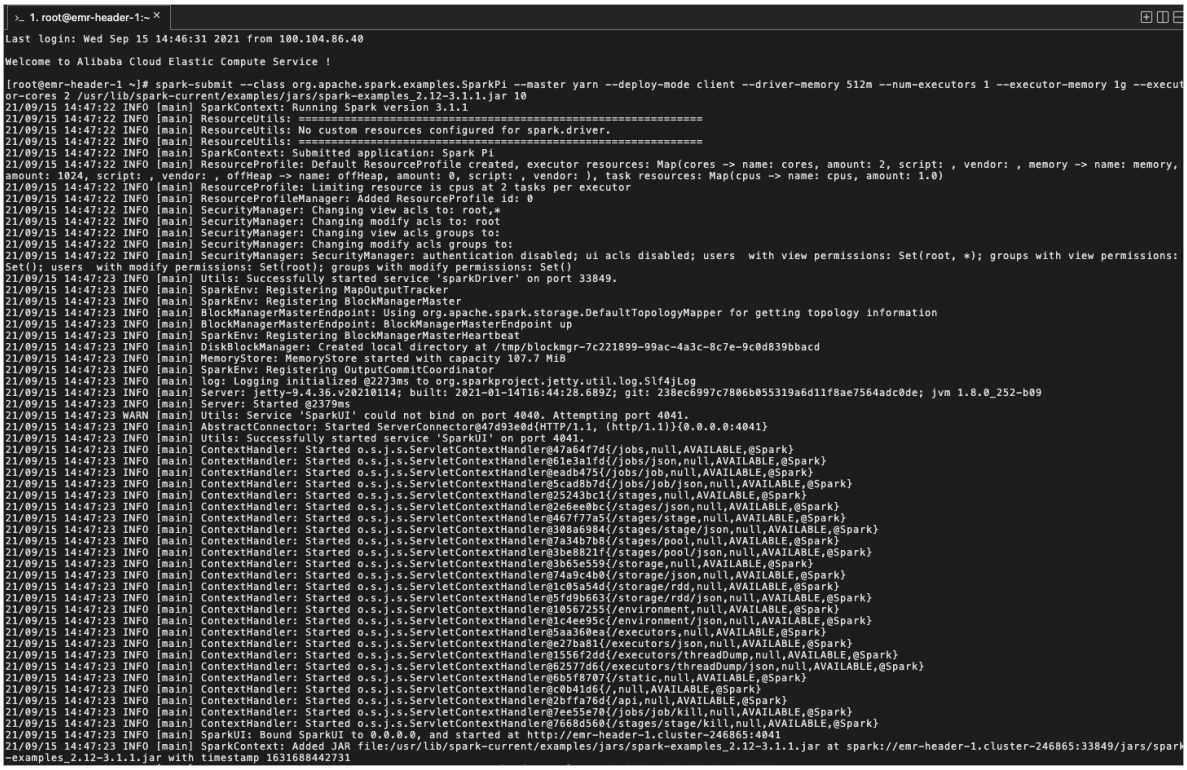
1. 通过SSH方式连接集群，详情请参见登录集群。
2. 在命令行执行以下命令，提交并运行作业。

本文以Spark 3.1.1版本为例，输入的命令示例如下。

```
spark-submit --class org.apache.spark.examples.SparkPi --master yarn --deploy-mode client --driver-memory 512m --num-ex
ecutors 1 --executor-memory 1g --executor-cores 2 /usr/lib/spark-current/examples/jars/spark-examples_2.12-3.1.1.jar 10
```

> **说明** *spark-examples_2.12-3.1.1.jar*为您集群中对应的JAR包名称，您可以登录集群，在*/usr/lib/spark-current/examples/jars*路径下获取。

作业提交后，执行信息如下。



YARN UI方式

步骤三：查看作业运行记录

提交作业后，您可以通过数据开发控制台方式和YARN UI方式查看作业运行记录。

该方式适用于通过控制台方式创建并提交作业的场景。

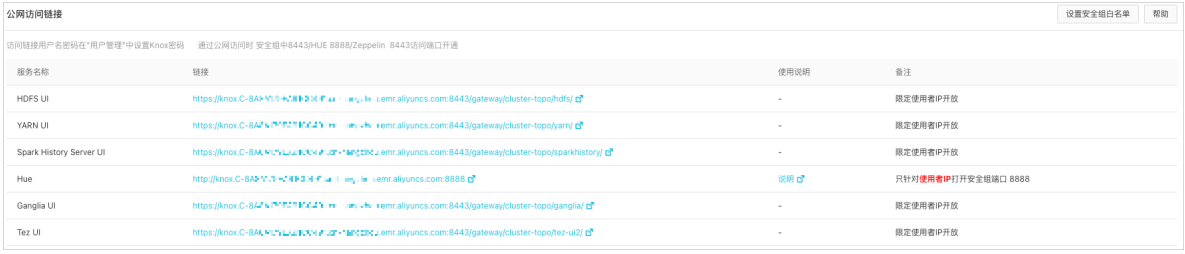
1. 作业运行后，您可以在**日志**页签中查看作业运行的日志。
2. 单击**运行记录**页签，可以查看作业实例的运行情况。

3. 单击目标运行记录右侧的**详情**，跳转到**运维中心**，可以查看作业实例信息、提交日志和YARN容器列表。

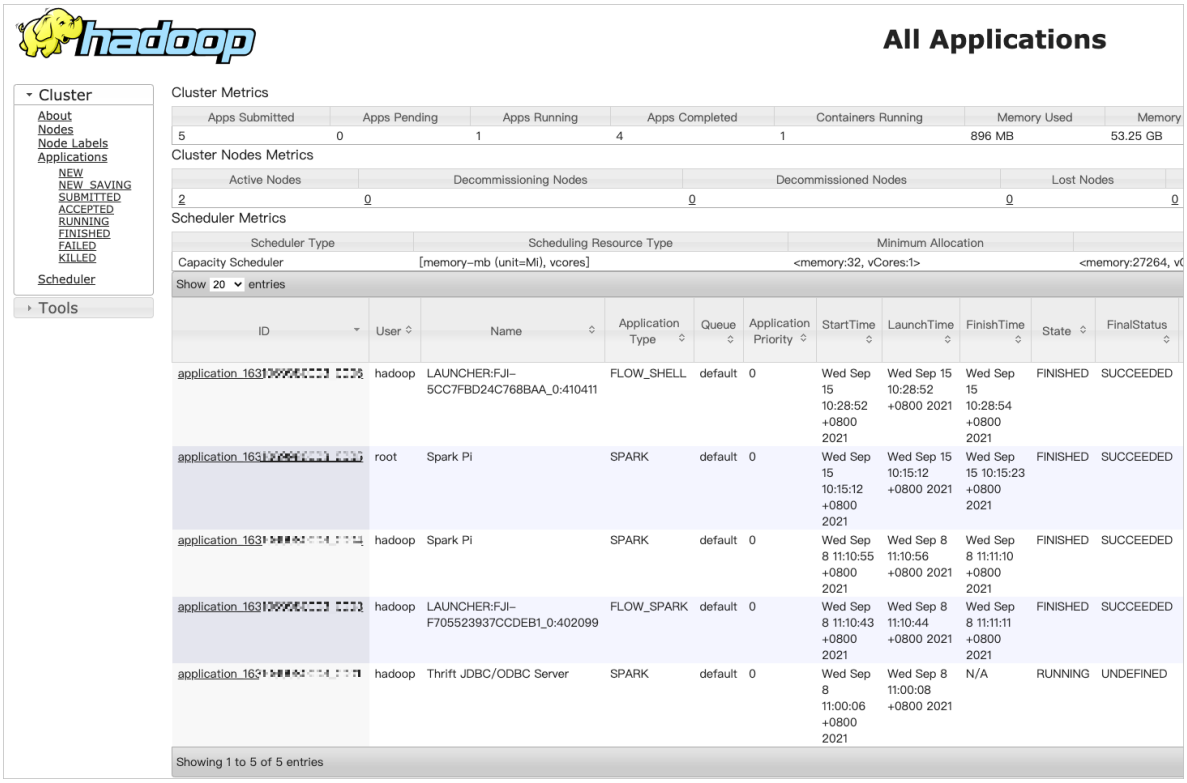
该方式适用于通过控制台方式和命令行方式创建并提交作业的场景。

1. 设置安全组访问，详情请参见设置安全组访问。
2. 在目标集群的**集群管理**页签下，单击左侧菜单**访问链接与端口**。
3. 单击**YARN UI**后面的链接。

   在使用Knox账号访问YARN UI页面时，需要Knox账号的用户名和密码，详情请参见管理用户。



4. 在Hadoop控制台，单击目标作业的ID，可以查看作业运行的详情。



（可选）步骤四：释放集群

如果您创建的集群不再使用时，可以释放集群节约成本。确认集群释放后，系统会对集群进行如下处理：

1. 强制终止集群上的所有作业。
2. 终止并释放所有的ECS实例。

这个过程所需时间取决于集群的大小，集群越小释放越快。通常在几秒内可以完成释放，至多不会超过5分钟。

> 注意
> - 按量付费的集群可以随时释放，包年包月的集群到期后才能释放。
> - 释放集群前，请确保集群状态是初始化中、运行中或空闲。

1. 在**作业编辑**页面，选择右上角**返回EMR控制台 > 集群管理**。
2. 在**集群管理**页面，选择目标集群所在行的**更多 > 释放**。
   您还可以单击目标集群所在行的**详情**，然后在**集群基础信息**页面，选择右上角**实例状态管理 > 释放**。
3. 在**集群管理-释放**对话框中，单击**释放**。

相关文档

- 了解如何登录集群：登录集群。
- 了解如何查看常用文件的安装路径：常用文件路径。
- 了解如何使用API管理集群和服务等：API概览。

常见问题

了解使用阿里云E-MapReduce的常见问题：常见问题。

3.运维管理指南（新版控制台）
3.1. 集群管理
3.1.1. 集群类型
3.1.1.1. 数据湖集群

E-MapReduce（简称EMR）新版控制台提供了数据湖集群，一个更灵活、可靠，以及高效的大数据计算集群。同时，您可以基于该集群轻松构建一个可扩展的数据管道。本文为您介绍数据湖集群支持的特性，以及与旧版Hadoop集群之间的差异。

数据湖集群特性介绍

可靠性

如果您启用了集群的高可用服务，则EMR将把3台Master节点分布在底层不同的硬件上以降低故障风险。另外，考虑到Master节点在损坏情况下的修复时间，数据湖集群将不再支持2台Master模式。同时在高可用集群模式下，EMR将不再支持本地MySQL（单机部署，非高可用） 作为Hive metastore数据库，仅支持DLF和外部RDS两种方式。

数据湖集群的创建以及扩容的过程中，会考虑ECS的健康状况，从而避免异常ECS加入集群；同时EMR管理器会自动识别在集群后期运行过程中出现的磁盘损坏和长时间不可读写等极端情况的问题，并启动补偿机制。补偿机制通常会优先在节点组中新增节点，随后会释放异常节点。

> 注意　补偿的过程会发生一部分额外的费用。

灵活性

数据湖集群的所有服务将调整为可选，允许您根据实际需求来规划集群应用，您甚至可以只勾选HDFS从而拥有一个独立的分布式存储系统的集群，或者只勾选Presto从而拥有一个独立的Ad-Hoc集群。

由于数据湖集群默认支持Private Zone，所以集群内部节点的访问不再依赖hosts文件，从而避免因依赖hosts文件引发的问题。

附加安全组与挂载公网

Hadoop集群默认有一个安全组，他简单并易于使用，但无法精细化管理一个集群对外开放的端口。例如，一个集群需要对外开放HDFS的50070端口，如果此时增加一个端口入规则，则该规则会对集群所有节点有效，但实际上Core或Task并不需要开放该端口。因此数据湖集群对每一个节点组增加了最多两个附加安全组，从而可以精细化控制节点组出入端口的规则。

另外，EMR数据湖集群将挂载公网特性移到了节点组，这样可以更好的搭配附加安全组实现精细化的对外开放公网的管理。

Spark部署

数据湖集群在原有Hadoop2 + Spark2和Hadoop3 + Spark3的基础上，增加了Hadoop2 + Spark3和Hadoop3 + Spark2的组合部署模式。您可以根据实际情况，选择满足自己的软件组合。另外，数据湖集群将支持Kyuubi，一个企业级的数据湖计算引擎Gateway，为SparkSQL提供标准化的JDBC接口，可实现多租户和多种计算资源管理方式。

域名支持

数据湖集群优化了节点的域名，从原来的`emr-header-1.cluster-13***`格式调整为`master-1-1.c-494bea2977d9***`格式。

如果是高可用集群，则取值master-1-*{1-3}*并加上集群ID作为后缀组成hostname。同时，机器名显示为emr-user@master-1-1(*{IP}*)，这意味着您可以在终端上看到当前节点的IP地址，从而进一步方便您平时的运维工作。

登录用户以及私钥对

通过私钥对的方式登录集群时，其默认用户名从root调整为emr-user，这样做的出发点在于更推荐在平时节点运维时使用emr-user用户。如果您仍然想使用root用户，可以在登录集群后通过sudo命令切换为root用户。

使用私钥对默认可以登录数据湖集群所有节点而不仅是Master节点。如果您仍使用密码方式登录集群，则可以继续使用root用户。

新增`emr-metadata`命令

数据湖集群的每个节点默认增加一个可执行命令`emr-metadata`，该命令将输出当前节点关于集群的相关元数据信息，例如节点的角色、instanceId，以及网络与硬件配置等，这可以配合您在使用引导脚本的过程中所依赖的本地节点信息。

与Hadoop集群的主要差异

对比项	数据湖集群	Hadoop集群
集群创建时间	平均时间小于5分钟	平均时间小于10分钟
集群节点组新增节点	平均时间小于3.5分钟	平均时间小于10分钟
集群节点组弹性伸缩	暂不支持	支持

对比项	数据湖集群	Hadoop集群
开放API	暂不支持	支持
新增服务	暂不支持	支持
域名支持	Private Zone	hosts地址映射
附加安全组与挂载公网	节点组支持附加安全组以及挂载公网选项	挂载公网为集群选项
可靠性	支持Task节点组异常节点容错与补偿	不支持
高可用	不再支持本地MySQL作为Hive元数据库	支持本地MySQL作为Hive元数据库
	支持部署集，3台Master分布在不同底层硬件以降低硬件风险	不支持部署集
	NameNode与Resource Manager部署于3节点，并不再支持2 Master模式	Namenode与Resource Manager仅部署于2节点，支持2 Master模式
集群服务	支持可选	必选 + 可选
Spark2与Hadoop3组合	支持	不支持
Spark3与Hadoop2组合	支持	EMR-3.38.0之后版本支持同时部署

已知限制

数据湖集群暂不支持的服务有RSS、Kerberos、Kudu、Hue、Superset、Livy和HBase。

3.1.2. 集群规划

3.1.2.1. 选型配置说明

选择合适的集群是E-MapReduce产品使用的第一步。E-MapReduce配置选型不仅要考虑企业大数据使用场景、估算数据量、服务可靠性要求，还应该考虑企业预算。

大数据使用场景

E-MapReduce产品当前主要满足企业的以下大数据场景：

- 批处理场景

  该场景对磁盘吞吐和网络吞吐要求高，处理的数据量也大，但对数据处理的实时性要求不高，您可以选用MapReduce、Pig和Spark组件。该场景对内存要求不高，选型时您需要重点关注作业对CPU和内存的需求，以及Shuffle对网络的需求。

- Ad-Hoc查询

  数据科学家或数据分析师使用即席查询工具检索数据。该场景对查询实时性、磁盘吞吐和网络吞吐要求高，您可以选用E-MapReduce的Impala和Presto组件。该场景对内存要求高，选型时需要考虑数据和并发查询的数量。

- 流式计算、高网络吞吐和计算密集型场景

  您可以选用E-MapReduce的Flink、Spark Streaming和Storm组件。

- 消息队列

  该场景对磁盘吞吐和网络吞吐要求高，并且内存消耗大，存储不依赖于HDFS。您可以选用E-MapReduce的Kafka集群。

- 数据冷备场景

  该场景对计算和磁盘吞吐要求不高，但要求冷备成本低，推荐使用JindoFS将阿里云OSS的归档型和深度归档型作为冷数据存储，以降低存储成本。

E-MapReduce节点

E-MapReduce节点有主实例（Master）、核心实例（Core）和计算实例（Task）三种实例类型。详情请参见实例类型。

E-MapReduce存储可以采用高效云盘、本地盘、SSD云盘和SSD本地盘。磁盘性能为SSD本地盘 > SSD云盘 > 本地盘 > 高效云盘。

E-MapReduce底层存储支持OSS（仅标准型OSS）和HDFS。相对于HDFS，OSS的数据可用性更高（99.99999999%），HDFS的数据可用性由云盘或本地盘存储的可靠性来保证。归档数据和深度归档数据需要解冻为标准型存储才能参与EMR引擎计算。

存储价格估算如下：

- 本地盘实例存储为0.04 元/GB/月
- OSS标准型存储为0.12 元/GB/月
- OSS归档型存储为0.033 元/GB/月

- OSS深度归档型存储为0.015 元/GB/月
- 高效云盘存储为0.35 元/GB/月
- SSD云盘存储为1.00 元/GB/月

E-MapReduce选型

- Master节点选型
  - Master节点主要部署Hadoop的Master进程。例如，NameNode和ResourceManager等。
  - 生产集群建议打开高可用HA，E-MapReduce的HDFS、YARN、Hive和HBase等组件均已实现HA。生产集群建议在创建集群的**硬件配置**步骤开启高可用。如果购买时未开启高可用，在后续使用过程中无法开启高可用功能。
  - Master节点主要用来存储HDFS元数据和组件Log文件，属于计算密集型，对磁盘IO要求不高。HDFS元数据存储在内存中，建议根据文件数量选择16 GB以上内存空间。
- Core节点选型
  - Core节点主要用来存储数据和执行计算，运行DataNode和Nodemanager。
  - HDFS数据量大于60 TB，建议采用本地盘实例（ECS.d1，ECS.d1NE），本地盘的磁盘容量为（CPU核数/2）*5.5TB*实例数量。

    例如，购买4台8核D1实例，磁盘容量为8/2*5.5*4 台=88 TB。因为HDFS采用3备份，所以本地盘实例最少购买3台，考虑到数据可靠性和磁盘损坏因素，建议最少购买4台。
  - HDFS数据量小于60 TB，可以考虑高效云盘和SSD云盘。
- Task节点选型

  Task节点主要用来补充Core节点CPU和内存计算能力的不足，节点并不存储数据，也不运行DataNode。您可以根据CPU和内存需求来估算实例个数。

E-MapReduce生命周期

E-MapReduce支持弹性扩展，可以快速的扩容，灵活调整集群节点配置，详情请参见扩容，或者升降配ECS节点，详情请参见升降配。

可用区选择

为保证效率，您应该部署E-MapReduce与业务系统在同一地域的同一个可用区。详情请参见地域和可用区。

3.1.2.2. 实例类型

E-MapReduce集群由多个不同类型的实例节点组成，包括主实例节点（Master）、核心实例节点（Core）和计算实例节点（Task）。

不同实例节点上部署的服务进程不同，负责完成的任务也不同。例如：

- 主实例节点（Master）：部署Hadoop HDFS的NameNode服务、Hadoop YARN的ResourceManager服务。
- 核心实例节点（Core）：部署DataNode服务、Hadoop YARN的NodeManager服务。
- 计算实例节点（Task）：只进行计算，部署Hadoop YARN的NodeManager服务，不部署任何HDFS相关的服务。

创建集群时，您需要确定对应的三种实例类型的ECS规格，相同实例类型的ECS在同一个实例组内。创建集群完成后，您可以通过扩容来增加实例组内的机器数量（主实例组除外）。

> **说明** EMR-3.2.0及后续版本支持计算实例节点（Task）。

主实例节点（Master）

主实例节点是集群服务部署管控等组件的节点，例如，Hadoop YARN的 ResourceManager。

当您需要查看集群上服务的运行情况时，您可以通过软件的Web UI来查看。当您需要快速测试或者运行作业时，您可以登录主实例节点，然后通过命令行直接提交作业。登录主节点的具体步骤请参见登录集群。

核心实例节点（Core）

核心实例节点是被主实例节点管理的节点。核心实例节点上会运行Hadoop HDFS的Datanode服务，并保存所有的数据。同时，核心实例节点也会部署计算服务来执行计算任务。例如，Hadoop YARN的NodeManager服务。

为满足存储数据量或计算量扩展的需求，核心实例节点支持随时扩容，并且扩容过程中不会影响当前集群的正常运行。核心实例节点可以使用多种不同的存储介质来保存数据，详情请参见本地盘和块存储。

计算实例节点（Task）

计算实例节点是专门负责计算的实例节点，不会保存HDFS数据，也不会运行Hadoop HDFS的Datanode服务，是一个可选的实例类型。如果核心实例的计算能力充足，则可以不使用计算实例。当集群计算能力不足时，您可以随时通过计算实例节点快速给集群增加额外的计算能力，例如Hadoop的MapReduce任务和Spark Executors等。

计算实例节点可以随时新增和减少，并且不会影响现有集群的运行。

3.1.2.3. Gateway实例说明

创建Gateway集群时必须关联到一个已经存在的集群。Gateway集群可以作为一个独立的作业提交点，以便您更好的对关联集群进行操作。

Gateway集群通常是一个独立的集群，由多台相同配置的节点组成，集群上会部署Hadoop（HDFS+YARN）、Hive、Spark和Sqoop等客户端。

未创建Gateway集群时，Hadoop等集群的作业是在本集群的Master或Core节点上提交的，会占用本集群的资源。创建Gateway集群后，您可以通过Gateway集群来提交其关联的集群的作业，这样既不会占用关联集群的资源，又可以提高关联集群Master或Core节点的稳定性，尤其是Master节点。

每一个Gateway集群均支持独立的环境配置。例如，在多个部门共用一个集群的场景下，您可以为这个集群创建多个Gateway集群，以满足不同部门的业务需求。

3.1.2.4. ECS实例说明

本文介绍E-MapReduce（简称EMR）支持的ECS实例类型，以及各实例类型适用的场景。

EMR支持的ECS实例类型

- 通用型

  vCPU：Memory = 1：4。例如，8核32 GiB，使用云盘作为存储。
- 计算型

  vCPU：Memory = 1：2。例如，8核16 GiB，使用云盘作为存储，提供了更多的计算资源。
- 内存型

  vCPU：Memory = 1：8。例如， 8核64 GiB，使用云盘作为存储，提供了更多的内存资源。
- 大数据型

  使用本地SATA盘作存储数据，存储性价比高，是大数据量（TB级别的数据量）场景下的推荐机型。

  > **说明** Hadoop、Data Science、Dataflow和Druid类型的集群支持Core节点；Zookeeper和Kafka类型的集群不支持Core节点。
- 本地SSD型

  使用本地SSD盘，具有高随机IOPS（Input/Output Operations Per Second）和高吞吐能力。
- 共享型（入门级）

  共享CPU的机型，在大计算量的场景下，稳定性不够。入门级学习使用，不推荐企业客户使用。
- GPU

  使用GPU的异构机型，可以用来运行机器学习等场景。

实例类型适用场景

- Master主实例

  适合通用型或内存型实例，数据直接使用阿里云的云盘来保存，确保了数据的高可靠性。
- Core核心实例
  - 小数据量（TB级别以下）或者是使用OSS作为主要的数据存储时，推荐使用通用型、计算型或内存型。
  - 大数据量（10 TB或以上）情况下，推荐使用大数据机型，可以获得极高的性价比。

  > **注意** 当Core核心实例使用本地盘时，HDFS数据存储在本地盘，需要您自行保证数据的可靠性。
- Task计算实例

  用于补充集群的计算能力，可以使用除大数据型外的所有机型。

3.1.2.5. 存储说明

本文介绍E-MapReduce集群中数据存储相关的信息，包括磁盘角色、云盘与本地盘，以及OSS。

背景信息

关于存储的类型、性能和相关的限制信息，请参见块存储概述。

磁盘角色

在E-MapReduce集群中，实例节点上有系统盘和数据盘两种角色的磁盘。每块磁盘的配置、类型和容量都可以不同。

磁盘角色	描述
系统盘	系统盘用于安装操作系统。 E-MapReduce默认使用ESSD云盘作为集群的系统盘。系统盘默认是一块。
数据盘	数据盘用于保存数据。 Master实例默认挂载1块云盘作为数据盘，Core实例默认挂载4块云盘作为数据盘。

云盘与本地盘

E-MapReduce集群支持使用以下两种类型的磁盘来存储数据。

块存储类型	描述	使用场景
云盘	包括SSD云盘、高效云盘和ESSD云盘。 磁盘不直接挂载在本地的计算节点上，而是通过网络访问远端的一个存储节点。每一份数据在后端都有两个实时备份，一共三份数据。当一份数据损坏时（磁盘损坏，不是业务上的破坏），E-MapReduce会自动使用备份数据进行恢复。	当业务数据量处于TB级别以下时，推荐您使用云盘，云盘的IOPS和吞吐相比本地盘都会小些。 **说明** 在使用云盘时，如果吞吐量明显不足，则可以新建集群以使用本地盘。
本地盘	包括大数据型的SATA本地盘和本地SSD盘。 磁盘直接挂载在计算节点上，性能高于云盘。本地盘不能选择磁盘数量，只能使用默认配置好的数量，数据也没有后端的备份机制，需要上层的软件来保证数据可靠性。	当数据量处于TB级别以上时，推荐您使用本地盘，本地盘的数据可靠性由E-MapReduce来保证。

在E-MapReduce集群中，当实例节点释放时，所有云盘和本地盘都会清除数据，磁盘无法独立的保存下来并再次使用。Hadoop HDFS会使用所有的数据盘作为数据存储。 Hadoop YARN也会使用所有的数据盘作为计算的临时存储。

OSS

在E-MapReduce集群中，您可以将OSS作为HDFS使用。 E-MapReduce可以方便的读写OSS上的数据，所有使用HDFS的代码经过简单的修改即可以访问OSS的数据。例如：

- 读取HDFS中的数据。

```
sc.textfile("hdfs://user/path")
```

- 更改存储类型为OSS。

```
sc.textfile("oss://user/path")
```

- 对于MR或Hive作业，HDFS命令可以直接操作OSS数据，示例如下。

```
hadoop fs -ls oss://bucket/path
hadoop fs -cp hdfs://user/path  oss://bucket/path
```

此过程中，您不需要输入AccessKey和Endpoint，E-MapReduce会使用当前集群所有者的信息自动补全AccessKey和Endpoint。但OSS的IOPS不高，不适合用在IOPS要求高的场景，例如，流式计算Spark Streaming和HBase。

3.1.2.6. 本地盘机型概述

为了满足大数据场景下的存储需求，阿里云在云上推出了D1系列本地盘机型。

D1系列

D1系列使用本地盘而非云盘作为存储，解决了之前使用云盘产生多份冗余数据而导致的高成本问题。D1系列的数据传输不需要全部通过网络，这样不仅提高了磁盘的吞吐能力，还能发挥Hadoop的就近计算的优势。相较于云盘，本地盘机型提高了存储性能，降低了存储单价，成本与线下物理机相同。

本地盘机型存在一个问题，即数据可靠性问题。对于云盘，阿里云默认具有磁盘多备份策略，您完全感知不到磁盘的损坏，云盘可以自动保证数据的可靠性。对于本地盘，数据可靠性需要由上层的软件来保证，并且磁盘与节点故障也需要人工进行运维处理。

EMR+D1方案

针对本地盘机型（例如D1），E-MapReduce产品推出了一整套的自动化运维方案，帮助您方便可靠的使用本地盘机型。使您不需要关心整个运维的过程，同时还保证了数据高可靠和服务高可用。

自动化运维方案的主要点如下：

- 强制节点的高可靠分布

- 本地盘与节点的故障监控
- 数据迁移时机自动决策
- 自动的故障节点迁移与数据平衡
- 自动的HDFS数据检测
- 网络拓扑调优

通过整个后台的管控系统的自动化运维，E-MapReduce可以协助您更好的使用本地盘机型，实现高性价比的大数据系统。

> 说明 如需使用D1机型搭建Hadoop集群，请提交工单。

3.1.2.7. 集群容灾能力

本文介绍E-MapReduce集群数据容灾和服务容灾能力。

数据容灾

在Hadoop分布式文件系统（HDFS）中，每一个文件的数据均是分块存储的，每一个数据块保存有多个副本（默认为3），并且尽量保证这些数据块副本分布在不同的机架之上。一般情况下，HDFS的副本系数是3，存放策略是将一个副本存放在本地机架节点上，一个副本存放在同一个机架的另一个节点上，最后一个副本放在不同机架的节点上。

HDFS会定期扫描数据副本，如果扫描到有数据副本丢失，则会快速复制这些数据以保证数据副本的数量。如果扫描到节点丢失，则节点上的所有数据也会快速复制恢复。在阿里云上，如果使用的是云盘技术，则每一个云盘在后台都会对应三个数据副本，当其中任一个出现问题时，副本数据都会自动进行切换并恢复，以保证数据的可靠性。

Hadoop HDFS是一个经历了长时间考验且具有高可靠性的数据存储系统，已实现了海量数据的高可靠性存储。同时基于云上的特性，您也可以再在OSS等服务上额外备份数据，以达到更高的数据可靠性。

服务容灾

Hadoop的核心组件都会进行HA部署，即有至少两个节点的服务互备，例如YARN、HDFS、Hive Server和Hive Meta。在任何一时刻，任一服务节点故障时，当前的服务节点都会自动进行切换，以保证服务不受影响。

3.1.2.8. EMR Hive元数据介绍与对比

本文为您介绍E-MapReduce（简称EMR）支持的元数据类型和各元数据类型的优势。

元数据类型介绍

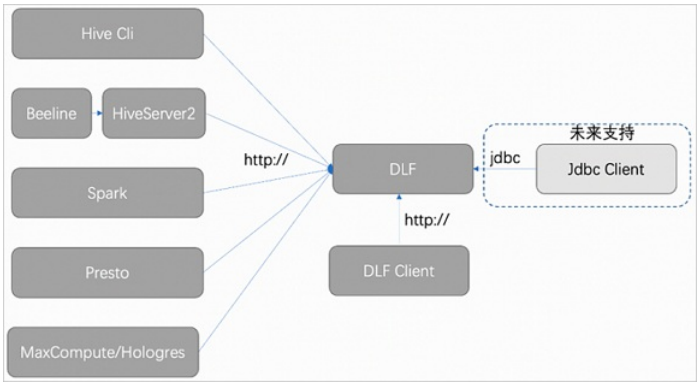
EMR Hive元数据支持DLF统一元数据、自建RDS和内置MySQL三种类型。

DLF统一元数据

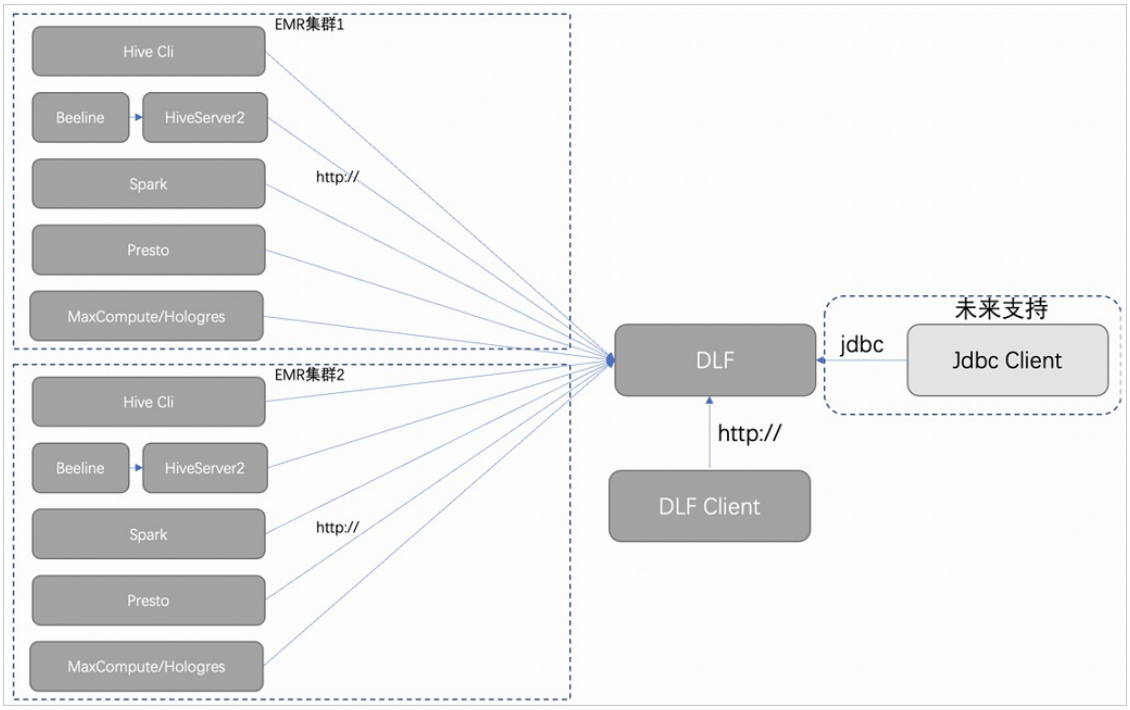
元数据存储在阿里云数据湖构建（Data Lake Formation，简称DLF）中。数据湖构建具有高可用、免运维和高性能等优点，兼容Hive Metastore，无缝对接EMR上开源计算引擎，并支持元数据多版本管理和Data Profile功能。另外，DLF还支持数据探索、湖管理和数据权限控制等功能，并与阿里云其他计算产品（例如MaxCompute、Databricks和Hologres等）无缝对接，可以扩展更丰富的计算场景，DLF详情请参见产品简介。

该元数据类型相比自建RDS和内置MySQL两种方式的最大区别是，无需在EMR集群上部署Hive Metastore，即元数据查询服务以及存储服务都托管到DLF产品上，免去运维成本，同时支持更多引擎（例如MaxCompute、Flink、DataBricks或Hologres等），进一步实现湖仓一体共享元数据，在多个集群上也能够实现元数据共享。DLF Client SDK提供了兼容Hive Metastore的接口，这样引擎基本不做任何改动就可以直接使用DLF Client SDK，进而访问DLF元数据。用户也可以直接使用DLF客户端访问DLF元数据。

DLF统一元数据在单集群部署架构图



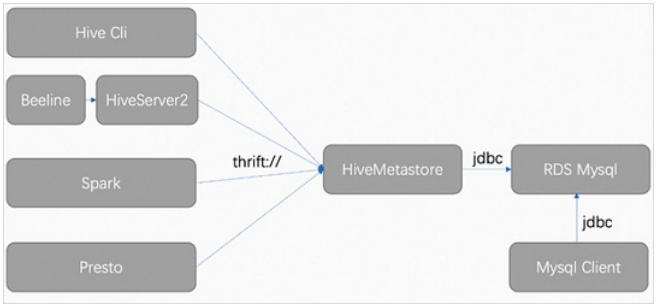
DLF统一元数据在多集群部署架构图



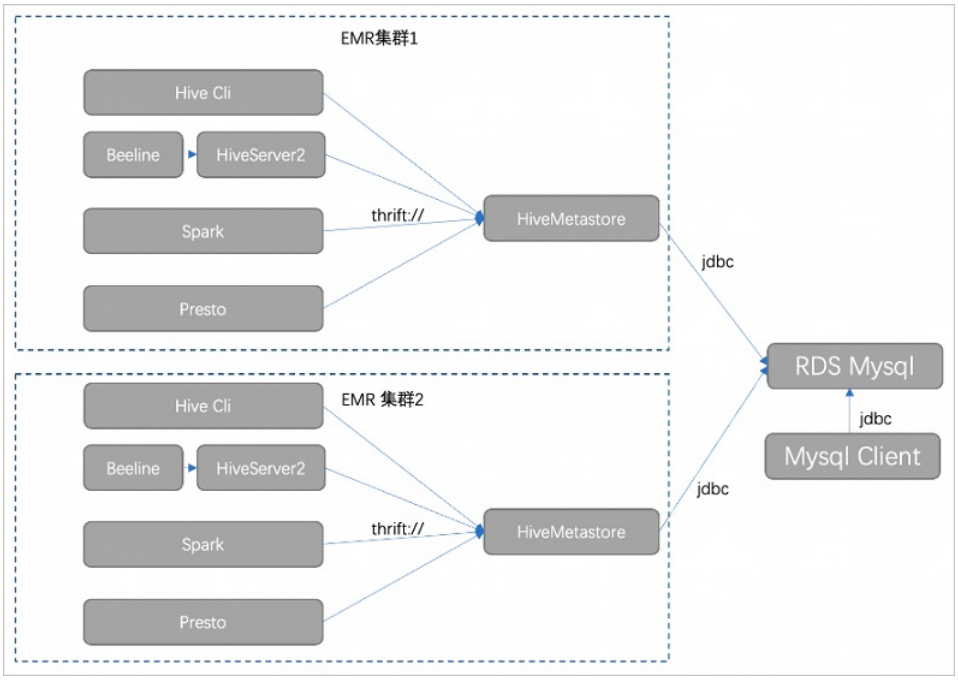
自建RDS

元数据存储在RDS中。自建RDS元数据类型和内置MySQL类型在架构上是一致的，区别只是存储由本地的MySQL，变成了RDS（云上的MySQL）。如自建RDS在多集群部署架构图所示，在多个集群环境中，RDS支持跨多个集群元数据共享，分别被不同集群中的Hive Metastore访问。

自建RDS在单集群部署架构图



自建RDS在多集群部署架构图

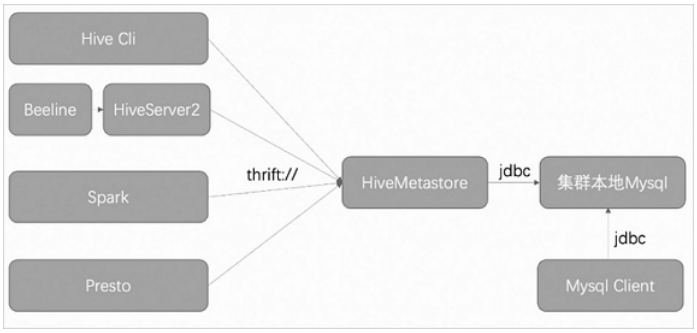


内置MySQL

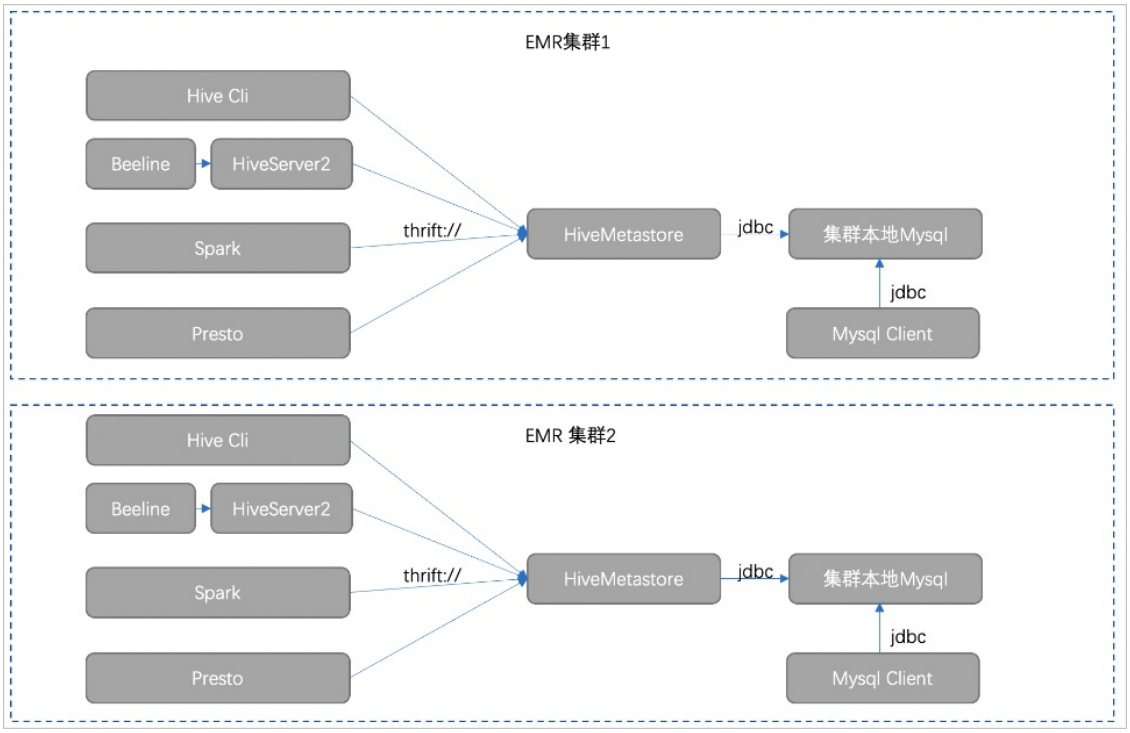
元数据存储在MySQL中，且MySQL Server实例部署在用户的EMR集群中（通常是Master节点），Hive、Spark或Presto等引擎访问元数据时，也不是直接访问MySQL，而是通过访问Hive Metastore间接访问MySQL。Hive Metastore提供元数据访问服务，引擎通过thrift协议访问Hive Metastore，Hive Metastore通过JDBC协议访问MySQL。

当然您也可以手动通过MySQL客户端直接连接MySQL Server，查看元数据。如内置MySQL在多集群部署架构图所示，由于每个集群都有一个MySQL，导致多个集群间的元数据不能共享。

内置MySQL在单集群部署架构图



内置MySQL在多集群部署架构图



元数据类型优势

内置MySQL和自建RDS的区别

自建RDS更直观的好处是元数据可以在多个集群间共享。

从可用性、可靠性和性能等方面对比，自建RDS要优于内置MySQL，详情请参见RDS与自建数据库对比优势。

DLF统一元数据和自建RDS的区别

对比项	DLF统一元数据	自建RDS
易用性	EMR集群开箱即用（需提前开通DLF产品）。	EMR集群开箱即用（需提前购买RDS）。
元数据管理	DLF产品提供了可视化的元数据检索、元数据管理、多版本管理、数据统计概况和生命周期管理等更丰富的能力。	无。
多引擎支持	• 支持Hive、Spark和Presto。 • 支持MaxCompute和Hologres。	• 支持Hive、Spark和Presto。 • 不支持MaxCompute和Hologres。
成本	免费，具体计费请参见数据湖构建的计费模式。	包月和按量计费。
运维成本	免运维，自动水平弹性扩容。	需要进行升级和扩容等运维操作。
高可用	主备等容灾措施。	主备等容灾措施。
性能	性能高，同时基于Hive Metastore进行了优化。	性能高，对Hive Metastore无优化。
更多能力	细粒度数据权限控制、湖存储分析和湖格式管理。	无。

非EMR集群访问DLF元数据

非EMR集群（本地测试环境或者其它云服务）访问DLF元数据，需要集成DLF Client SDK，具体操作请参见阿里云数据湖构建（DLF）。

> **说明** 访问DLF和访问MySQL一样，需要提供访问地址、用户名和密码。
>
> - DLF中的访问地址称为Endpoint，不同region使用不同的Endpoint。
> - 用户名和密码为DLF产品的AccessKey ID和AccessKey Secret。
> - 您还需要通过修改Hive参数的方式，切换Hive MetaStore的存储方式。
>
> 在Hive服务页面，修改参数hive.imetastoreclient.factory.class的值为com.aliyun.datalake.metastore.hive2.DlfMetaStoreClientFactory。修改配置项的具体操作请参见修改配置项。

3.1.3. 集群配置

3.1.3.1. 创建集群

本文为您介绍在EMR on ECS控制台创建集群的详细操作步骤和相关配置。

前提条件

已完成RAM授权，详细信息请参见角色授权。

操作步骤

1. 进入**集群管理**页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   - 地域：创建的集群将会在对应的地域内，一旦创建不能修改。
   - 资源组：默认显示账号全部资源。
2. 单击上方的**创建集群**。
3. 配置集群信息。

   创建集群时，您需要对集群进行软件配置、硬件配置、基础配置和确认订单。

   > 注意　集群创建完成后，除了集群名称以外，其他配置均无法修改，所以在创建时请仔细确认各项配置。

   i. 软件配置。

配置项	描述
地域	集群节点ECS实例所在的物理位置。

配置项	描述
业务场景	■ **新版数据湖**：提供更灵活、可靠、高效的管理集群，更快的运行大数据计算引擎并提供出色的数据分析能力。 ■ 支持构建数据湖架构，使用JindoFS进行数据湖加速。 ■ 支持OSS-HDFS（全托管HDFS）作为存储，减少您的运维成本，完全基于使用量计费。 更多信息，请参见数据湖集群。 创建的集群类型为DataLake。 ■ **数据分析**：将海量数据通过导入或者外表等形式引入到OLAP分析引擎里，例如，Clickhouse、StarRocks，提供高效、实时和灵活的数据分析能力，满足用户画像、人群圈选、BI报表和业务分析等一系列的业务场景。 创建的集群类型为OLAP。 ■ **实时数据流**：是EMR平台上提供的实时计算一站式解决方案，拥有分布式的、高吞吐量和高可扩展性的消息系统Kafka和基于Apache Flink官方产品Ververica提供的Flink商业内核两大组件，专注于解决实时计算端到端的各类问题、广泛应用于实时数据ETL和日志采集分析等场景，您也可以单独使用其中任一组件。 创建的集群类型为DataFlow。 ■ **大数据开发平台**：是EMR平台上基于开源组件的大数据开发平台，提供一站式的端到端大数据开发体验。 ■ 为数据工程师、数据分析师和数据科学家提供Notebook数据应用开发环境。 ■ 集成Airflow ，提供跨集群的任务调度能力。 ■ 自动适配EMR的计算引擎，无需复杂配置，实现多引擎之间的协同工作。 更多信息，请参见EMR Studio概述。 创建的集群类型为EMR Studio。 ■ **机器学习**：主要面向大数据+AI场景。 ■ 提供分布式深度学习框架。 ■ 提供两百多种经典机器学习算法包。 ■ 提供AutoML能力，10余种深度学习算法，覆盖推荐和广告等场景。 创建的集群类型为Data Science。 ■ **数据湖**：用于构建大规模数据处理框架和管道，适用于大数据分析，支持Apache Hive、Spark和Presto等开源框架。支持的集群类型如下： ■ **Hadoop**： ■ 提供最丰富的开源组件列表，完全兼容Hadoop生态。 ■ 可应用于大数据离线处理、实时处理和交互式查询等多种使用场景。 ■ 支持构建数据湖架构，使用JindoFS进行数据湖加速。 ■ **Zookeeper**：提供独立的分布式一致性锁服务，适用于大规模的Hadoop集群、HBase集群和Kafka集群。 ■ **Presto**：是基于内存的分布式SQL交互式查询引擎。支持多种数据源，适合PB级海量数据的复杂分析，以及跨数据源的查询。
产品版本	默认最新的软件版本。
服务高可用	默认关闭。开启后，集群会有多个Master节点来支持ResourceManager和NameNode的高可用。 HBase集群原本就支持高可用，只是另一个节点用其中一个Core节点来充当，如果打开高可用，会独立使用一个Master节点来支持高可用，更加的安全可靠。
可选服务	根据您的实际需求选择其他的一些组件，被选中的组件会默认启动相关的服务进程。 **说明** 组件越多，对机器的配置要求也越高，所以在下面的步骤中您需要根据实际的组件数量进行机器选型，否则可能没有足够的资源运行这些服务。

配置项	描述
元数据	说明 仅Hadoop集群类型支持配置该参数，具体以控制台为准。 ■ **DLF统一元数据**：表示元数据存储在数据湖中。 ■ **自建RDS**：表示使用自建的阿里云RDS作为元数据库，更多信息请参见配置独立RDS。 ■ **内置MySQL**：表示元数据存储在集群本地环境的MySQL数据库中。 仅限在测试场景下使用该方式，生产场景建议选择前两种方式。
高级设置	■ **Kerberos身份认证**：是否开启集群的Kerberos认证功能。默认不开启。 说明 仅Hadoop和Presto集群类型支持配置该参数，具体以控制台为准。 ■ **软件自定义配置**：可指定JSON文件对集群中的基础软件（例如Hadoop、Spark和Hive等）进行配置，详细使用方法请参见配置自定义软件。默认不开启。

ii. 硬件配置。

配置项	说明
付费类型	默认包年包月。当前支持的付费类型如下： ■ **按量付费**：一种后付费模式，即先使用再付费。按量付费是根据实际使用的小时数来支付费用，每小时计费一次，适合短期的测试任务或是灵活的动态任务。 ■ **包年包月**：一种预付费模式，即先付费再使用。 说明 ■ 建议测试场景下使用**按量付费**，测试正常后再新建一个**包年包月**的生产集群正式使用。 ■ 包年包月实例还需选择付费时长和是否开启自动续费。默认续费时长为1个月，且未开启自动续费。开启自动续费后，实例到期前7天会执行自动续费操作，续费时长为1个月，详情请参见续费流程。
可用区	可用区为在同一地域下的不同物理区域，可用区之间内网互通。通常使用默认的可用区即可。
专有网络	默认选择已有的专有网络。 如需创建新的专有网络，请在专有网络控制台新创建一个，详情请参见创建和管理专有网络。
交换机	选择在对应VPC下可用区的交换机，如果在这个可用区没有可用的交换机，则需要在专有网络控制台新创建一个，详情请参见创建和管理交换机。
安全组	默认选择已有的安全组。安全组详情请参见安全组概述。 您也可以单击**新建安全组**，在ECS控制台新建一个安全组，详情请参见创建安全组。 注意 禁止使用ECS上创建的企业安全组。
挂载公网	集群是否挂载弹性公网IP地址，默认不开启。 DataLake集群类型的挂载公网在**实例**区域，您可以针对实例选择是否开启挂载公网。 说明 创建后如果您需要使用公网IP地址访问，请在ECS上申请开通公网IP地址，详情请参见弹性公网IP中的申请EIP的内容。

配置项	说明
实例	您可以根据需要选择实例规格，详情请参见实例规格族。 ■ **Master实例**：主要负责ResourceManager和NameNode等控制进程的部署。 ■ **系统盘配置**：根据需要选择SSD云盘、ESSD云盘或者高效云盘。 ■ **系统盘大小**：根据需要调整磁盘容量，推荐至少120 GiB。取值范围为60 ~ 500 GiB。 ■ **数据盘配置**：根据需要选择SSD云盘、ESSD云盘或者高效云盘。 ■ **数据盘大小**：根据需要调整磁盘容量，推荐至少80 GiB。取值范围为40 ~ 32768 GiB。 ■ **Master数量**：默认1台。如果开启高可用，可以有多台Master实例。 ■ **Core实例**：主要负责集群所有数据的存储，创建集群完成后也支持按需进行扩容。 ■ **系统盘配置**：根据需要选择SSD云盘、ESSD云盘或者高效云盘。 ■ **系统盘大小**：根据需要调整磁盘容量，推荐至少120 GiB。取值范围为60 ~ 500 GiB。 ■ **数据盘配置**：根据需要选择SSD云盘、ESSD云盘或者高效云盘。 ■ **数据盘大小**：根据需要调整磁盘容量，推荐至少80 GiB。取值范围为40 ~ 32768 GiB。 ■ **Core数量**：默认2台，根据需要调整。 ■ **Task实例**：不保存数据，调整集群的计算力使用。默认不开启，需要时再追加。

iii. 基础配置。

配置项	说明
集群名称	集群的名字，长度限制为1~64个字符，仅可使用中文、字母、数字、短划线（-）和下划线（_）。
身份凭证	**密钥对**（默认）：使用SSH密钥对登录Linux实例。 关于密钥对的使用详情，请参见SSH密钥对。
	密码：设置Master节点的登录密码，使用密码对登录Linux实例。 密码规则：8~30个字符，且必须同时包含大写字母、小写字母、数字和特殊字符。 特殊字符包括：感叹号（!）、at（@）、井号（#）、美元符号（$）、百分号（%）、乘方（^）、and（&）和星号（*）。
高级设置	■ **添加用户**：添加访问开源大数据软件Web UI的账号。 ■ **ECS应用角色**：当用户的程序在EMR计算节点上运行时，可不填写阿里云AccessKey来访问相关的云服务（例如OSS），EMR会自动申请一个临时AccessKey来授权本次访问。**ECS应用角色**用于控制这个AccessKey的权限。 ■ **引导操作**：可选配置，您可以在集群启动Hadoop前执行您自定义的脚本，详情请参见管理引导操作。 ■ **标签**：可选配置，您可以在创建集群时绑定标签，也可以在集群创建完成后，在集群详情页绑定标签，详情请参见设置标签。 ■ **资源组**：可选配置。详情请参见使用资源组。

> **说明** 页面右边会显示您所创建集群的配置清单以及集群费用。根据不同的付费类型，展示不同的价格信息。

4. 当所有的信息确认正确后，选中服务协议，单击**确认订单**。

> 注意
> - 按量付费集群：立刻开始创建。
>
>   集群创建完成后，集群的状态变为**运行中**。
> - 包年包月集群：先生成订单，支付完成订单以后集群才会开始创建。

3.1.3.2. 创建Gateway集群

您可以通过Gateway集群实现负载均衡和安全隔离，也可以通过Gateway集群向E-MapReduce集群提交作业。本文为您介绍如何在E-MapReduce中创建Gateway集群。

前提条件

已经在E-MapReduce中创建了Hadoop或Kafka类型的集群，详情请参见创建集群。

使用限制

在E-MapReduce中，Gateway集群仅支持关联Hadoop或Kafka类型的集群。

操作步骤

1. 登录EMR on ECS控制台。
2. 在**集群管理**页面，单击**创建Gateway**。
3. 在**创建Gateway**页面，配置各参数。

模块	参数	描述
关联设置	地域	Gateway集群所在的物理位置。
	资源组	选择Gateway集群所属的资源组。 如果需要创建新的资源组，单击**创建资源组**，详细信息请参见创建资源组。
	关联集群	根据所选地域筛选出Gateway集群可以关联的计算集群。待关联的集群有以下要求： ○ 集群状态须为运行中。 ○ 仅支持关联Hadoop或Kafka类型的集群。 **说明** 当选择关联集群之后，Gateway集群的VPC默认与关联集群一致。新旧版控制台的集群均可关联。
基础设置	**付费类型**	○ **包年包月**：一种预付费模式，即先付费再使用。 ○ **按量付费**：一种后付费模式，即先使用再付费。按量付费是根据实际使用的小时数来支付费用，每小时计费一次，适合短期的测试任务或是灵活的动态任务。
	可用区	关联集群所在的可用区（Zone）。
	交换机	选择在对应的VPC下对应可用区的交换机。
	安全组名称	关联集群所属的安全组。
	挂载公网	Gateway是否挂载弹性公网IP地址。
	实例	该地域内可选择的ECS实例规格，详细说明请参见实例规格族。 ○ **系统盘配置**：Gateway节点使用的系统盘类型。系统盘有高效云盘、ESSD云盘和SSD云盘三种，根据不同机型和不同的Region，系统盘显示类型会有不同。系统盘默认随着集群的释放而释放。 ○ **系统盘大小**：每块最小为40 GB，最大为500 GB。默认值为300 GB。 ○ **数据盘配置**：Gateway节点使用的数据盘类型。数据盘有高效云盘、ESSD云盘和SSD云盘三种，根据不同机型和不同的Region，数据盘显示类型会有不同。数据盘默认随着集群的释放而释放。 ○ **数据盘大小**：每块最小为200 GB，最大为4000 GB。默认值为300 GB。 ○ **数量**：数据盘的数量，最小设置为1台，最大设置为10台。
	集群名称	Gateway集群的名称，长度限制为1~64个字符，只允许包含中文、字母、数字、短划线（-）、下划线（_）。
	身份凭证	登录Gateway集群所有节点的用户凭证。 ○ **登录密码**：在文本框中输入登录Gateway的密码。长度限制为8~30个字符，且必须同时包含大写字母、小写字母、数字和特殊字符。 支持输入以下字符： !@#$%^&* ○ **密钥对**：在列表中选择登录Gateway的密钥对名称。如果还未创建过密钥对，则您可以单击后面的**新建密钥对**，进入ECS管理控制台进行创建。 请妥善保管好密钥对所对应的私钥文件（*.pem*文件）。Gateway创建成功后，该密钥对的公钥部分会自动绑定到Gateway所在的云服务器ECS上，当通过SSH登录Gateway时，您需要输入私钥文件中的私钥。
	ECS应用角色	通过RAM角色为在集群上运行的应用程序提供调用其他阿里云服务所需的必要权限，无需调整，使用默认即可。默认值为AliyunECSInstanceForEMRRole。

模块设置	参数	描述
	引导操作	可选配置，您可以在创建集群时绑定标签，也可以在集群创建完成后，在集群详情页绑定标签，详情请参见管理引导操作和手动执行脚本。
	标签	可选配置，您可以在集群启动前执行您自定义的脚本，详情请参见设置标签。

4. 完成上述参数配置后，选中**E-MapReduce服务条款**，单击**创建**。
   创建成功后，集群的**状态**变为**空闲**。

3.1.3.3. 管理权限

3.1.3.3.1. 角色授权

阿里云E-MapReduce服务（例如Hadoop和Spark），在运行时需要有访问其他阿里云资源和执行操作的权限。每个E-MapReduce集群必须有服务角色以及ECS应用角色。本文为您介绍EMR角色授权的流程及其关联的角色。

背景信息

阿里云E-MapReduce为确定权限的角色提供默认角色和默认系统策略。系统策略由阿里云创建和维护，因此如果服务要求发生变化，策略会自动更新。

首次使用E-MapReduce服务时，您需要使用阿里云账号为E-MapReduce服务授权名为AliyunEMRDefaultRole、AliyunECSInstanceForEMRRole或AliyunEmrEcsDefaultRole的服务角色。授权成功后，您可以在RAM控制台上查看角色，并为角色挂载策略。角色详细信息，请参见RAM角色概览。

注意

- E-MapReduce不同版本需要授权的服务角色名称不同：
  - EMR-3.32.0及之前版本和EMR-4.5.0及之前版本：AliyunEmrEcsDefaultRole
  - EMR-3.32.0之后版本和EMR-4.5.0之后版本：AliyunECSInstanceForEMRRole
- 首次使用E-MapReduce服务时，必须使用阿里云账号完成默认角色授权，否则RAM用户和阿里云账号不能使用E-MapReduce。
- 如果删除服务角色，请确保使用该角色的资源已经释放，否则会影响资源的正常使用。
- 如果只授权了部分角色，E-MapReduce控制台会提醒授权，只有在全部角色授权完成之后才可以创建集群。

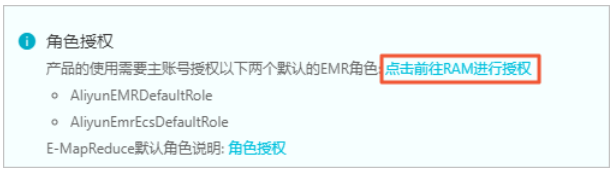
角色授权流程

本流程以EMR-3.30版本为例。

1. 在登录EMR on ECS控制台。，单击**点击前往RAM进行授权**。

   说明 首次使用E-MapReduce服务时需要授权，授权成功后，再次使用无需重复授权。

   如果未正确地给E-MapReduce的服务账号授予默认角色，则在创建集群或创建按需执行计划时，会弹出如下提示。

2. 单击**同意授权**，将默认角色**AliyunEMRDefaultRole**和**AliyunEmrEcsDefaultRole**授予给E-MapReduce服务。

3. 完成以上授权后，您需要刷新E-MapReduce控制台，即可进行相关操作。

   如果您想查看**AliyunEMRDefaultRole**和**AliyunEmrEcsDefaultRole**相关的详细策略信息，您可以登录RAM的控制台查看。

服务角色

下表列出了与E-MapReduce关联的RAM角色。

属性	默认角色	描述	系统策略
EMR服务角色	AliyunEMRDefaultRole	允许E-MapReduce服务在配置资源和执行服务级别操作时调用其他阿里云服务。所有集群都需要该角色，且不能更改。 详情请参见EMR服务角色。	AliyunEMRRolePolicy
ECS应用角色（EMR 3.32及之前版本和EMR 4.5及之前版本）	AliyunEmrEcsDefaultRole	集群实例上运行的应用程序进程在调用其他阿里云服务时将使用该角色。在创建集群时既可以使用该服务角色，也可以使用自定义的角色。 AliyunEmrEcsDefaultRole角色，详情请参见ECS应用角色（EMR 3.32及之前版本和EMR 4.5及之前版本）。	AliyunEMRECSRolePolicy
ECS应用角色（EMR 3.32之后版本和EMR 4.5之后版本）	AliyunECSInstanceForEMRRole	集群实例上运行的应用程序进程在调用其他阿里云服务时将使用该角色。在创建集群时既可以使用该服务角色，也可以使用自定义的角色。 AliyunECSInstanceForEMRRole角色，详情请参见ECS应用角色（EMR 3.32之后版本和EMR 4.5之后版本）。	AliyunECSInstanceForEMRRolePolicy
ECS应用角色（EMR Studio默认使用）	AliyunECSInstanceForEMRStudioRole	EMR Studio使用此角色来访问您在其他云产品中的资源。 当您的账号未授权该角色，首次创建EMR Studio集群时会弹出授权该角色窗口，请使用阿里云账号授权该角色。	AliyunECSInstanceForEMRStudioRolePolicy

3.1.3.3.2. EMR服务角色

EMR服务角色允许E-MapReduce服务在配置资源或执行服务级别操作时调用其他阿里云服务。例如，服务角色用于在EMR集群启动时创建ECS实例。本文为您介绍EMR服务角色AliyunEMRDefaultRole及其权限策略。

注意事项

为了避免影响EMR服务稳定性，请注意：

- EMR服务角色名称无法修改。
- 不要在RAM访问控制台上删除或修改EMR服务角色的系统策略。

权限内容

AliyunEMRDefaultRole是默认的EMR服务角色，该服务角色包含**AliyunEMRRolePolicy**系统权限策略。其中**AliyunEMRRolePolicy**包含的权限内容如下：

- ECS相关权限

名称（Action）	说明
ecs:CreateInstance	创建ECS实例。
ecs:RenewInstance	为ECS实例续费。
ecs:DescribeRegions	查询ECS地域信息。
ecs:DescribeZones	查询Zone信息。
ecs:DescribeImages	查询镜像信息。
ecs:CreateSecurityGroup	创建安全组。
ecs:AllocatePublicIpAddress	分配公网IP地址。
ecs:DeleteInstance	删除ECS实例。
ecs:StartInstance	启动ECS实例。
ecs:StopInstance	停止ECS实例。
ecs:DescribeInstances	查询ECS实例。
ecs:DescribeDisks	查询ECS相关磁盘信息。
ecs:AuthorizeSecurityGroup	设置安全组入规则。
ecs:AuthorizeSecurityGroupEgress	设置安全组出规则。
ecs:DescribeSecurityGroupAttribute	查询安全组详情。
ecs:DescribeSecurityGroups	查询安全组列表信息。

- OSS相关权限

名称（Action）	说明
oss:PutObject	上传文件或文件夹对象。
oss:GetObject	获取文件或文件夹对象。
oss:ListObjects	查询文件列表信息。

3.1.3.3.3. ECS应用角色（EMR 3.32及之前版本和EMR 4.5及之前版本）

E-MapReduce环境提供了MetaService服务，MetaService服务是一种特殊的ECS应用角色。EMR 3.32及之前版本和EMR 4.5及之前版本，创建时会自动绑定该角色。在EMR集群之上运行的应用程序通过该角色来获得与其他云服务交互的权限，实现以免AccessKey的方式访问阿里云资源，避免了在配置文件中暴露AccessKey的风险。

前提条件

已授权该角色，详情请参见角色授权。

背景信息

当前MetaService服务仅支持免AccessKey访问OSS、LogService和MNS数据。

权限内容

默认服务角色**AliyunEmrEcsDefaultRole**包含系统权限策略为**AliyunEmrECSRolePolicy**，OSS相关权限内容如下。

权限名称（Action）	权限说明
oss:PutObject	上传文件或文件夹对象。
oss:GetObject	获取文件或文件夹对象。
oss:ListObjects	查询文件列表信息。
oss:DeleteObject	删除某个文件。

权限名称（Action）	权限说明
oss:AbortMultipartUpload	终止MultipartUpload事件。

注意 请谨慎编辑和删除默认角色AliyunEmrEcsDefaultRole，否则会造成集群创建失败或作业运行失败。

支持MetaService的数据源

目前在E-MapReduce上支持MetaService的产品有OSS、LogService和MNS。您可以在E-MapReduce集群上使用E-MapReduce SDK接口免AccessKey读写上述数据源。

MetaService默认只有OSS的读写权限，如果您希望MetaService支持LogService和MNS，请前往RAM控制台为AliyunEmrEcsDefaultRole应用角色增加LogService和MNS的读写权限，RAM控制台链接为RAM控制台。

RAM角色授权，详情请参见为RAM角色授权。

使用MetaService

基于MetaService服务，您可通过E-MapReduce作业免AccessKey访问阿里云资源（OSS、LogService和MNS），其优势如下：

- 降低AccessKey泄漏的风险。基于RAM，您可按最小够用原则给角色授权，做到权限最小化，这样可以将安全风险降到最低。
- 提高用户体验。尤其在交互式访问OSS资源时，可避免让您输入一长串的OSS路径。
- EMR自带服务

  EMR自带服务中运行的作业均可以自动基于MetaSerivce服务免明文AccessKey访问阿里云资源（OSS、LogService和MNS）

  以下是使用MetaService（新）和不使用MetaService（旧）的对比示例：

  - 通过Hadoop命令行查看OSS数据
    - 旧方式

      ```
      hadoop fs -ls oss://ZaH******As1s:Ba23N**************sdaBj2@bucket.oss-cn-hangzhou-internal.aliyuncs.com/a/b/c
      ```

    - 新方式

      ```
      hadoop fs -ls oss://bucket/a/b/c
      ```

  - 通过Hive建表
    - 旧方式

      ```
      CREATE EXTERNAL TABLE test_table(id INT, name string)
              ROW FORMAT DELIMITED
              FIELDS TERMINATED BY '/t'
              LOCATION 'oss://ZaH******As1s:Ba23N**************sdaBj2@bucket.oss-cn-hangzhou-internal.aliyuncs.com/a/b/c';
      ```

    - 新方式

      ```
      CREATE EXTERNAL TABLE test_table(id INT, name string)
              ROW FORMAT DELIMITED
              FIELDS TERMINATED BY '/t'
              LOCATION 'oss://bucket/a/b/c';
      ```

  - 使用Spark查看OSS数据
    - 旧方式

      ```
      val data = sc.textFile("oss://ZaH******As1s:Ba23N**************sdaBj2@bucket.oss-cn-hangzhou-internal.aliyuncs.com/a/b/c")
      ```

    - 新方式

      ```
      val data = sc.textFile("oss://bucket/a/b/c")
      ```

- 自行部署服务

  MetaService是一个HTTP服务，您可以直接访问这个HTTP服务的URL获取STS临时凭证，然后在自行搭建的系统中使用该STS临时凭证免AccessKey访问阿里云资源。

  注意 STS临时凭证失效前半小时会生成新的STS临时凭证，在这半小时内，新旧STS临时凭证均可使用。

  例如，通过**curl http://localhost:10011/cluster-region**，即可获得当前集群所在Region。

  当前MetaService支持以下几类信息：

  - Region：/cluster-region

- 角色名：/cluster-role-name
- AccessKeyID：/role-access-key-id
- AccessKeySecret：/role-access-key-secret
- SecurityToken：/role-security-token
- 网络类型：/cluster-network-type

3.1.3.3.4. ECS应用角色（EMR 3.32之后版本和EMR 4.5之后版本）

EMR 3.32之后版本和EMR 4.5之后版本，将Metaservice服务替换为ECS应用角色，在EMR集群创建和扩容时自动分配给EMR集群中的每个ECS实例。在EMR集群之上运行的应用程序通过该角色来获得与其他云服务交互的权限，实现以免AccessKey的方式访问阿里云资源，避免了在配置文件中暴露AccessKey的风险。

前提条件

已授权该角色，详情请参见角色授权。

权限内容

默认角色**AliyunECSInstanceForEMRRole**包含系统权限策略为**AliyunECSInstanceForEMRRolePolicy**，OSS相关权限内容如下。

权限名称（Action）	权限说明
oss:PutObject	上传文件或文件夹对象。
oss:GetObject	获取文件或文件夹对象。
oss:ListObjects	查询文件列表信息。
oss:DeleteObject	删除某个文件。
oss:AbortMultipartUpload	终止MultipartUpload事件。

> 注意　请谨慎编辑和删除默认角色AliyunEmrEcsDefaultRole，否则会造成集群创建失败或作业运行失败。

使用ECS应用角色获取临时凭证

基于STS临时凭证可以获取本账号下其他云资源的访问权限，详情请参见使用实例RAM角色访问其他云产品。

3.1.3.3.5. 使用自定义ECS应用角色访问同账号云资源

本文介绍在E-MapReduce控制台上，通过创建集群时在基础配置页面的高级设置区域设置ECS应用角色，实现以免密的方式访问同账号下的其它资源。例如，对象存储OSS和日志服务SLS。

背景信息

您在创建集群时可以使用自定义的角色，通过给该角色不同的权限策略，以限制集群访问外部资源的权限。例如，您可以进行如下操作：

- 指定集群只能访问指定OSS的数据目录。
- 指定集群访问指定的外部资源。

前提条件

- 已创建EMR集群，详情请参见创建集群。
- 已完成角色授权，详情请参见角色授权。
- 已在OSS管理控制台，创建与EMR集群同一地域下的存储空间，详情请参见创建存储空间。

操作流程

1. 步骤一：新建权限策略
2. 步骤二：创建RAM角色
3. 步骤三：创建集群并访问外部资源

步骤一：新建权限策略

1. 进入新建自定义权限策略页面。
   i. 使用云账号登录RAM控制台。
   ii. 在RAM访问控制页面，选择**权限管理 > 权限策略**。
   iii. 在**权限策略**页面，单击**创建权限策略**。

2. 在**新建自定义权限策略**页面，配置以下信息。

参数	描述
策略名称	本示例为test-emr。
配置模式	选择**脚本配置**。
策略内容	添加如下策略。 `{ "Version": "1", "Statement": [ { "Action": [ "oss:GetObject", "oss:ListObjects" ], "Resource": [ "acs:oss:*:*:emr-logs2", "acs:oss:*:*:emr-logs2/*" ], "Effect": "Allow" } ] }` **说明** 策略中涉及的元素如下所示： ◦ Action：是指对具体资源的操作。本示例是OSS的读取和查询目录的权限。 ◦ Resource：是指被授权的具体对象。本示例是访问*test-emr*的OSS Bucket及其中的内容。 更多权限策略的基本元素，请参见权限策略基本元素。

3. 单击**确定**。

步骤二：创建RAM角色

1. 在RAM访问控制页面，选择**身份管理 > 角色**。
2. 在**角色**页面，单击**创建角色**。
3. 创建RAM角色。
   i. 单击**阿里云服务**。
   ii. 单击**下一步**。
   iii. 在**配置角色**面板，配置以下信息。

参数	描述
角色名称	本示例为test-emr。
选择授信服务	选择**云服务器**。

   iv. 单击**完成**。
4. （可选）修改授信服务。

> **注意** 如果您创建的集群是EMR 3.32之后版本、EMR 4.5之后版本或EMR 5.x及之后版本，则无需执行本步骤。

   i. 在**角色**页面，单击刚创建的**角色名称**。
   ii. 单击**信任策略管理**页签。
   iii. 单击**修改信任策略**。

iv. 修改 `ecs.aliyuncs.com` 为 `emr.aliyuncs.com` 。

```
{
  "Statement": [
    {
        "Action": "sts:AssumeRole",
        "Effect": "Allow",
        "Principal": {
          "Service": [
            "ecs.aliyuncs.com"
          ]
        }
    }
  ],
  "Version": "1"
}
```

v. 单击**确定**。

5. 添加相应权限。
   i. 在**角色**页面，单击刚创建**角色名称**的操作列的**添加权限**。
   ii. 在**添加权限**页面，单击**自定义策略**，添加新建的权限策略。
   iii. 单击**确定**。
   iv. 单击**完成**。

步骤三：创建集群并访问外部资源

1. 登录EMR on ECS控制台。
2. 在顶部菜单栏处，根据实际情况选择地域和资源组。
3. 单击**创建集群**，在**基础配置**页面的**高级设置**区域，添加步骤二：创建RAM角色中创建的角色名称。创建详情请参见创建集群。

4. 集群创建成功后，通过SSH登录主节点，详情请参见登录集群。
   执行以下命令，验证授权是否成功。

```
hdfs dfs -ls oss://<yourBucketName>/
```

> **说明** 示例中的<yourBucketName>为您OSS Bucket的名称。

- 没有该Bucket访问权限时，提示如下信息。

```
[root@emr-header-1 ~]# hdfs dfs -ls oss://emr-logs2/
ls: java.io.IOException: ErrorCode : 403 , ErrorMsg: HTTP/1.1 403 Forbidden: <?xml version="1.0" enco
ding="UTF-8"?>
```

- 有该Bucket访问权限时，提示如下信息。

```
[root@emr-header-1 ~]# hdfs dfs -ls oss://emr-logs2/
Found 10 items
-rw-rw-rw-   1         30 2020-03-10 16:08 oss://emr-logs2/
drwxrwxrwx   -          0 2019-12-20 14:49 oss://emr-logs2/
drwxrwxrwx   -          0 2020-03-12 23:56 oss://emr-logs2/
-rw-rw-rw-   1        161 2020-05-18 22:01 oss://emr-logs2/
drwxrwxrwx   -          0 2019-12-20 14:22 oss://emr-logs2/
drwxrwxrwx   -          0 2020-05-18 22:28 oss://emr-logs2/
```

常见问题

- Q：创建集群时提示NoPermission。

  A：您可以参照如下方式排查解决。

i. 您创建集群使用的用户是否有创建集群和更换ECS应用角色的权限，如果该RAM用户权限为AliyunEMRDevelopAccess可以修改为AliyunEMRFullAccess。

ii. 创建集群时ECS应用角色名称是否填写正确。

iii. 授信策略是否修改为*emr.aliyuncs.com*。

- Q：HDFS无法访问OSS路径

```
[root@emr-header-1 ~]# hdfs dfs -ls oss://
ls: java.io.IOException: ErrorCode : 403 , ErrorMsg: HTTP/1.1 403 Forbidden ERROR_CODE : 1010
```

A：您可以参照如下方式排查解决。

i. 确认访问的OSS Bucket是否和集群在一个地域（Region），如果不在同一地域（Region），在访问链接中需要添加相应的Endpoint。

ii. 确认访问的OSS Bucket是否包含在新建的权限策略中，如果没有，需要修改权限策略。

iii. 确认是否在OSS控制台上设置了该Bucket的相关权限。如果设置了相关权限，您可以在OSS控制台上取消相关权限的设置，通过设置权限策略中的Action内容来设置相关权限。

3.1.3.3.6. 为RAM用户授权

为确保RAM用户能正常使用E-MapReduce控制台功能，您需要使用阿里云账号登录访问控制RAM（Resource Access Management）控制台，授予RAM用户相应的权限。

背景信息

访问控制RAM是阿里云提供的资源访问控制服务，详情请参见什么是访问控制。在E-MapReduce中，RAM的典型使用场景如下：

- 用户：如果您购买了多台E-MapReduce集群实例，您的组织里有多个用户（如运维、开发或数据分析）需要使用这些实例，您可以创建一个策略允许部分用户使用这些实例。避免了将同一个AccessKey泄露给多人。
- 用户组：您可以创建多个用户组，并授予不同权限策略，授权过程与授权用户过程相同，实现批量管理用户权限。

权限策略

权限策略分为系统策略和自定义策略：

- 系统策略

  阿里云提供多种具有不同管理目的的系统策略，E-MapReduce使用的系统策略如下。

系统策略名称	描述	包含的权限
AliyunEMRFullAccess	E-MapReduce管理员权限	所有权限。
AliyunEMRDevelopAccess	E-MapReduce开发者权限	除集群的创建和释放等权限外的所有权限。
AliyunEMRFlowAdmin	E-MapReduce数据开发的管理员权限	创建项目、开发和管理作业权限（不包含添加项目成员和管理集群权限）。

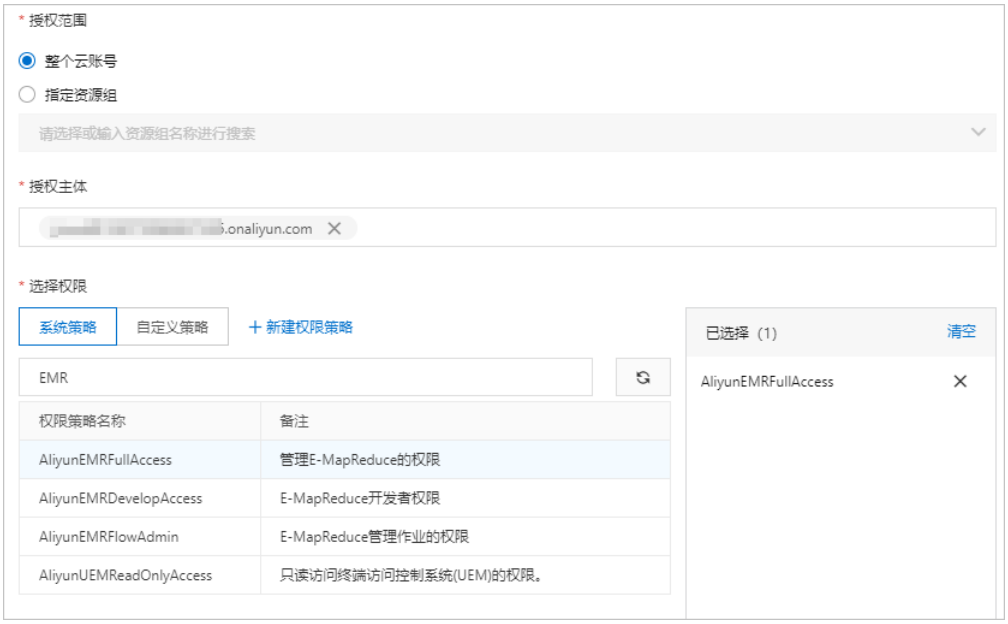
- 自定义策略

  如果您熟悉阿里云各种云服务API，并且需要精细化的权限控制策略，可参见权限策略语法和结构，创建自定义策略。创建时，您需要精准地设计权限策略脚本。

授权RAM用户

执行以下步骤，在访问控制RAM控制台，为RAM用户授予E-MapReduce相关权限：

1. 使用阿里云账号登录RAM控制台。
2. 在左侧导航栏，选择**身份管理 > 用户**。
3. 单击待授权RAM用户所在行的**添加权限**。
4. 在**添加权限**页面，根据需求选择授权范围、授权主体和权限。



参数	描述
授权范围	○ **整个云账号**：权限在当前阿里云账号内生效。 ○ **指定资源组**：权限在指定的资源组内生效。
授权主体	需要授权的RAM用户。
选择权限	在**系统策略**中，输入EMR搜索EMR相关权限策略，然后单击需要授予RAM用户的权限策略，选择该权限。各权限策略的详细说明，请参见权限策略。

5. 单击**确定**。
6. 单击**完成**。
   完成授权后，权限立即生效，被授权的RAM用户可以登录RAM控制台，执行已被授权的操作。

3.1.3.4. 管理用户

本文为您介绍如何通过E-MapReduce（简称EMR）的用户管理功能，管理集群中的EMR用户。

背景信息

EMR用户信息存储在集群自带的OpenLDAP中，主要用于E-MapReduce集群内的身份认证。

EMR用户可以用于访问链接与端口，查看开源组件Web UI时的用户身份认证，也可以在开启组件LDAP认证之后进行身份认证。如果将Ranger的用户源设置为LDAP，则可以对用户管理中的用户进行权限控制。如果是高安全集群，EMR用户可以用于Kinit操作。

用户管理中的EMR用户以列表的形式进行展示和操作。根据登录EMR控制台的RAM用户权限的不同，可以将使用E-MapReduce用户管理模块的RAM用户分为两类：

- 管理员：阿里云账号或者拥有emr:ManageUserPlatform（例如系统策略AliyunEMRFullAccess）和emr:CreateLdapUser权限的RAM用户。管理员可以查看集群中所有用户列表，并对所有用户执行重置密码、删除、修改备注、下载认证凭据（高安全集群）操作，也可以添加用户。
- 普通用户：拥有其他权限策略（例如系统策略AliyunEMRDevelopAccess）的RAM用户。普通用户只能在用户列表中查看与自己同名的EMR用户，并只能进行重置密码、修改备注和下载认证凭据（高安全集群）操作，不能添加和删除用户。

前提条件

- 已创建集群，详情请参见创建集群。
- 已创建RAM用户，详情请参见创建RAM用户。

> **说明** 因为在E-MapReduce用户管理中添加的用户需要与RAM用户同名，因此需要先创建RAM用户。

添加用户

> **注意** 如果当前用户使用的是RAM用户，则添加用户时需要该RAM用户拥有ram:ListUsers的权限。您可以使用阿里云账号在访问控制台上给该RAM用户添加AliyunRAMReadOnlyAccess系统策略，或者自定义权限策略并添加ram:ListUsers权限。

1. 进入用户管理页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群的**集群ID**。
   iv. 单击上方的**用户管理**页签。
2. 在**用户管理**页面，单击**添加用户**。
3. 在**添加用户**对话框中，在**用户名**下拉列表中，选择已有的RAM用户作为EMR用户的名称，输入**密码**和**确认密码**。
4. 单击**确定**。

删除用户

1. 进入用户管理页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群的**集群ID**。
   iv. 单击上方的**用户管理**页签。
2. 在**用户管理**页面，单击目标用户操作列的**删除**。
3. 在**删除用户**对话框中，单击**确定**。

重置用户密码

您可以修改已添加用户的密码。

> 注意　重置密码可能导致正在运行的任务失败。

1. 进入用户管理页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群的**集群ID**。
   iv. 单击上方的**用户管理**页签。
2. 在**用户管理**页面，单击目标用户操作列的**重置密码**。
3. 在**重置用户密码**对话框中，输入新的**密码**和**确认密码**。
4. 单击**确定**。

下载认证凭据

> 注意　下载认证凭据功能仅支持开启高安全的集群，通过该功能，您可以下载目标用户的Keytab文件。

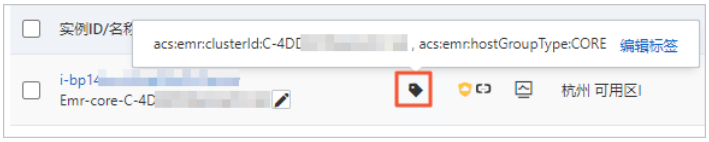
1. 进入用户管理页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群的**集群ID**。
   iv. 单击上方的**用户管理**页签。
2. 在**用户管理**页面，单击目标用户操作列的**下载认证凭据**。

3.1.3.5. 设置标签

标签是集群的标识。为集群添加标签，可以方便您识别和管理拥有的集群资源。您可以在创建集群时绑定标签，也可以在集群创建完成后，在集群基础信息页面绑定标签，您最多可以给集群绑定二十个标签。本文为您介绍如何新建、查看和绑定标签等。

背景信息

E-MapReduce集群中包含多个ECS节点。创建E-MapReduce集群时，系统会自动为ECS节点绑定两个系统标签。通过登录云服务器ECS控制台，在实例列表中将鼠标移动到对应ECS节点的标签图标上，可以查看该ECS节点属于哪个集群以及在集群中的角色。



例如，某个ECS节点上的系统标签如下：

- acs:emr:clusterId=C-A510C93EA117****
- acs:emr:hostGroupType=CORE

表示该节点是集群ID为*C-A510C93EA117****的EMR集群下的一个CORE节点。

注意事项

- 更新E-MapReduce的标签会同步到对应的ECS节点上。
- 更新ECS节点的标签不会同步到集群E-MapReduce上，因此为了保持ECS节点与E-MapReduce上标签的一致性，建议不要单独在ECS控制台上修改ECS的标签。并且当集群中某个ECS节点的标签数量达到上限时，集群将不能再创建标签。
- 不同地域中的标签信息不互通。

  例如，在华东 1（杭州）地域创建的标签在华东 2（上海）地域不可见。
- 一个资源绑定标签的上限为20个。如果超出上限，您需要解绑部分标签后再继续绑定新标签。

为集群创建或绑定标签

创建集群时进行新建或绑定已有标签的操作。

1. 登录EMR on ECS控制台。
2. 在顶部菜单栏处，根据实际情况选择地域和资源组。
3. 单击**创建集群**，在**基础配置**页面的**高级设置**区域，单击**标签**所在行的**添加标签**。

   创建集群详情，请参见创建集群。
4. 选择已有标签键和标签值，或输入新标签键和标签值。

   选择已有标签键和标签值意味着绑定标签，输入新标签键和标签值意味着创建标签。

   > **说明** 标签都是由一对键值对组成，标签键和标签值需满足以下约束信息：
   > - 标签键不可以为空，最长为128位；标签值可以为空，最长为128位。
   > - 标签键和标签值不能以aliyun或acs:开头，不能包含http://和https://。
   > - 任一标签的标签键（Key）必须唯一。例如，集群先绑定了city/shanghai，后续如果绑定city/newyork，则city/shanghai自动被解绑。

管理已有集群的标签

对已有集群进行新建、绑定和解绑标签操作。

1. 登录EMR on ECS控制台。
2. 在顶部菜单栏处，根据实际情况选择地域和资源组。
3. 在**集群管理**页面，选择目标集群所在行的 > **编辑标签**。
4. 在**编辑标签**对话框中，管理目标集群的标签。
   - 创建新标签。
     a. 单击上方的**绑定新建标签**。
     b. 输入**标签键**和**标签值**，单击**确定**。
     c. 单击**确定**。
   - 绑定已有标签。
     a. 单击上方的**已有标签**。
     b. 选择已有**标签键**和**标签值**。
     c. 单击**确定**。
   - 解绑标签。

     单击标签栏中每条标签后的图标，解绑该标签。

     > **说明** 解绑标签时，如果解绑之后该标签不再绑定任何资源，则该标签会自动被删除。

使用标签搜索集群

在集群管理页，按标签键和标签值搜索目标集群。

1. 登录EMR on ECS控制台。
2. 在顶部菜单栏处，根据实际情况选择地域和资源组。

3. 在**集群管理**页面的**标签选择**下拉列表中，选择一个标签键和标签值。

   下方列表会为您展示满足搜索条件的集群。

   > **说明** 如果您未选择具体的标签值，默认展示该标签键绑定的所有集群。

3.1.3.6. 使用资源组

资源组会对您拥有的云资源从用途、权限和归属等维度上进行分组，实现企业内部多用户、多项目的资源分级管理。一个云资源只能属于一个资源组，云资源之间的关联关系不会因加入资源组而发生变化。E-MapReduce目前支持资源组的云资源为集群和项目。本文介绍如何为云资源指定资源组及相关的示例。

背景信息

使用资源组时，您需要了解：

- 一个资源组可以包含不同地域的云资源。例如，资源组A中可以包含华东1（杭州）地域的集群和华东2（上海）地域的集群。
- 一个资源组可以包含不同资源类型。例如，资源组A中可以包含集群、ECS实例和项目等多种云资源。
- 同一个账号内不同资源组中，相同地域的集群和项目可以进行关联。如果当前账号同时具有资源组A和资源组B的相关权限，则资源组A中华北2（北京）地域项目的作业可以运行在资源组B中华北2（北京）地域的集群上。
- 资源组会继承RAM用户的全局权限。即如果您授权RAM用户管理所有的阿里云资源，那么阿里云账号下所有的资源组都会在该RAM用户中显示出来。

使用限制

- 资源组的创建、管理和RAM授权都是在阿里云资源管理（Resource Management）控制台上进行，详情请参见什么是资源管理。
- E-MapReduce支持资源组的云资源为集群和项目。在创建、扩容集群或转移集群资源组时，集群每个节点会同步加入集群所属资源组，节点支持资源组的云资源包括ECS实例、云盘、镜像、弹性网卡、安全组和密钥对。
- 不允许跨账号在资源组之间转移云资源。

> **注意** 转移节点相关资源的资源组不会同步到集群上，为了保持集群相关资源的统一管理和RAM授权，不建议在资源管理控制台上单独对节点相关资源（例如，ECS实例、云盘、镜像、弹性网卡、安全组和密钥对）转移所属资源组。

指定资源组

云资源在创建之后必须属于一个资源组，如果在创建时未指定，则会加入默认的资源组。下面介绍如何在创建集群和项目时指定资源组。

> **说明** 每个账号在创建云资源时只能将云资源加入有相关权限的资源组中。

1. 登录EMR on ECS控制台。
2. 在顶部菜单栏处，根据实际情况选择地域和资源组。
3. 单击**创建集群**。
4. 完成软硬件配置后，在**基础配置**页面，展开**高级设置**，在**资源组**区域中选择已有的资源组。

   如果需要创建新的资源组，您可以单击下方的**创建资源组**，详情请参见创建资源组。

   > **说明** 创建集群的详细信息，请参见创建集群。

应用场景

资源组有以下两个应用场景：

- 场景一：将不同用途的云资源加入不同的资源组中分开管理。
- 场景二：为各个资源组设置完全独立的管理员，实现资源组范围内的用户与权限管理。

> **说明** 由于授权原因RAM用户仅能看到被授权的资源，因此对于只有部分资源组权限而没有全局权限的RAM用户，在顶部菜单栏选择**账号全部资源**时会报无权限的提示。

场景一：按云资源的用途分组

您可以将生产环境和测试环境的集群，分别放入生产环境和测试环境两个资源组中。产品测试时，建议只对测试环境资源组中的集群进行操作，避免对生产环境的集群发生误操作。当产品需要上线时，再选择生产环境资源组中的集群进行操作。

1. 创建资源组**测试环境**和**生产环境**。

   详情请参见创建资源组。

2. 为资源组**测试环境**和**生产环境**设置同一个管理员。

   详情请参见添加RAM身份并授权。

3. 创建集群TestEnv1和TestEnv2。

   创建集群时，请指定加入**测试环境**资源组。

4. 创建集群ProdEnv1和ProdEnv2。

   创建集群时，请指定加入**生产环境**资源组。

5. 使用**测试环境**和**生产环境**资源组的管理员账号登录EMR on ECS控制台。

6. 在顶部菜单栏处选择相应资源组。

   您可以看到相应资源组的集群都显示在集群列表页面。例如，选择**测试环境**，您只可以看到测试环境的集群显示在集群列表页面。

场景二：资源组范围内的用户与权限管理

您可以将公司不同部门使用的集群和项目分别放入多个资源组中，并设置相应的管理员分部门管理集群和项目，实现资源组的隔离。本示例以**开发部**和**测试部**进行介绍。

1. 创建资源组**开发部**和**测试部**。

   详情请参见创建资源组。

2. 分别为资源组**开发部**和**测试部**设置管理员。

   详情请参见添加RAM身份并授权。

3. 创建集群ITCluster，在创建时指定**开发部**资源组。

   集群创建的详细信息请参见创建集群。

4. 创建集群FinanceCluster1，在创建时指定**测试部**资源组。

5. 使用测试部管理员账号登录EMR on ECS控制台。

6. 在顶部菜单栏处选择**测试部**。

   您可以看到属于测试部的集群都显示在集群列表页面。

3.1.3.7. 管理安全组

安全组用于设置集群内ECS实例的网络访问控制，是重要的安全隔离手段。本文为您介绍如何添加安全组及安全组规则。

背景信息

您在创建E-MapReduce集群时，可以使用已有的安全组或者新建安全组，对某个安全组下的所有ECS实例的出方向和入方向进行网络控制。您可以将ECS实例按照功能划分，放于不同的安全组中。例如，通过E-MapReduce创建的安全组为E-MapReduce安全组，而您已有的安全组为用户安全组，每个安全组按照不同的需要设置不同的访问控制策略。

新建安全组详情，请参见创建安全组。

使用限制

- 经典网络类型下，实例必须加入同一地域下经典网络类型的安全组。
- 专有网络类型下，实例必须加入同一专有网络下的安全组。

注意事项

- 添加安全组规则时，一定要限制访问IP地址范围，且不要使用0.0.0.0/0，避免被攻击。
- 添加安全组规则时，开放应用出入规则应遵循最小授权原则，授权对象只针对当前服务器的公网访问IP地址开放。您可以通过访问IP地址，获取当前服务器的公网访问IP地址。
- 禁止使用在ECS上创建的企业安全组。

添加安全组

1. 进入节点管理页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群所在行的**节点管理**。
2. 进入安全组列表页面。
   i. 在**节点管理**页面，单击机器组前面的+图标。
   ii. 单击ECS实例的ECS ID。
   iii. 在该ECS实例页面，单击上方的**安全组**页签。
3. 在**安全组列表**页面，单击**加入安全组**。
4. 在**ECS实例加入安全组**对话框中，从**安全组**列表中选择需要加入的安全组。

   如果您需要将该ECS实例一次加入多个安全组，选择一个安全组后，单击后面的**加入到批量选择栏**，即会把该安全组加入到批量选择栏中，依次按照相同方法再选择其他安全组，把其他安全组也加入到批量选择栏中即可。

5. 单击**确定**。

   请重复步骤2~步骤4操作，直至把E-MapReduce集群中的其他ECS实例也加入到相应安全组中。

添加安全组规则

1. 获取机器的公网访问IP地址。

   为了安全地访问集群组件，在设置安全组策略时，推荐您只针对当前的公网访问IP地址开放。您可以通过访问IP地址，获取当前服务器的公网访问IP地址。

2. 进入安全组页面。

   i. 登录EMR on ECS控制台。

   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。

   iii. 在**集群管理**页面，单击目标集群的集群ID。

   iv. 在**基础信息**区域，单击默认安全组。

3. 在**安全组规则**页面的**访问规则**区域，单击**手动添加**，填写安全组策略。

   填写**端口范围**和**授权对象**，其余参数保持默认。详细信息，请参见添加安全组规则。

参数	说明
端口范围	填写允许访问该ECS实例的端口。
授权对象	填写步骤1中获取的公网访问IP地址。 **注意** 为防止被外部的用户攻击导致安全问题，**授权对象**禁止填写为**0.0.0.0/0**。

4. 单击**保存**。

3.1.3.8. 管理元数据

3.1.3.8.1. 数据湖元数据

本文介绍如何配置数据湖构建（Data Lake Formation，DLF），作为E-MapReduce（简称EMR）上Hadoop集群的元数据。

背景信息

阿里云数据湖构建是一款全托管的快速帮助用户构建云上数据湖的服务，产品为云原生数据湖提供了统一的元数据管理、统一的权限与安全管理、便捷的数据入湖能力以及一键式数据探索能力，详细信息请参见数据湖构建产品简介。

您可以快速完成云原生数据湖方案的构建与管理，并可无缝对接多种计算引擎，打破数据孤岛，洞察业务价值。

前提条件

已在数据湖构建（Data Lake Formation）控制台开通数据湖构建。

使用限制

- 数据湖元数据适配EMR的Hive 2.x、Hive 3.x、Presto和SparkSQL。
- 仅EMR-3.30.0及之后版本和EMR-4.5.0及之后版本，支持选择数据湖元数据作为Hive数据库。
- 数据湖元数据产品支持华北2（北京）、华东1（上海）、华东2（杭州）和华南1（深圳）地域。

切换元数据存储类型

您可以通过修改Hive参数的方式，切换Hive MetaStore的存储方式。

说明 如果需要迁移数据库的元数据信息，请提交工单处理。

1. 进入Hive服务页面。

   i. 登录EMR on ECS控制台。

   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。

   iii. 在**集群管理**页面，单击目标集群所在行的**集群服务**。

   iv. 单击Hive服务区域的**配置**

2. 在配置搜索中，输入配置项**hive.imetastoreclient.factory.class**，单击图标。

   - 切换为MySQL（包括集群内置MySQL、统一meta数据库和独立RDS MySQL）：

设置hive.imetastoreclient.factory.class的值为org.apache.hadoop.hive.ql.metadata.SessionHiveMetaStoreClientFactory。

- 切换为数据湖元数据：

  设置hive.imetastoreclient.factory.class的值为com.aliyun.datalake.metastore.hive2.DlfMetaStoreClientFactory。

3. 保存配置。
   i. 在Hive服务页面的**服务配置**区域，单击**保存**。
   ii. 在**保存**对话框中，输入执行原因，单击**保存**。
4. 重启Hive MetaStore服务。
   i. 在Hive服务页面，选择右上角的**更多操作 > 重启**。
   ii. 在弹出的对话框中，填写**执行原因**，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

   您可以单击上方的**操作历史**，查看任务进度，等待任务完成。

3.1.3.8.2. 配置独立RDS

本文介绍如何配置独立的阿里云RDS，作为E-MapReduce（简称EMR）上Hadoop或EMR Studio集群的元数据。

背景信息

EMR Studio正处于公测阶段，如果您需要体验，请提交工单。更多EMR Studio信息，请参见EMR Studio概述。

前提条件

已购买RDS MySQL实例，详情请参见创建RDS MySQL实例。

> **说明** 本文以MySQL 5.7版本为例介绍。

使用限制

EMR上创建的集群类型与MySQL关系如下：

- 如果创建的是Hadoop集群，则数据库**类型**选择**MySQL**，版本选择5.7；**系列**选择**高可用版**。
- 如果创建的是EMR Studio集群，则数据库**类型**选择**MySQL**，版本选择8.0。

操作流程

1. 步骤一：元数据库准备

   准备元数据库信息。
2. 步骤二：创建集群

   在EMR控制台上创建集群，关联元数据库信息。
3. （可选）步骤三：Metastore初始化

   根据Hive版本初始化Metastore。

   > **注意** 如果您创建的是Hadoop集群，则需要执行该步骤。

步骤一：元数据库准备

1. 创建hivemeta的数据库。

   详情请参见创建数据库和账号中的创建数据库。

2. 创建用户并授权读写权限。

> 注意
> - 如果您创建的是Hadoop集群，则可以创建普通账号，详情请参见创建数据库和账号。
> - 如果您创建的是EMR Studio集群，则需要创建高级权限账号，详情请参见创建数据库和账号。

请记录创建账号的用户名和密码，步骤二：创建集群会用到。

3. 获取数据库内网地址。
   i. 设置白名单，详情请参见通过客户端、命令行连接RDS MySQL。
   ii. 在实例详细页面，单击左侧导航栏中的**数据库连接**。
   iii. 在**数据库连接**页面，单击内网地址进行复制。

请记录内网地址，步骤二：创建集群时会用到。

步骤二：创建集群

在创建集群的**基础配置**页面，配置以下参数，其他参数的配置请参见创建集群。

参数	描述
集群名称	集群的名字，长度限制为1~64个字符，仅可使用中文、字母、数字、中划线（-）和下划线（_）。
元数据选择	选择**独立RDS MySQL**。
数据库链接	**数据库连接**填写格式为*jdbc:mysql://rm-xxxxxx.mysql.rds.aliyuncs.com/<数据库名称>?createDatabaseIfNotExist=true&characterEncoding=UTF-8*。 • *rm-xxxxxx.mysql.rds.aliyuncs.com*为步骤一：元数据库准备中获取的数据库内网地址。 • *<数据库名称>*为步骤一：元数据库准备中设置的数据库名称。
数据库用户名	填写步骤一：元数据库准备中账号的用户名。
数据库密码	填写步骤一：元数据库准备中账号的密码。
数据开发存储	设置Airflow的logs、dags以及Zeppelin的notebook在OSS上的存储位置。 **说明** 该参数仅在创建EMR Studio集群时可见。

步骤三：Metastore初始化

注意 如果您创建的是Hadoop集群，则需要按照以下步骤根据Hive版本初始化Metastore。

1. 进入集群详情页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群的集群ID。
2. 在**集群基础信息**页面的**软件信息**区域，检查Hive的版本，并进行初始化。
   - 如果Hive是2.3.x版本，请按照以下步骤进行初始化。
     a. 使用ssh方式登录集群Master节点，详情请参见登录集群。
     b. 执行以下命令，进入mysql目录。
     ```
     cd /usr/lib/hive-current/scripts/metastore/upgrade/mysql/
     ```
     c. 执行以下命令，登录MySQL数据库。
     ```
     mysql -h {RDS数据库内网或外网地址} -u{RDS用户名} -p{RDS密码}
     ```
     d. 执行以下命令，进行初始化。
     ```
     use {RDS数据库名称};
     source /usr/lib/hive-current/scripts/metastore/upgrade/mysql/hive-schema-2.3.0.mysql.sql;
     ```
     上述命令中参数描述如下：
     - `{RDS用户名}` 为步骤一：元数据库准备中账号的用户名。
     - `{RDS密码}` 为步骤一：元数据库准备中账号的密码。
     - `{RDS数据库名称}` 为步骤一：元数据库准备中设置的数据库名称。
   - 如果Hive是其他版本时，执行以下命令进行初始化。
     a. 使用SSH方式登录集群的Master节点，详情请参见登录集群。
     b. 执行以下命令，切换为hadoop用户。
     ```
     su hadoop
     ```
     c. 执行以下命令，登录MySQL数据库。
     ```
     schematool -initSchema -dbType mysql
     ```
     待初始化成功后，则可以使用自建的RDS作为Hive的元数据库。

     说明 在初始化之前，Hive的Hive MetaStore、HiveServer2和Spark的ThriftServer可能会出现异常，待初始化之后会恢复正常。

常见问题

- Metastore初始化时提示Failed to get schema version异常信息，该如何处理？
- 如果Hive元数据信息中包含中文信息，例如列注释和分区名等，该如何处理？

3.1.3.9. 脚本操作

3.1.3.9.1. 管理引导操作

添加引导操作，可以安装您需要的第三方软件或者修改集群运行环境。本文为您介绍如何添加引导操作及其示例。

背景信息

引导操作功能可以在集群扩容或弹性伸缩时自动在新增节点上运行指定脚本。手动执行功能，可以批量选择在已有节点上运行指定脚本，以实现个性化需求，手动执行脚本的详情，请参见手动执行脚本。

引导操作类似手动执行，在集群创建时或者创建完成后，您可以通过引导操作功能，完成很多目前E-MapReduce集群尚未支持的操作，例如：

- 使用Yum安装已经提供的软件。
- 直接下载公网上的一些公开的软件。
- 读取OSS中您的自有数据。
- 安装并运行一个服务，例如Flink或者Impala。

使用限制

- 您最多可以添加10个引导操作。添加的引导操作会按照您指定的顺序执行。
- 您指定的脚本默认使用root账户执行，您也可以在脚本中使用**su hadoop**命令，切换为hadoop用户执行。

添加引导操作

添加引导操作支持以下两种方式。

1. 进入集群管理页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
2. 单击右上角的**创建集群**。
3. 在**基础配置**的**高级设置**区域，单击**引导操作**所在行的**添加引导操作**。
4. 填写以下配置项。

参数	描述
操作名称	引导操作的名称。
脚本地址	选择脚本所在OSS的路径。 脚本路径格式必须为*oss://**/*.sh*格式。
参数	引导操作脚本的参数，指定脚本中所引用的变量的值。
执行时间	○ **组件启动前**：组件启动前执行该脚本。 ○ **组件启动后**：组件启动后执行该脚本。
执行失败策略	○ **继续执行**：如果该脚本执行失败，继续执行下一个脚本。 ○ **停止执行**：如果该脚本执行失败，停止当前脚本。
执行范围	○ **集群**：此引导操作适用于整个集群。 ○ **机器组**：此引导操作适用于您选择的机器组。 您可以选择主实例、核心实例组或者已有的机器组。

引导操作示例请参见示例。

> **说明** 引导操作可能会执行失败，但引导操作失败并不会影响集群的创建。

创建集群详情，请参见创建集群。集群创建成功后，您可以在集群的**基础信息**页面，查看**引导操作/软件配置**是否有异常发生。如果有异常，您可以登录到各个节点上查看运行日志，运行日志在*/var/log/bootstrap-actions*目录下。

1. 进入脚本操作页面。

i. 登录EMR on ECS控制台。
ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
iii. 在**集群管理**页面，单击目标集群的集群ID。
iv. 单击上方的**脚本操作**页签。

2. 在**引导操作**页面，单击**添加引导操作**。
   - 编辑引导操作：单击目标引导操作所在行的**编辑**。
   - 克隆引导操作：单击目标引导操作所在行的**克隆**。
   - 删除引导操作：单击目标引导操作所在行的**删除**。
3. 在**添加引导操作**对话框中，填写配置项。

参数	描述
名称	引导操作的名称。
脚本位置	选择脚本所在OSS的位置。 脚本路径格式必须为*oss://**/*.sh*格式。
参数	引导操作脚本的参数，指定脚本中所引用的变量的值。
执行范围	- **集群**：此引导操作适用于整个集群。 - **机器组**：此引导操作适用于您选择的机器组。 您可以选择主实例、核心实例组或者已有的机器组。
执行时间	- **组件启动前**：组件启动前执行该脚本。 - **组件启动后**：组件启动后执行该脚本。
执行失败策略	- **继续执行**：如果该脚本执行失败，继续执行下一个脚本。 - **停止执行**：如果该脚本执行失败，停止当前脚本。

4. 单击**确定**。
   引导操作示例请参见示例。

方式一：创建集群时添加引导操作

方式二：创建集群后添加引导操作

示例

引导操作需要指定引导操作的名称和执行脚本在OSS上的位置，根据需要指定脚本的参数。执行引导操作时各个节点会下载您指定的OSS脚本，直接执行或者附加上可选参数执行。引导操作示例如下：

- 示例1

  您可以在脚本中指定需要从OSS下载的文件。例如，添加以下脚本，将*oss://<yourBucket>/<myFile>.tar.gz*文件下载到本地，并解压到*/<yourDir>*目录下。

  > 注意　OSS地址有内网地址、外网地址和VPC网络地址之分。如果使用经典网络，则需要指定内网地址，例如杭州是*oss-cn-hangzhou-internal.aliyuncs.com*。如果使用VPC网络，则需要指定VPC内网可访问的域名，例如杭州是*vpc100-oss-cn-hangzhou.aliyuncs.com*。

```
#!/bin/bash
osscmd --id=<yourAccessKeyId> --key=<yourAccessKeySecret> --host=oss-cn-hangzhou-internal.aliyuncs.com get oss://<yourBucketName>/<yourFile>.tar.gz ./<yourFile>.tar.gz
mkdir -p /<yourDir>
tar -zxvf <yourFile>.tar.gz -C /<yourDir>
```

- 示例2

  您可以通过Yum安装额外的系统软件包，例如安装*ld-linux.so.2*。

```
#!/bin/bash
yum install -y ld-linux.so.2
```

3.1.3.9.2. 手动执行脚本

集群创建完成后，您可以通过手动执行脚本功能批量选择节点来运行指定脚本，以实现个性化需求。本文为您介绍如何添加手动执行脚本。

背景信息

手动执行功能适用于长期存在的集群，对按需创建的临时集群，应使用引导操作来完成集群初始化工作。引导操作详情，请参见管理引导操作。

手动执行类似引导操作，在集群创建完成后，您可以通过手动执行功能来安装集群尚未支持的软件和服务，例如：

- 使用YUM安装已经提供的软件。
- 直接下载公网上公开的软件。
- 读取您OSS中的自有数据。
- 安装并运行服务（例如，Flink或者Impala），但需要编写的脚本会复杂些。

前提条件

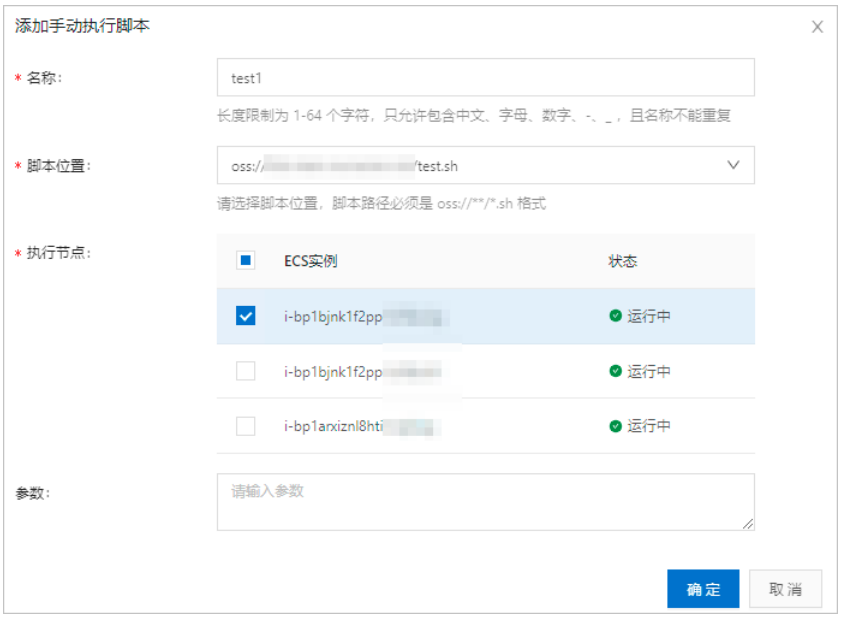
- 已创建集群，详情请参见创建集群。
- 请确保集群状态处于运行中，其他状态时集群不支持运行集群脚本。
- 已开发或已获取集群脚本（示例），并上传到OSS。

注意事项

- 一个集群同一时间只能运行一个集群脚本，如果有正在运行的集群脚本，则无法再提交执行新的集群脚本。每个集群最多保留10个集群脚本记录，如果超过10个，则需要您将之前的记录删除才能创建新的集群脚本。
- 集群脚本可能在部分节点上运行成功，部分节点上运行失败。例如，节点重启导致脚本运行失败。在解决异常问题后，您可以单独指定失败的节点再次运行。当集群扩容后，您也可以指定扩容的节点单独运行集群脚本。
- 集群脚本会在您指定的节点上下载OSS上的脚本并运行，如果运行状态是失败，则您可以登录到各个节点上查看运行日志，运行日志存储在每个节点的*/var/log/cluster-scripts/clusterScriptId*目录下。如果集群配置了OSS日志目录，运行日志也会上传到*osslogpath/clusterId/ip/cluster-scripts/clusterScriptId*目录。

操作步骤

1. 进入脚本操作页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群的集群ID。
   iv. 单击上方的**脚本操作**页签。
2. 在**集群脚本**页面，单击上方的**手动执行**。
3. 单击上方的**创建并执行**。
4. 在**创建脚本**对话框中，输入**名称**，在**脚本位置**下拉列表中选择脚本位置，选中**执行节点**，输入自定义参数。



> 说明
> - 在使用集群脚本功能时，强烈建议您先在单个节点对集群脚本进行测试，待测试全部正常后，再在整个集群上操作。
> - 脚本路径格式必须是*oss://**/*.sh*格式。

5. 完成上述参数配置后，单击**确定**。
   集群脚本创建完成后，会显示在集群脚本列表中，并且脚本处于运行中状态。脚本状态包括运行中、运行完成和提交失败。
   - 单击**详情**，可以查看手动执行脚本的详情。
     脚本对应节点的状态包括等待中、运行中、完成、失败、提交失败和取消。
   - 单击**删除**，可以删除手动执行脚本。

示例

与引导操作的脚本相似，您可以在集群脚本中指定需要从OSS下载的文件。下载示例文件*oss://<yourBucket>/<myFile>.tar.gz*到本地，并解压到*/yourDir*目录下。

```
#!/bin/bash
osscmd --id=<yourAccessKeyId> --key=<yourAccessKeySecret> --host=oss-cn-hangzhou-internal.aliyuncs.com get oss://<yourBucke
tName>/<yourFile>.tar.gz ./<yourFile>.tar.gz
mkdir -p /<yourDir>
tar -zxvf <yourFile>.tar.gz -C /<yourDir>
```

> 说明 OSS地址有内网地址、外网地址和VPC网络地址之分。如果是经典网络，则需要指定内网地址（例如，杭州是*oss-cn-hangzhou-internal.aliyuncs.com*）。如果是VPC网络，则需要指定VPC内网可以访问的域名（例如，杭州是*vpc100-oss-cn-hangzhou.aliyuncs.com*）。

集群脚本也可以通过YUM安装额外的系统软件包。例如安装*ld-linux.so.2*。

```
#!/bin/bash
yum install -y ld-linux.so.2
```

集群默认使用root用户来执行您指定的脚本。您也可以在脚本中使用`su hadoop`切换到hadoop账户。

3.1.3.10. 查看集群列表

本文介绍如何查看您账号下拥有的集群概况和单个集群的详情。

前提条件

已创建集群，详情请参见创建集群。

查看集群列表

1. 登录EMR on ECS控制台。
2. 在顶部菜单栏处，根据实际情况选择地域和资源组。
3. 在EMR on ECS的集群管理页面，集群列表展示您所拥有的所有集群的基本信息，以及各集群支持的操作。

参数	描述
集群ID/名称	集群的ID以及名称。
集群类型	当前集群的类型。
状态	集群的状态。
创建时间/运行时间	○ 创建时间：显示集群创建的时间。 ○ 运行时间：从创建开始到目前的运行时间。集群一旦被释放，计时终止。
付费类型	集群的付费类型。
集群标签	集群的标识。

参数	描述
操作	集群支持的操作： ○ **集群服务**：进入集群服务的页面，可以查看服务和管理服务。 ○ **节点管理**：进入节点管理页面，可以查看机器组信息，进行扩容集群和扩容磁盘等操作。 ○ **释放**：选择目标集群操作列的 > **释放**。 仅按量付费的集群支持释放，释放集群详细信息，请参见释放集群。 ○ **续费**和**自动续费**：选择目标集群操作列的 > **续费**或**自动续费**。 仅按量付费的集群支持续费和自动续费，详细信息请参见续费流程（新版控制台）。

查看集群详情

1. 进入集群基础信息页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在EMR on ECS的集群管理页面，单击目标集群的**集群ID**。
2. 在集群基础信息页面查看集群的详情。

   基础信息页面为您展示集群的详细信息，包括集群信息、软件信息、网络信息和监控数据四部分。

   ○ 集群信息

   该区域信息与您创建集群时设置的参数有关，所以具体参数以控制台展示的为准。

参数	描述
集群名称	集群的名称。
集群ID	集群的实例ID。
默认安全组	集群加入的安全组的ID。
当前状态	当前集群的状态。
付费类型	当前集群的付费类型。
开始时间	集群创建的时间。
运行时间	集群的运行时间。
ECS应用角色	列出了ECS应用角色。 当用户程序在EMR计算节点上运行时，可不填写阿里云的AccessKey来访问相关的云服务（例如OSS），EMR会自动申请一个临时AccessKey来授权本次访问。ECS应用角色用于控制这个AccessKey的权限。
高可用	是否开启了高可用。
Kerberos认证	集群中的软件以Kerberos安全模式启动。
元数据类型	当前集群的元数据类型。
标签	集群的标签，可以编辑标签。编辑标签详情，请参见设置标签。

   ○ 软件信息

   该区域为您展示信息如下表。

软件信息	说明
集群版本	使用的E-MapReduce的版本。
集群类型	当前集群的类型。
软件信息	展示用户安装的所有的应用程序及其版本，例如，HDFS 3.2.1、Hive 3.2.1和Hive 3.1.2。

   ○ 网络信息

该区域为您展示信息如下表。

网络信息	说明
网络类型	默认vpc。
专有网络	当前集群专有网络的ID。

- 监控数据

  该区域为您展示信息如下表。

监控项	描述
CPU Idle	CPU空闲率。
CPU User	用户态CPU使用率。
CPU System	系统态CPU使用率。
MEM Free	空闲内存容量。
MEM Used Percent	已使用的内存容量的百分比。
Load One Minute	平均每分钟负载。
Disk Partition Capacity Max Used	已使用的最大磁盘分区容量。
Disk Partition Utilization Max	最大磁盘分区利用率。
Total Processes	总进程或线程数目。
Num Processes Running	运行中的进程或线程数目。
Num Processes Blocked	阻塞的进程或线程数目。
Num Processes Created	创建的进程或线程数目。

3.1.4. 服务管理

3.1.4.1. 管理服务

创建E-MapReduce集群时，不同类型集群的实例节点上会部署不同的服务。集群创建成功并正常运行后，您可以管理集群服务，包括新增、配置、启动、停止或重启服务。本文为您介绍如何新增和重启集群服务。

前提条件

已创建集群，详情请参见创建集群。

注意事项

- 为确保服务重启过程中，尽量减少或不影响业务运行，可以通过滚动重启服务。对于有主备状态的实例，会先重启备实例，再重启主实例。
- 滚动执行关闭后，所有节点同时重启可能导致服务不可用，请谨慎选择。

新增服务

1. 进入集群服务页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
2. 在**集群服务**页签，单击**新增服务**。
3. 在**新增服务**对话框中，选择待添加的服务，单击**确定**。
   服务新增成功后，系统提示**新增服务成功**。您可以在当前页面查看新增的服务。

重启服务

配置项修改后，需要重启对应的服务使配置生效。您还可以在**集群服务**页签，配置、启动或停止服务。

1. 进入集群服务页面。
   i. 登录EMR on ECS控制台。

ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
2. 在**集群服务**页签，选择目标服务区域的 **... > 重启**。
3. 在重启服务对话框中，填写执行原因，单击**确定**。
可修改参数说明如下。

参数	描述
滚动执行	默认开启。开启后，系统会按照**每批间隔时间**执行任务。您可以设置**每批间隔时间**，即每隔多长时间执行一次任务。
失败处理策略	○ **单节点失败继续执行**：执行任务时，如果单节点检查失败，继续其他节点的执行任务。 ○ **单节点失败终止任务**：执行任务时，如果单节点检查失败，终止执行任务。
执行原因	执行任务的原因。

4. 在**确认**对话框中确认无误后，单击**确定**。
确定后，系统提示**执行操作成功**。系统会按照您配置的策略执行对应任务。

3.1.4.2. 回滚配置

E-MapReduce支持在控制台对已进行过修改的服务配置进行回滚。本文为您介绍如何通过控制台回滚各服务的配置。

前提条件

已对服务配置进行过修改操作。

操作步骤

1. 进入集群服务页面。
i. 登录EMR on ECS控制台。
ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
2. 在左侧导航栏，选择**集群服务 > HDFS**。
本文以HDFS服务为例。
3. 单击HDFS服务区域的**配置**。
4. 在**配置**页签，单击**配置修改历史**。
该页面会显示组件的配置历史。
5. 在**配置修改历史**页面，单击待回滚配置操作列的**回滚**。

6. 在**回滚**对话框中，单击**确定**。

3.1.4.3. 管理配置项

E-MapReduce提供控制台的方式修改或添加HDFS、YARN和Spark等服务的配置项。本文为您介绍如何在E-MapReduce控制台修改或添加配置项。

前提条件

已创建集群，详情请参见创建集群。

修改配置项

1. 进入服务的配置页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 在**集群服务**页面，单击目标服务区域的**配置**。
2. 修改配置。
   i. 在**配置过滤**中，输入待修改的配置项，单击图标。
   ii. 找到您修改的参数后，修改对应的参数值。
3. 保存配置。
   i. 在**服务配置**区域，单击**保存**。
   ii. 在**确认修改配置**对话框中，输入**执行原因**，打开**自动更新配置**开关。
   iii. 单击**确定**。
4. 生效配置。

   请根据您修改的参数类型执行以下操作，使修改的配置生效。
   - 客户端类型配置
     a. 如果修改的参数类型为客户端类型配置，修改完成后，单击**服务配置**区域的**部署客户端配置**。
     b. 在弹出的对话框中，输入**执行原因**，单击**确定**。
     c. 在**确认**对话框中，单击**确定**。
   - 服务端类型配置
     a. 如果修改的参数类型为服务端类型配置，修改完成后，在**配置**页面，选择**更多操作 > 重启**。
     b. 在弹出的对话框中，输入**执行原因**，单击**确定**。
     c. 在**确认**对话框中，单击**确定**。

添加配置项

1. 进入服务的配置页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 在**集群服务**页面，单击目标服务区域的**配置**。
2. 新增配置。
   i. 在目标服务的配置页面的**服务配置**区域，单击待操作的页签。
   ii. 单击上方的**新增配置项**。
   iii. 根据您的实际情况，添加配置项。

   一次可以添加多个配置项。

配置项	描述
Key	参数名。
Value	参数值。
描述	参数描述。
操作	支持删除配置项。

   iv. 新增完成后，单击**确定**。
   v. 在**确认修改配置**对话框中，输入执行原因，打开**自动更新配置**开关。
3. 生效配置。

   请根据您修改的参数类型执行以下操作，使修改的配置生效。
   - 客户端类型配置
     a. 如果修改的参数类型为客户端类型配置，修改完成后，单击**服务配置**区域的**部署客户端配置**。
     b. 在弹出的对话框中，输入**执行原因**，单击**确定**。

c. 在**确认**对话框中，单击**确定**。

- 服务端类型配置
  a. 如果修改的参数类型为服务端类型配置，修改完成后，在**配置**页面，选择**更多操作 > 重启**。
  b. 在弹出的对话框中，输入**执行原因**，单击**确定**。
  c. 在**确认**对话框中，单击**确定**。

3.1.4.4. 配置自定义软件

Hadoop和Hive等软件含有大量的配置，当您需要对其软件配置进行修改时，可以在创建集群时通过软件自定义配置功能实现。本文为您介绍如何配置自定义软件。

使用限制

软件配置操作仅在集群创建时执行一次。

操作步骤

1. 进入集群管理页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
2. 在**集群管理**页面，单击**创建集群**。
3. 在**软件配置**的**高级设置**区域，开启**软件自定义配置**开关。

您可以添加JSON格式的配置文件，在创建集群过程中覆盖或添加集群服务的默认参数。JSON文件的内容示例如下。

```
[
    {
        "ApplicationName":"YARN",
        "ConfigFileName":"yarn-site",
        "ConfigItemKey":"yarn.nodemanager.resource.cpu-vcores",
        "ConfigItemValue":"8"
    },
    {
        "ApplicationName":"YARN",
        "ConfigFileName":"yarn-site",
        "ConfigItemKey":"aaa",
        "ConfigItemValue":"bbb"
    }
]
```

- 各参数含义如下表所示。

参数	描述
ApplicationName	服务名，需要全部大写。
ConfigFileName	文件名称，实际传参的文件名称，需要去掉后缀。
ConfigItemKey	配置项的名称。
ConfigItemValue	该配置项要设置的具体的值。

- 各服务的配置文件如下表所示。

服务	配置文件

服务	配置文件
YARN	■ core-site.xml ■ log4j.properties ■ hdfs-site.xml ■ mapred-site.xml ■ yarn-site.xml ■ httpsfs-site.xml ■ capacity-scheduler.xml ■ hadoop-env.sh ■ httpfs-env.sh ■ mapred-env.sh ■ yarn-env.sh
Hive	■ hive-env.sh ■ hive-site.xml ■ hive-exec-log4j.properties ■ hive-log4j.properties

集群组件的参数配置好后，您可以继续创建集群，详情请参见创建集群。

3.1.4.5. 访问Web UI

3.1.4.5.1. 通过SSH隧道方式访问开源组件Web UI

本文为您介绍如何通过SSH隧道方式访问开源组件的Web UI。

背景信息

在E-MapReduce集群中，为保证集群安全，Hadoop、Spark和Flink等开源组件的Web UI的端口均未对外开放。您可以通过控制台的方式访问Web UI，也可以通过在本地服务器上建立SSH隧道以端口转发的方式来访问Web UI，端口转发方式包括端口动态转发和本地端口转发两种。

如果您需要通过控制台的方式访问Web UI时，请参见访问链接与端口。

前提条件

- 已创建集群，详情请参见创建集群。
- 确保本地服务器与集群主节点网络连通。您可以在创建集群时打开**挂载公网**开关，或者在集群创建好之后在ECS控制台上为主节点挂载公网，为主节点ECS实例分配固定公网IP或EIP，详情请参见绑定弹性网卡。

获取主节点的节点名称和公网IP地址

1. 进入节点管理页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**节点管理**。
2. 单击主实例组所在行的+图标。
3. 在**节点名称**和**公网IP**列，可以查看主节点的节点名称和公网IP地址。

使用动态端口转发方式

创建从本地服务器开放端口到集群主节点的SSH隧道，并运行侦听该端口的本地SOCKS代理服务器，端口的数据会由SSH隧道转发到集群主节点。

1. 创建SSH隧道。
   - 密钥方式

     ```
     ssh -i <密钥文件路径> -N -D 8157 root@<主节点公网IP地址>
     ```

   - 密码方式

     ```
     ssh -N -D 8157 root@<主节点公网IP地址>
     ```

     相关参数描述如下：
     - `8157`：本地服务器端口以`8157`为例，实际配置时，您可使用本地服务器未被使用的任意一个端口。
     - `-D`：使用动态端口转发，启动SOCKS代理进程并侦听用户本地端口。
     - `<主节点公网IP地址>`：获取方式请参见获取主节点的节点名称和公网IP地址。
     - `<密钥文件路径>`：密钥文件保存的路径。
2. 配置浏览器。

   > **注意** 完成隧道创建之后，请保持终端打开状态，此时并不会返回响应。

   完成动态转发配置以后，您可以从以下两种方式中选择一种来进行浏览器配置。
   - Chrome浏览器命令行方式
     a. 打开命令行窗口，进入本地Google Chrome浏览器客户端的安装目录。

        操作系统不同，Chrome浏览器的默认安装目录不同。

操作系统	Chrome默认安装路径
Mac OS X	*/Applications/Google Chrome.app/Contents/macOS/Google Chrome*
Linux	*/usr/bin/google-chrome*
Windows	*C:\Program Files (x86)\Google\Chrome\Application*

     b. 在Chrome浏览器的默认安装目录下，执行以下命令。

        ```
        chrome --proxy-server="socks5://localhost:8157" --host-resolver-rules="MAP * 0.0.0.0 , EXCLUDE localhost" --user-data-dir=/tmp/
        ```

        相关参数描述如下：
        - `/tmp/`：如果是Windows操作系统，则`/tmp/`可以写成类似*/c:/tmppath/*的路径。如果是Linux或者Mac OS X操作系统，则可直接写成*/tmp/*路径。
        - `8157`：本地服务器端口以`8157`为例，实际配置时，您可使用本地服务器未被使用的任意一个端口。

     c. 在浏览器地址栏输入*http://<主节点的主机名>:<port>*，即可访问相应的Web UI。

        组件端口信息请参见服务常用端口及配置，主机名的获取请参见获取主节点的节点名称和公网IP地址。

        例如：访问Yarn页面时，在浏览器地址栏输入*http://emr-header-1:8088*。
   - 代理扩展程序方式

     代理扩展程序可以帮助您更加轻松地在浏览器中管理和使用代理，确保网页浏览和集群Web UI访问互不干扰。

     a. 安装Chrome的SwitchyOmega插件。

     b. 安装完成以后，单击SwitchyOmega插件，然后在弹出框中选择**选项**进行配置。

     c. 单击**新建情景模式**，输入**情景模式名称**（例如SSH tunnel），情景模式类型选择**PAC情景模式**。

     d. 在**PAC脚本**中配置以下内容。

        ```
        function regExpMatch(url, pattern) {
          try { return new RegExp(pattern).test(url); } catch(ex) { return false; }
        }
        function FindProxyForURL(url, host) {
            // Important: replace 172.31 below with the proper prefix for your VPC subnet
            if (shExpMatch(url, "*localhost*")) return "SOCKS5 localhost:8157";
            if (shExpMatch(url, "*emr-header*")) return "SOCKS5 localhost:8157";
            if (shExpMatch(url, "*emr-worker*")) return "SOCKS5 localhost:8157";
            return 'DIRECT';
        }
        ```

     e. 完成上述参数配置后，在左侧导航栏中单击**应用选项**。

f. 打开Chrome浏览器，在Chrome中单击SwitchyOmega插件，切换到之前创建的SSH tunnel情景模式下。

g. 在浏览器地址栏输入*http://<主节点的主机名>:<port>*，即可访问相应的Web UI。

组件端口信息请参见服务常用端口及配置，主机名的获取方式请参见获取主节点的节点名称和公网IP地址。

例如：访问Yarn页面时，在浏览器地址栏输入*http://emr-header-1:8088*。

使用本地端口转发方式

> 注意　此方式只能查看最外层的页面，无法查看详细的作业信息。

您可以通过SSH本地端口转发（即将主实例端口转发到本地端口），访问当前主节点上运行的网络应用界面，而不使用SOCKS代理。

1. 在本地服务器终端输入以下命令，创建SSH隧道。
   - 密钥方式

     ```
     ssh -i <密钥文件路径> -N -L 8157:<主节点主机名>:8088 root@<主节点公网IP地址>
     ```

   - 密码方式

     ```
     ssh -N -L 8157:<主节点主机名>:8088 root@<主节点公网IP地址>
     ```

     相关参数描述如下：
     - `-L`：使用本地端口转发，您可以指定一个本地端口，用于将数据转发到主节点本地Web服务器上标识的远程端口。
     - `8088`：主节点上ResourceManager的访问端口，您可以替换为其他组件的端口。

       组件端口信息请参见服务常用端口及配置，主机名的获取请参见获取主节点的节点名称和公网IP地址。
     - `8157`：本地服务器端口以`8157`为例，实际配置时，您可使用本地服务器未被使用的任意一个端口。
     - `<主节点公网IP地址>`：获取方式请参见获取主节点的节点名称和公网IP地址。
     - `<密钥文件路径>`：密钥文件保存的路径。
2. 保持终端打开状态，打开浏览器，在浏览器地址栏中输入*http://localhost:8157/*。

服务常用端口及配置

服务	端口	描述
Hadoop 2.X	50070	HDFS Web UI的端口。 配置参数为dfs.namenode.http-address或dfs.http.address。 说明　dfs.http.address已过期但仍能使用。
	50075	DataNode Web UI的端口。
	50010	Datanode服务端口，用于数据传输。
	50020	IPC服务的端口。
	8020	高可用的HDFS RPC端口。
	8025	ResourceManager端口。 配置参数为yarn.resourcemanager.resource-tracker.address。
	9000	非高可用的HDFS RPC端口。 配置参数为fs.defaultFS或fs.default.name。 说明　fs.default.name已经过期但仍能使用。
	8088	YARN Web UI的端口。
	8485	JournalNode的RPC端口。
	8019	ZKFC的端口。

服务	端口	描述
	19888	JobHistory Server的Web UI端口。 配置参数为mapreduce.jobhistory.webapp.address。
	10020	JobHistory Server的Web UI端口。 配置参数为mapreduce.jobhistory.address。
Hadoop 3.X	8020	NameNode的端口。 配置参数为dfs.namenode.http-address或dfs.http.address。 说明 dfs.http.address已过期但仍能使用。
	9870	
	9871	NameNode的端口。
	9866	DataNode的端口。
	9864	DataNode的端口。
	9865	DataNode的端口。
	8088	ResourceManager的端口。 配置参数为yarn.resourcemanager.webapp.address。
MapReduce	8021	JobTracker的端口。 配置参数为mapreduce.jobtracker.address。
Zookeeper	2181	客户端连接Zookeeper的端口。
	2888	Zookeeper集群内通讯使用，Leader监听此端口。
	3888	Zookeeper端口，用于选举Leader。
HBase	16010	Hbase的Master节点的Web UI端口。 配置参数为hbase.master.info.port。
	16000	HMaster的端口。 配置参数为hbase.master.port。
	16030	Hbase的RegionServer的Web UI管理端口。 配置参数为hbase.regionserver.info.port。
	16020	HRegionServer的端口。 配置参数为hbase.regionserver.port。
	9099	ThriftServer的端口。
Hive	9083	MetaStore服务默认监听端口。
	10000	Hive的JDBC端口。
	10001	Spark Thrift Sever的JDBC端口。
Spark	7077	• Spark的Master与Worker节点进行通讯的端口。 • Standalone集群提交Application的端口。
	8080	Master节点的Web UI端口，用于资源调度。
	8081	Worker节点的Web UI端口，用于资源调度。
	4040	Driver的Web UI端口，用于任务调度。

服务	端口	描述
	18080	Spark History Server的Web UI 端口。
Kafka	9092	Kafka集群节点之间通信的RPC端口。
Redis	6379	Redis服务端口。
HUE	8888	Hue Web UI的端口。
Oozie	11000	Oozie Web UI的端口。
Druid	18888	Druid Web UI的端口。
	18090	Overlord的端口。 配置参数为**overlord.runtime**页签下的**druid.plaintextPort**。
	18091	MiddleManager的端口。 配置参数为**middleManager.runtime**页签下的**druid.plaintextPort**。
	18081	Coordinator的端口。 配置参数为**coordinator.runtime**页签下的**druid.plaintextPort**。
	18083	Historical的端口。 配置参数为**historical.runtime**页签下的**druid.plaintextPort**。
	18082	Broker的端口。 配置参数为**broker.runtime**页签下的**druid.plaintextPort**。
Ganglia	9292	Ganglia Web UI的端口。
Ranger	6080	Ranger Web UI的端口。
Kafka Manager	8085	Kafka Manager的端口。
Superset	18088	Superset Web UI的端口。
Impala	21050	使用JDBC连接Impala的端口。
Presto	9090	Presto Web UI的端口。

3.1.4.5.2. 访问链接与端口

本文介绍如何设置安全组来访问集群上的开源组件。集群创建完成以后，E-MapReduce会为您的集群默认绑定几个域名，方便您访问开源组件的Web UI。

前提条件

已创建E-MapReduce集群，详情请参见创建集群。

> **说明** 确保创建的集群已开启**挂载公网**。

设置安全组访问

当您初次使用组件时，您需要按照以下步骤来打开您的安全组访问权限：

1. 获取机器的公网访问IP地址。

   为了安全的访问集群组件，在设置安全组策略时，推荐您只针对当前的公网访问IP地址开放。获取您当前公网访问IP地址的方法是，访问IP地址，即可查看您当前的公网访问IP地址。

2. 进入集群基础信息页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在EMR on ECS的集群管理页面，单击目标集群的**集群ID**。
3. 增加安全组策略。

i. 在**基础信息**页面，单击默认安全组的链接。

ii. 在**安全组规则**页面，根据需求开启以下端口。

> **注意** 为防止被外部的用户攻击导致安全问题，**授权对象**禁止填写为**0.0.0.0/0**。

访问不同组件Web UI需要打开的端口如下表。

服务名称	需要打开的端口
YARN UI	8443 **说明** 当集群开启Ranger后，可以访问RANGER UI。
HDFS UI	
Spark History Server UI	
Ganglia UI	
Oozie	
Tez	
ImpalaCatalogd	
ImpalaStatestored	
Storm	
RANGER UI	
Zeppelin	
Hue	8888

以添加**8443**端口为例，执行以下操作：

a. 在**安全组规则**页面，单击**手动添加**。

b. **端口范围**填写为**8443/8443**，**授权对象**填写为步骤1中获取的公网访问IP地址。

c. 单击**保存**。

> **说明**
> - 如果集群的网络类型是**VPC**，则配置内网入方向。如果集群的网络类型是**经典网络**，则配置公网入方向。此处以**VPC**网络为例。
> - 开放应用出入规则时应遵循最小授权原则。根据应用需求，只开放对应的端口。

iii. 完成以上策略配置后，在**入方向**页面，可查看新增策略。

上述配置成功后，您即已安全的开放了网络访问路径，网络配置完成。

访问开源组件的Web UI

1. 进入访问链接与端口页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群的**集群ID**。
   iv. 单击上方的**访问链接与端口**页签。
2. 在**访问链接与端口**页面，单击服务所在行的链接，即可正常的访问Web UI页面。

- 在2.7.x以上版本，或者是3.5.x以上版本，您可以使用Knox账号访问相应的UI页面，Knox账号创建请参见管理用户，Knox使用请参见Knox使用说明；访问Hue时请输入Hue的账户和密码，Hue使用请参见使用说明；Zeppelin可直接访问，无需账户密码。
- 当集群开启Ranger后，您可以使用默认的用户名和密码访问RANGER UI。

  > **说明** Hadoop集群默认的用户名和密码均为admin。DataLake集群访问Ranger UI的具体操作请参见。

- 您可以根据集群的版本来访问Flink的Web UI：
  - EMR-3.29.0之前版本

    仅支持通过SSH隧道方式访问Web UI时，请参见通过SSH隧道方式访问开源组件Web UI。

    > **说明** 访问YARN UI页面上的Flink作业：您可以在EMR控制台的**访问链接与端口**页面，单击YARN UI所在行的链接，在Hadoop控制台，单击Flink作业的ID，可以查看Flink作业运行的详情。详情请参见快速入门中的YARN UI方式。

  - EMR-3.29.0及后续版本
    - Flink-Vvp：您可以通过EMR控制台的方式访问Web UI，详情请参见通过Web UI查看作业状态。
    - Flink（VVR）：您可以通过SSH隧道方式访问Web UI，详情请参见通过SSH隧道方式访问开源组件Web UI。

      > **说明** 访问YARN UI页面上的Flink作业：您可以在EMR控制台的**访问链接与端口**页面，单击YARN UI所在行的链接，在Hadoop控制台，单击Flink作业的ID，可以查看Flink作业运行的详情。详情请参见快速入门中的YARN UI方式。

访问DataLake集群的Ranger UI

1. 添加6080端口，具体操作请参见添加安全组规则。
2. 获取公网IP地址。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在EMR on ECS的集群管理页面，单击目标集群的操作列的**节点管理**。
   iv. 在节点管理页面，单击主实例组所在行的 + 图标。

   在**公网IP**列可以获取到公网IP地址。
3. 在浏览器中输入*<公网IP地址>:6080*。

   <公网IP地址>为您在前一步骤中获取到的。
4. 在Ranger UI登录界面，输入用户名和密码。

   > **注意** 默认用户名为admin，密码为admin1234。

5. 首次登录后，需要修改密码。
   i. 单击上方的**Settings**。
   ii. 修改**admin**的密码。
   iii. 单击右上角的**admin > Log Out**。

   修改后即可使用新的密码登录。

3.1.5. 集群运维

3.1.5.1. 常用文件路径

本文为您介绍E-MapReduce中常用文件的路径。您可以登录Master节点查看常用文件的安装路径。

新版数据湖集群

大数据组件目录

- HDFS：/opt/apps/HDFS/hdfs-current
- Hive：/opt/apps/HIVE/hive-current
- YARN：/opt/apps/YARN/yarn-current
- SPARK3：/opt/apps/SPARK3/spark3-current
- SPARK2：/opt/apps/SPARK2/spark2-current

您也可以通过登录Master节点，执行 `env |grep xxx` 命令查看软件的安装目录，其中 `xxx` 为服务名。

例如，执行命令 `env |grep hive` ，查看Hive的安装目录。

日志目录

- Hadoop：/var/log/taihao-apps/hadoop/
- Hive：/var/log/taihao-apps/hive/
- YARN：/var/log/taihao-apps/yarn/
- Ranger：/var/log/taihao-apps/ranger/

配置文件

配置文件目录在*/etc/taihao-apps/xxx*下，例如：

- HDFS：*/etc/taihao-apps/hdfs-conf/*
- Spark：*/etc/taihao-apps/spark-conf/*
- Hive：*/etc/taihao-apps/hive-conf/*
- Knox：*/etc/taihao-apps/knox-conf/*
- YARN：*/etc/taihao-apps/yarn-conf/*

数据湖集群

大数据组件目录

软件安装目录在*/usr/lib/xxx*下，例如：

- Hadoop：*/usr/lib/hadoop-current*
- Spark：*/usr/lib/spark-current*
- Hive：*/usr/lib/hive-current*
- Flink：*/usr/lib/flink-current*
- Flume：*/usr/lib/flume-current*

您也可以通过登录Master节点，执行**env |grep xxx**命令查看软件的安装目录。

例如，执行以下命令，查看Hadoop的安装目录。

```
env |grep hadoop
```

返回如下信息，其中*/usr/lib/hadoop-current*为Hadoop的安装目录。

```
HADOOP_LOG_DIR=/var/log/hadoop-hdfs
HADOOP_HOME=/usr/lib/hadoop-current
YARN_PID_DIR=/usr/lib/hadoop-current/pids
HADOOP_PID_DIR=/usr/lib/hadoop-current/pids
HADOOP_MAPRED_PID_DIR=/usr/lib/hadoop-current/pids
JAVA_LIBRARY_PATH=/usr/lib/hadoop-current/lib/native:
PATH=/usr/lib/sqoop-current/bin:/usr/lib/spark-current/bin:/usr/lib/hive-current/hcatalog/bin:/usr/lib/hive-current/bin:/us
r/lib/datafactory-current/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/usr/lib/b2monitor-current//bin:/usr/lib/b2
smartdata-current//bin:/usr/lib/b2jindosdk-current//bin:/usr/lib/flow-agent-current/bin:/usr/lib/hadoop-current/bin:/usr/li
b/hadoop-current/sbin:/usr/lib/hadoop-current/bin:/usr/lib/hadoop-current/sbin:/root/bin
HADOOP_CLASSPATH=/usr/lib/hadoop-current/lib/*:/usr/lib/tez-current/*:/usr/lib/tez-current/lib/*:/etc/ecm/tez-conf:/opt/app
s/extra-jars/*:/usr/lib/spark-current/yarn/spark-2.4.5-yarn-shuffle.jar
HADOOP_CONF_DIR=/etc/ecm/hadoop-conf
YARN_LOG_DIR=/var/log/hadoop-yarn
HADOOP_MAPRED_LOG_DIR=/var/log/hadoop-mapred
```

日志目录

组件日志目录在*/mnt/disk1/log/xxx*下，例如：

- Yarn ResourceManager日志：Master节点*/mnt/disk1/log/hadoop-yarn*
- Yarn NodeNanager日志：Slave节点*/mnt/disk1/log/hadoop-yarn*
- HDFS NameNode日志：Master节点*/mnt/disk1/log/hadoop-hdfs*
- HDFS DataNode日志：Slave节点*/mnt/disk1/log/hadoop-hdfs*
- Hive日志：Master节点*/mnt/disk1/log/hive*
- ESS日志：Master和Worker节点*/mnt/disk1/log/ess/*

配置文件

配置文件目录在*/etc/ecm/xxx*下，例如：

- Hadoop：*/etc/ecm/hadoop-conf/*
- Spark：*/etc/ecm/spark-conf/*

- Hive：*/etc/ecm/hive-conf/*
- Flink：*/etc/ecm/flink-conf/*
- Flume：*/etc/ecm/flume-conf/*

如果您需要修改配置文件中的参数，请登录E-MapReduce控制台操作，通过SSH方式只能浏览配置文件中的参数。

数据目录

JindoFS缓存数据：*/mnt/disk*/bigboot/*

3.1.5.2. 登录集群

本文为您介绍如何使用SSH方式（SSH密钥对或SSH密码方式）在Windows和Linux环境中登录集群。

背景信息

在本地计算机的终端与集群主节点创建SSH连接之后，您可以通过Linux命令监控集群并与集群交互，也可以在SSH连接中创建隧道以查看开源组件的Web页面，详情请参见通过SSH隧道方式访问开源组件Web UI。

前提条件

- 已在EMR on ECS创建集群，详情请参见创建集群。
- 确保本地服务器与集群主节点网络连通。您可以在创建集群时打开**挂载公网**开关，或者在集群创建好之后在ECS控制台上为主节点挂载公网，为主节点ECS实例分配固定公网IP或EIP，详情请参见绑定弹性网卡。

获取主节点公网IP地址和节点名称

1. 进入节点管理页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**节点管理**。
2. 在**节点管理**页面，单击主实例组所在行的图标。

   - **公网IP**：可以获取主节点的公网IP地址。
   - **节点名称**：可以获取主节点的节点名称。
     集群类型不同，对应的节点名称不同：
     - DataLake、DataFlow和OLAP集群：master-1-1。
     - 其余集群类型：emr-header-1。

SSH密钥方式

> **说明** 主节点公网IP地址请参见获取主节点公网IP地址和节点名称。

您可以通过以下三种方式登录集群，详细信息如下：

- 本地使用Linux操作系统
  下面步骤以私钥文件*ecs.pem*为例进行介绍：
  i. 执行以下命令，修改私钥文件的属性。

  ```
  chmod 400 ~/.ssh/ecs.pem
  ```

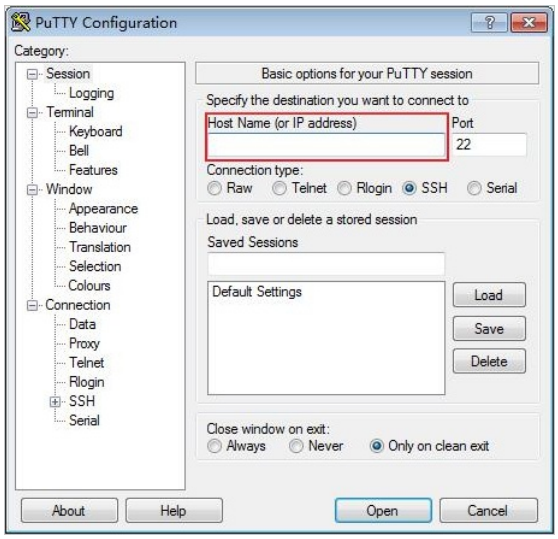
  `~/.ssh/ecs.pem` 为*ecs.pem*私钥文件在本地服务器上的存储路径。
  ii. 执行以下命令，连接主节点。

  ```
  ssh -i ~/.ssh/ecs.pem emr-user@<主节点公网IP地址>
  ```

- 本地使用Windows操作系统（通过PuTTY配置信息）

您可以按照以下方式登录主节点。

i. 下载PuTTY和PuTTYgen。
ii. 将*.pem*私钥文件转换为*.ppk*私钥文件。
   a. 运行PuTTYgen。本示例中PuTTYgen版本为0.73。
   b. 在**Actions**区域，单击**Load**，导入创建集群时保存的私钥文件。
      导入时注意确保导入的格式要求为**All files (*.*)**。
   c. 选择待转换的*.pem*私钥文件，单击**打开**。
   d. 单击**Save private key**。
   e. 在弹出的对话框中，单击**是**，指定*.ppk*私钥文件的名称，然后单击**保存**。
      保存转化后的私钥到本地。例如：*kp-123.ppk*。
iii. 运行PuTTY。
iv. 选择**Connection > SSH > Auth**，在最下面一个配置项**Private key file for authentication**中，单击**Browse**，选择转化后的密钥文件。
v. 单击**Session**，在**Host Name (or IP address)**下的输入框中，输入登录账号和主节点公网IP地址。
   格式为*emr-user@[主节点公网IP地址]*，例如*emr-user@10.10.**.***。



vi. 单击**Open**。

- 本地使用Windows操作系统（通过命令配置信息）

  打开CMD，输入以下命令登录集群。

```
ssh -i <.pem私钥文件在本地机上的存储路径> emr-user@<主节点公网IP地址>
```

SSH密码方式

> **说明** 以下步骤中涉及的用户名，密码分别是root用户和创建集群时设置的密码。主节点公网IP地址请参见获取主节点公网IP地址和节点名称。

针对不同操作系统，详细的操作步骤如下：

- 本地使用Linux操作系统

  您可以在本地终端的命令行中运行如下命令连接主节点。

```
ssh root@[主节点公网IP地址]
```

- 本地使用Windows操作系统
  i. 下载并安装PuTTY。
     下载链接：PuTTY。
  ii. 启动PuTTY。
  iii. 配置连接Linux实例所需的信息。
     - **Host Name (or IP address)**：输入实例的固定公网IP或EIP。

- **Port**：输入22。
- **Connection Type**：选择SSH。
- （可选）**Saved Sessions**：输入一个便于识别的名称，然后单击**Save**即可保存会话，下次登录时无需输入公网IP等信息。

iv. 单击**Open**。

v. 输入用户名（默认为root），然后按回车键。

输入完成后按回车键即可，登录Linux实例时界面不会显示密码的输入过程。

常见问题

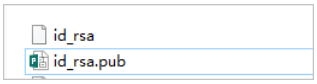
- Q：如何在本地以免密方式登录集群？

  A：您可以通过以下步骤在本地以免密方式登录集群。

  i. 打开CMD，输入以下命令生成公钥。

  ```
  ssh-keygen
  ```

  本地服务器目录下会生成相应的公钥文件。

  ii. 将生成的公钥加入至待访问集群的主节点上。

  a. 进入待访问集群的*/.ssh*目录。

  ```
  cd ~/.ssh
  ```

  b. 配置主节点的密钥。

  ```
  vim authorized_keys
  ```

  c. 添加本地公钥中*id_rsa*的内容至*authorized_keys*中。

  iii. 加入本地机器的IP地址至安全组。

  a. 获取机器的公网访问IP地址。

  为了安全的访问集群组件，在设置安全组策略时，推荐您只针对当前的公网访问IP地址开放。获取您当前公网访问IP地址的方法是，访问IP地址，即可查看您当前的公网访问IP地址。

  b. 添加安全组规则，以开启22端口。

  添加安全组规则，详情请参见添加安全组规则。

  iv. 在CMD中，输入以下命令免密登录集群。

  ```
  ssh root@<主节点公网IP地址>
  ```

- Q：如何登录Core节点？

  A：您可以通过以下步骤登录Core节点。

  - DataLake、DataFlow和OLAP集群

    a. 在Master节点上切换到emr-user账号。

    ```
    su emr-user
    ```

    b. 免密码登录到对应的Core节点。

    ```
    ssh core-1-1
    ```

  - 其余集群

    a. 在Master节点上切换到hadoop账号。

    ```
    su hadoop
    ```

    b. 免密码登录到对应的Core节点。

    ```
    ssh emr-worker-1
    ```

3.1.5.3. 扩容集群

当E-MapReduce集群计算资源或存储资源不足时，您可以对集群进行水平扩展。本文为您介绍如何扩容集群，以及如何修改新增节点的密码。

前提条件

已创建集群，详情请参见创建集群。

使用限制

仅支持扩展Core节点和Task节点，且新扩展节点的配置默认与已有节点一致。

操作步骤

1. 进入节点管理页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**节点管理**。
2. 在**节点管理**页面，单击目标机器组操作列的**扩容**。
3. 在**扩容**对话框中，根据实际情况修改相应参数。

配置项	说明
节点组名称	当前节点组的名称。
节点类型	当前节点组的类型。
当前配置	当前节点组中的实例信息。
付费类型	当前集群的付费类型。新增节点的付费类型继承集群的付费类型，不可更改。如果是包年包月类型，则您可设置新增节点的**付费时长**。
当前数量	当前节点组中实例的数量。
增加数量	单击调整框的上下箭头或直接在调整框中输入数字，设置需要增加当前节点组实例的数量。
服务协议	阅读并同意服务条款后，选中即可。

4. 完成上述参数配置后，单击**确定**。

修改新增节点密码

集群扩容成功后，您可以通过SSH方式登录新增节点来修改root密码。

1. 登录集群的Master节点，详情请参见登录集群。
2. 执行如下命令，切换到hadoop用户。

```
su hadoop
```

3. 执行如下命令，登录新增节点。

```
ssh <ip.of.worker>
```

新增节点的内网IP地址，您可以在**节点管理**页面获取。

4. 执行如下命令，根据提示信息修改root用户密码。

```
sudo passwd root
```

3.1.5.4. 缩容集群

当E-MapReduce上按量付费类型集群的计算资源或存储资源过剩时，您可以缩减Task节点的数量。本文为您介绍如何缩容集群。

前提条件

已在EMR on ECS创建集群，详情请参见创建集群。

使用限制

仅支持对EMR集群的Task节点缩容，集群还需满足以下条件：

- EMR集群版本2.x高于2.5.0，3.x高于3.2.0。

- 集群状态为运行中。
- 集群付费类型为按量付费。

操作步骤

1. 进入节点管理页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**节点管理**。
2. 在**节点管理**页面，单击目标机器组操作列的**缩容**。
3. 在**缩容**对话框中，单击调整框的下箭头或直接在调整框中输入数字，设置需要保留的Task节点的数量。
4. 完成上述参数配置后，单击**确定**。

3.1.5.5. 扩容磁盘

当E-MapReduce集群的数据存储空间不足时，您可以根据本文进行磁盘（数据盘）扩容。本文为您介绍如何对数据盘进行扩容。

前提条件

已在E-MapReduce控制台创建集群，详情请参见创建集群。

使用限制

E-MapReduce控制台仅支持数据盘扩容操作，不支持系统盘扩容。系统盘扩容请在ECS控制台操作，详情请参见在线扩容云盘（Linux系统）或离线扩容云盘（Linux系统）。

操作步骤

1. 进入节点管理页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**节点管理**。
2. 在**节点管理**页面，单击待操作机器组所在行的**磁盘扩容**。
3. 在**磁盘扩容**对话框中，修改数据盘容量。
4. 单击**确定**。
   扩容完成后，您可以在机器组的磁盘信息列查看扩容后的数据盘信息。

> **说明** 如果扩容没有生效，请重启对应的ECS实例。

3.1.5.6. 配置弹性伸缩

3.1.5.6.1. 弹性伸缩概述

本文介绍E-MapReduce的弹性伸缩功能，您可以根据业务需求和策略设置伸缩策略。该功能目前仅支持E-MapReduce的Hadoop集群，弹性伸缩开启并配置完成后，当业务需求增长时E-MapReduce会自动为您增加Task节点以保证计算能力，当业务需求下降时E-MapReduce会自动减少Task节点以节约成本。

应用场景

在以下场景中，开启E-MapReduce的弹性伸缩功能，可帮助您节省成本，提高执行效率。

- 临时需要按照时间段添加Task节点，补充计算能力。
- 为确保重要作业按时完成，需要按照某些集群指标扩充Task节点。

功能介绍

E-MapReduce支持按时间伸缩规则配置和按负载伸缩规则配置两种伸缩策略，但使用时两者只能二选一。如果切换伸缩策略，原伸缩规则会保留，但处于失效状态，不会被触发执行；当前已扩容的节点也会保留，除非缩容规则触发，否则不会被缩容。

E-MapReduce弹性伸缩支持抢占式和按量付费两种实例，您可根据实际需要选择，详细信息请参见配置弹性伸缩。

3.1.5.6.2. 配置弹性伸缩

当您的业务量需求不断波动时，建议您开启弹性伸缩功能并配置相应的伸缩规则，以便于E-MapReduce（简称EMR）可以按业务量波动增加或减少Task节点。确保作业完成的同时，可以节省成本。本文为您介绍如何在EMR控制台配置弹性伸缩。

前提条件

已新建集群，创建详情请参见创建集群。

注意事项

弹性伸缩配置可以指定伸缩的节点的硬件规格。您只能在开启弹性伸缩功能时配置，保存后不能更改。如果特殊情况确实需要修改，可以关闭弹性伸缩功能后，再次开启。

- 系统会根据您选择的vCPU和内存规格，自动匹配出满足条件的实例，并显示在备选实例列表中。您需要选中备选的实例，以便集群按照已选的实例规格进行伸缩。
- 为避免由于ECS库存不足造成的弹性伸缩失败，您最多可以选择3种ECS实例。
- 无论是选择高效云盘还是SSD云盘，数据盘最小设置为40 GB。

操作步骤

1. 进入弹性伸缩页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群的集群ID。
   iv. 单击上方的**弹性伸缩**页签。
2. 新建弹性伸缩组。
   i. 在**伸缩配置**页签，单击**新建弹性伸缩机器组**。
   ii. 在**新增弹性伸缩机器组**对话框中，输入**机器组名称**，单击**确定**。
3. 在**伸缩配置**页签，单击目标机器组操作列的**配置规则**。
4. 在**弹性伸缩配置**面板的**基础信息**区域，配置相关参数。

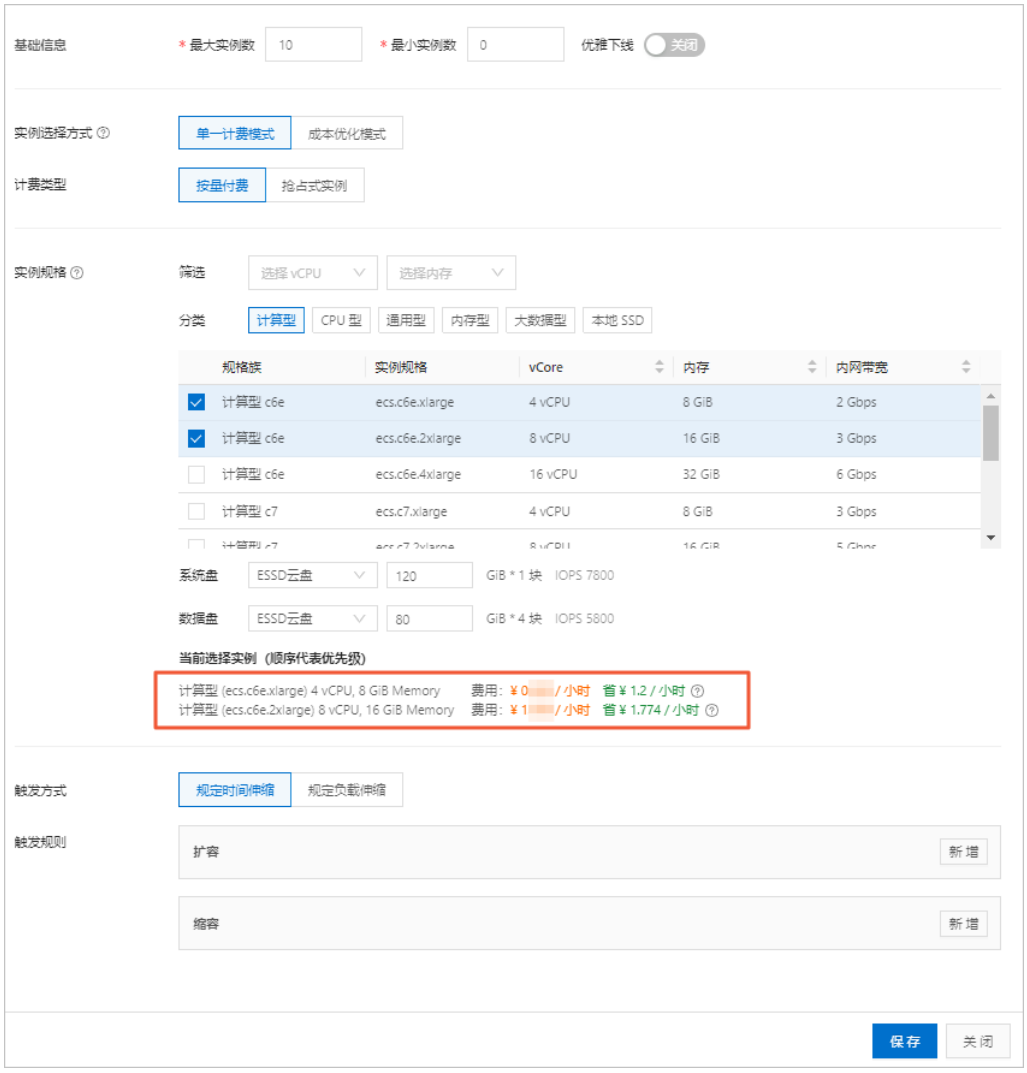
参数	描述
最大实例数	弹性伸缩组的Task节点上限。一旦达到上限，即使满足弹性伸缩的规则，也不会继续执行弹性伸缩。最大实例数的上限为1000。
最小实例数	弹性伸缩的Task节点下限。如果弹性伸缩规则中设置的增加或减少Task节点数小于此处设置的最小实例数，则在首次执行时，集群会以最小节点数为准进行伸缩。 例如，您设置弹性扩容规则为每天零点动态添加1个节点，最小节点数为3，则系统在第一天的零点时会添加3个节点，以满足最小节点数的要求。
优雅下线	您可以设置超时时间，下线YARN上作业所在的Task节点。如果Task节点没有运行YARN上的作业或者作业运行超出您设置的超时时间，则下线Task节点。超时时间最大值为3600秒。 **说明** 开启优雅下线时，请先将YARN配置项yarn.resourcemanager.nodes.exclude-path的值修改为/etc/ecm/hadoop-conf/yarn-exclude.xml。

5. 在**弹性伸缩配置**面板的实例区域，选择实例方式、计费类型和实例规格。
   - **单一计费模式**

     系统会根据您选择的vCPU和内存规格，自动匹配出满足条件的实例，并显示在实例规格区域。您需要选中实例规格区域的实例，以便集群按照已选的实例规格进行伸缩。单一计费模式支持以下两种计费类型：

- **按量付费**

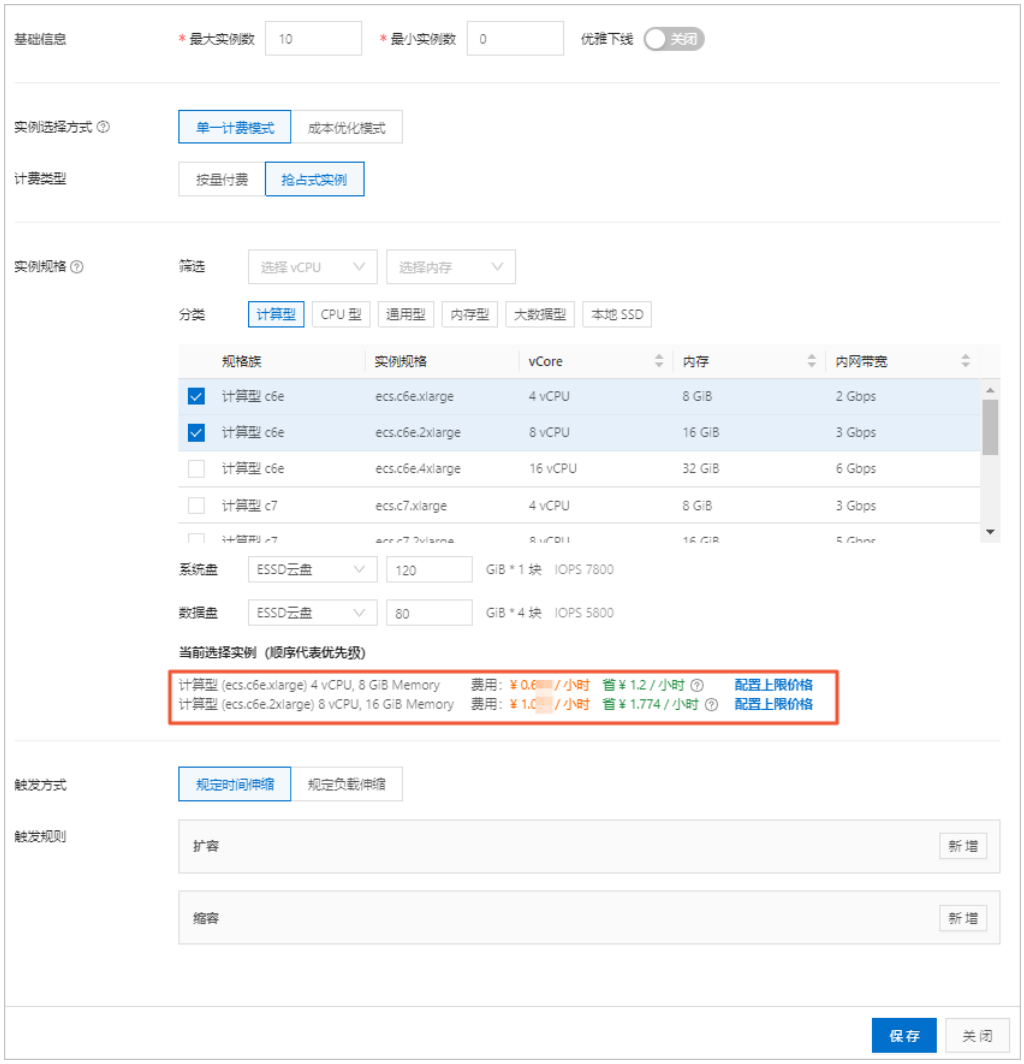
  勾选实例的顺序决定显示节点的优先级，勾选实例后下方区域会显示每一个实例每小时的价格（包含EMR实例价格和ECS实例价格）。



- **抢占式实例**

  > 注意　当您的作业对SLA（Service Level Agreement）要求较高时，请慎用该实例。因为竞价失败等原因会导致抢占式实例被释放，所以请谨慎使用抢占式实例。

  勾选实例的顺序决定显示节点的优先级，每一个实例后会显示该实例在按量付费时每小时的价格。您还可以设置每台实例的上限价格，当满足规则时显示该类实例。抢占式实例详情，请参见抢占式实例概述。



  - **成本优化模式**

    该模式下，您可以制定更详细的成本控制策略，在成本和稳定性之间进行调整和权衡。

参数	描述
组内最小按量节点数量	弹性伸缩组需要的按量实例的最小个数，当伸缩组中按量实例个数小于该值时，将优先创建按量实例。
按量节点所占比例	弹性伸缩组内最小按量节点数量满足之后，创建实例中按量实例所占的比例。
最低价的多个实例规格	指定最低价的多个实例规格种类数。当创建抢占实例时，将在这些规格种类中进行均衡分布。最大值为3。

参数	描述
抢占实例补偿	是否开启竞价实例的补偿机制。开启抢占实例补偿后，在竞价实例被回收前5分钟左右，将主动替换掉当前竞价实例。

当您不指定**组内最小按量节点数量**、**按量节点所占比例**和**最低价的多个实例规格**参数时，您创建的是普通成本优化伸缩组。否则，您创建的是成本优化混合实例伸缩组。成本优化混合实例伸缩组与普通成本优化伸缩组在接口和功能方面是完全兼容的。

对于成本优化混合实例伸缩组，您可以通过合理制定混合实例策略，以实现与普通成本优化伸缩组完全相同的行为。例如：

- 普通成本优化伸缩组创建的全为按量实例

  指定**组内最小按量节点数量**=0，**按量节点所占比例**=100，**最低价的多个实例规格**=1。
- 普通成本优化伸缩组优先创建竞价实例

  指定**组内最小按量节点数量**=0，**按量节点所占比例**=0，**最低价的多个实例规格**=1。

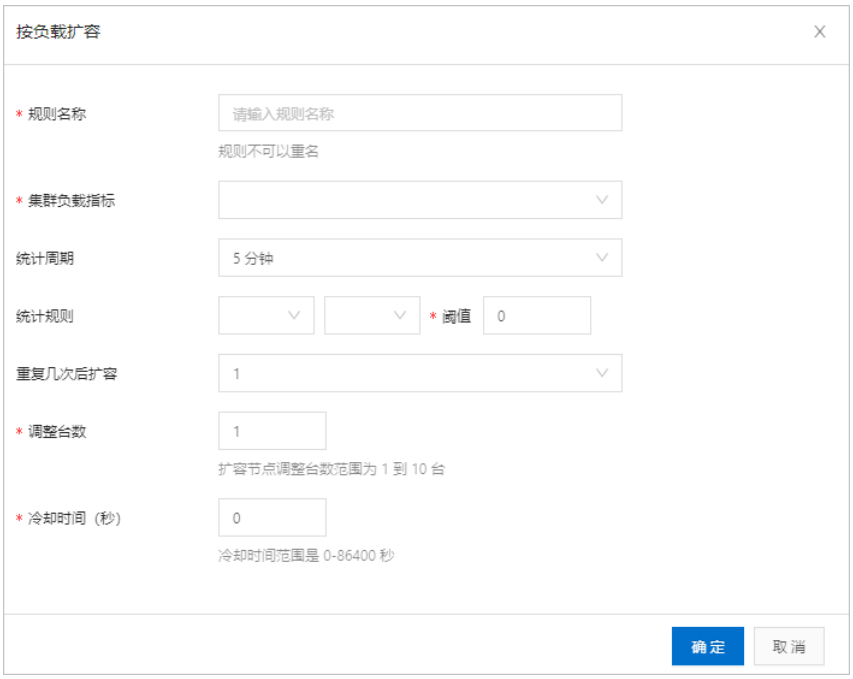
> **说明** 成本优化模式详情，请参见AutoScaling成本优化模式升级--混合实例策略。

6. 在**弹性伸缩配置**面板的**触发方式**区域，选择触发方式和规则。
   - **规定时间伸缩**：如果Hadoop集群计算量在一定的周期内存在明显的波峰和波谷，则您可以设置在每天、每周或每月的固定时间段扩展一定量的Task节点来补充计算能力，这样在保证作业完成的同时，也可以节省成本。

     伸缩规则分为扩容规则和缩容规则，本示例以扩容规则为例介绍。集群关闭弹性伸缩功能后，所有规则会被清空，再次开启弹性伸缩功能时，需要重新配置伸缩规则。

参数	描述
规则名称	在同一个集群中，伸缩规则名称（包括扩容规则和缩容规则）不允许重复。
时间执行规则	■ **重复执行**：您可以选择每天、每周或每月的某一特定时间点执行一次弹性伸缩动作。 ■ **只执行一次**：集群在指定的时间点执行一次弹性伸缩动作。
执行时间	选择规则执行的时间。
规则有效期	选择规则的有效期。
重试过期时间(秒)	弹性伸缩在到达指定时间时可能由于各种原因不能执行，通过设置重试过期时间，系统会在该时间范围内每隔30秒尝试执行一次，直到在满足条件时执行伸缩。设置范围为0~21600秒。 假设在指定时间段需要进行弹性伸缩动作A，如果有其他弹性伸缩动作B正在执行或正处在冷却期，则动作A无法执行。在您设置的重试过期时间内，每隔30秒会重试一次，尝试执行A，一旦条件满足，集群会立刻执行弹性伸缩。
调整台数	规则被触发时，集群每次增加Task节点数量。
冷却时间(秒)	每次弹性伸缩动作执行完成，到可以再次进行弹性伸缩的时间间隔。在冷却时间内，不会发生弹性伸缩动作。
   - **规定负载伸缩**：如果您无法准确的预估大数据计算的波峰和波谷，则可以使用按负载伸缩配置的策略。

伸缩规则分为扩容规则和缩容规则，本示例以扩容规则为例介绍。集群关闭弹性伸缩功能后，所有规则会被清空，再次开启弹性伸缩功能时，需要重新配置伸缩规则。切换伸缩策略时（例如，从按负载伸缩切换到按时间伸缩），原策略下的伸缩规则处于失效状态，不会被触发，但已经扩容的节点会继续保留，不会被释放。



参数	描述
规则名称	在同一个集群中，伸缩规则名称（包括扩容规则和缩容规则）不允许重复。
集群负载指标	在YARN的负载指标中获取，详细信息请参见Hadoop官方文档。 E-MapReduce弹性伸缩指标与YARN负载指标的对应关系，请参见E-MapReduce弹性伸缩指标与YARN所属服务的对应关系。
统计周期	您选定的集群负载指标在一个统计周期内，按照选定的聚合维度（平均值、最大值和最小值），达到触发阈值为一次触发。
统计规则	
重复几次后扩容	负载指标聚合后达到阈值触发的次数，达到该次数后触发集群弹性伸缩的动作。
扩容(台)	规则被触发时，集群每次执行增加的Task节点数量。
冷却时间(秒)	每次弹性伸缩动作执行完成，到可以再次进行弹性伸缩的时间间隔。在冷却时间内，即使满足弹性伸缩条件也不会发生弹性伸缩动作。即忽略本次在冷却时间内触发的弹性伸缩动作，直到下一次满足伸缩条件且不在冷却时间内再执行。

7. 单击**保存**。

   您可以根据实际情况，开启弹性伸缩。开启弹性伸缩详情，请参见开启或关闭弹性伸缩。

E-MapReduce弹性伸缩指标与YARN所属服务的对应关系

E-MapReduce弹性伸缩指标	所属服务	说明
YARN.AvailableVCores	YARN	可供分配的虚拟核数。
YARN.PendingVCores	YARN	待分配的虚拟核数。
YARN.AllocatedVCores	YARN	已分配的虚拟核数。
YARN.ReservedVCores	YARN	预留的虚拟核数。
YARN.AvailableMemory	YARN	可供分配的内存量。

E-MapReduce弹性伸缩指标	所属服务	说明
YARN.PendingMemory	YARN	待分配的内存量。
YARN.AllocatedMemory	YARN	已分配的内存量。
YARN.ReservedMemory	YARN	预留的内存量。
YARN.AppsRunning	YARN	运行中的任务数。
YARN.AppsPending	YARN	挂起的任务数。
YARN.AppsKilled	YARN	终止的任务数。
YARN.AppsFailed	YARN	失败的任务数。
YARN.AppsCompleted	YARN	完成的任务数。
YARN.AppsSubmitted	YARN	提交的任务数。
YARN.AllocatedContainers	YARN	已分配的容器数。
YARN.PendingContainers	YARN	待分配的容器数。
YARN.ReservedContainers	YARN	预留的容器数。
YARN.MemoryAvailablePrecentage	YARN	剩余内存的百分比 `(MemoryAvailablePrecentage= AvailableMemory/Total Memory)` 。
YARN.ContainerPendingRatio	YARN	待分配的容器数与已分配的容器数的比率 `(ContainerPendingRatio = PendingContainers/AllocatedContainers)` 。

3.1.5.6.3. 开启或关闭弹性伸缩

当您的业务量需求不断波动时，建议您配置相应的伸缩规则并开启弹性伸缩功能，以使E-MapReduce可以按业务量波动来增加或减少Task节点。当您需要更改实例配置或者当您的业务量需求趋于稳定时，您可以关闭弹性伸缩功能。本文为您介绍如何开启或关闭弹性伸缩。

前提条件

已完成弹性伸缩的配置，详情请参见配置弹性伸缩。

注意事项

当伸缩组内节点数为0时，您才可以关闭弹性伸缩。当伸缩组内节点不为0时，您需要先为伸缩组设置缩容规则或者修改最大实例数为0，直至伸缩组内节点全部缩容，才可以关闭弹性伸缩。

操作步骤

1. 进入弹性伸缩页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群的集群ID。
   iv. 单击上方的**弹性伸缩**页签。
2. 在**伸缩配置**页面，可以开启或者关闭弹性伸缩。
   - 开启弹性伸缩：打开目标机器组所在行的**弹性伸缩状态**开关。

     当弹性伸缩开启之后，如果您修改基础信息或触发规则，在保存配置后，需要在**弹性伸缩配置**页面，单击**应用最新配置**，使修改后的规则生效。
   - 关闭弹性伸缩：关闭目标机器组所在行的**弹性伸缩状态**开关。

3.1.5.6.4. 查看弹性伸缩记录

本文为您介绍在弹性伸缩执行完成后，如何查看弹性伸缩活动的执行记录。

前提条件

集群已进行弹性伸缩的相关配置。

操作步骤

1. 进入弹性伸缩页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群的集群ID。
   iv. 单击上方的**弹性伸缩**页签。
2. 在弹性伸缩页面，单击**伸缩记录**页签。
   您可以查看弹性伸缩活动执行完成后的节点数量和状态等信息。弹性伸缩的状态及描述见下表。

状态	描述
执行成功	根据伸缩规则，所有弹性伸缩中的所有节点被加入或移出集群。
部分执行成功	根据伸缩规则，有部分节点成功被加入或移出集群，但是受磁盘配额管理或ECS库存的影响，部分节点执行失败。
执行失败	根据伸缩规则，没有一个节点被加入或移出集群。
正在执行	弹性伸缩活动正在执行。
请求被拒绝	当运行伸缩规则后的实例数大于最大实例数或者小于最小实例数时，或者当运行规则触发时上一次伸缩活动还未结束，就会拒绝该规则运行。

3.1.5.7. 管理节点组

本文为您介绍如何新增和删除节点组。

背景信息

您可以新增节点组，以满足不同实例节点的需求。例如，内存型实例节点（vCore：vMem = 1 vCPU：8 GiB）用于大数据离线处理，计算型实例（vCore：vMem =1 vCPU：2 GiB）用于模型训练。

前提条件

已在EMR on ECS控制台创建集群，详情请参见创建集群。

使用限制

- 支持新增Core和Task节点组。
- 最多支持新增10个节点组。
- 仅支持删除Task节点组，不支持删除Core节点组。

注意事项

当待操作节点组的节点数大于0时，需要先释放节点组中的ECS实例才能删除节点组。

新增节点组

1. 进入节点管理页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**节点管理**。
2. 在**节点管理**页面，单击**新增节点组**。
3. 在**新增节点组**面板中，配置以下信息。

参数	描述
节点组类型	支持**Core（核心实例组）**和**TASK（任务实例组）**类型。
付费类型	当前节点组的付费类型。
节点组名称	节点组名称不允许重复。
实例类型	根据您实际情况选择实例。
存储配置	○ **系统盘**：根据需要选择ESSD云盘或者高效云盘。系统盘取值范围为60 ~ 500 GiB。推荐至少120 GiB。 ○ **数据盘**：根据需要选择ESSD云盘或者高效云盘。系统盘取值范围为40 ~ 32768 GiB。推荐至少80 GiB。

4. 单击**确定**。

   添加完成后，即可在**节点管理**页面查看到新增的节点组。

删除节点组

1. 进入节点管理页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**节点管理**。
2. 在**节点管理**页面，单击目标集群所在行的**删除节点组**。

> **说明** 删除节点组前需要先释放节点组下的所有节点。

3. 在弹出的**提示**对话框中，单击**删除**。

相关文档

- 如果需要给节点组扩容，详情请参见扩容集群。
- 如果需要给节点组缩容，详情请参见缩容集群。
- 如果需要给磁盘扩容，详情请参见扩容磁盘。

3.1.5.8. 释放集群

当E-MapReduce按量付费的集群不再使用时，您可以随时释放集群，以节约成本。

背景信息

确认释放集群后，系统会对集群进行如下处理：

1. 强制终止集群上的所有作业。
2. 终止并释放所有的ECS实例。

   这个过程所需时间取决于集群的大小，集群越小释放越快，通常几秒内均可完成，至多不会超过5分钟。

前提条件

请确保待释放集群的状态是创建中、运行中或空闲中。

使用限制

仅支持按量付费集群释放。包年包月集群请提交工单处理。

操作步骤

1. 进入节点管理页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**节点管理**。
2. 在**集群管理**页面，选择目标集群操作列的 **> 释放**。

您也可以在集群**基础信息**页面，在上方选择**集群状态管理 > 释放**。

3. 在弹出的**释放集群**对话框中，单击**确定**。

3.2. 集群告警

3.2.1. 创建阈值报警规则

云监控（CloudMonitor）是阿里云的一种监控告警服务，当您需要监控E-MapReduce资源的使用和运行情况时，可以通过创建阈值报警规则，实现监控项超过设定阈值后自动发送报警通知的功能，帮助您及时了解监控数据异常并快速进行处理。

前提条件

已创建集群，详情请参见创建集群。

操作步骤

1. 登录云监控控制台。
2. 在左侧导航栏，选择**报警服务 > 报警规则**。
3. 在**阈值报警**页面，单击**创建报警规则**。
4. 在**创建报警规则**页面，设置报警规则相关信息，具体操作请参见创建报警规则。

参数	说明
产品	从**产品**列表中选择**E-MapReduce**。
资源范围	报警规则的作用范围。取值： ○ **全部资源**：表示该规则作用在用户名下E-MapReduce的全部集群上。 ○ **应用分组**：表示该规则作用在用户名下E-MapReduce的指定应用分组内的全部集群上。 ○ **实例**：表示该规则只作用在指定的集群上。
规则描述	报警规则的主体，定义在监控数据满足指定条件时，触发报警规则。例如：CPU使用率5分钟平均值>=90%，持续3个周期，则报警服务5分钟检查一次数据是否满足平均值>=90%，只检测3次。 **说明** 规则详细信息，请参见报警规则。 规则描述的设置方法如下： i. 单击**添加规则**。 ii. 在**添加规则描述**面板，设置规则名称、监控指标类型、监控指标、阈值、报警级别和报警方式等。 iii. 单击**确定**。
通道沉默周期	报警发生后未恢复正常，间隔多久重复发送一次报警通知。 **说明** 单击**高级设置**，可设置该参数。
生效时间	报警规则的生效时间，报警规则只在生效时间内才会检查监控数据是否需要报警。 **说明** 单击**高级设置**，可设置该参数。
报警联系人组	发送报警的联系人组。 如果您需要新建联系人组，创建详情请参见创建报警联系人或报警联系组。
报警回调	填写公网可访问的URL，云监控会将报警信息通过POST请求推送到该地址，目前仅支持HTTP协议。
弹性伸缩	如果您选中**弹性伸缩**，当报警发生时，会触发相应的伸缩规则。您需要设置弹性伸缩的**地域**、**弹性伸缩组**和**弹性伸缩规则**。 ○ 创建弹性伸缩组的操作方法，请参见创建伸缩组。 ○ 创建弹性伸缩规则的操作方法，请参见配置伸缩规则。
日志服务	如果您选中**日志服务**，当报警发生时，会将报警信息写入日志服务。您需要设置日志服务的**地域**、**Project**和**Logstore**。 创建Project和Logstore的操作方法，请参见快速入门。

参数	说明
消息服务MNS-Topic	如果您打开消息服务MNS-Topic开关，当报警发生时，会将报警信息写入消息服务的主题。您需要设置消息服务的地域和主题。 关于如何创建主题，请参见创建主题。

5. 单击**确定**。

报警规则

服务名	指标名	指标含义
HDFS	NameNodeIpcPortOpen	NameNode的IPC端口的可用性： • 1：可用 • 0：不可用
	TotalDFSUsedPercent	集群的HDFS总容量使用百分比。
	DataNodeDfsUsedPercent	DataNode节点的DFS使用率。
	DataNodeIpcPortOpen	DataNode中IPC端口的可用性： • 1：可用 • 0：不可用
	JournalNodeRpcPortOpen	JournalNode的RPC端口的可用性： • 1：可用 • 0：不可用
	ZKFCPortOpen	ZKFC端口的可用性： • 1：可用 • 0：不可用
	dfs.FSNamesystem.MissingBlocks	丢失的块数。
	dfs.datanode.VolumeFailures	HDFS检测出的坏盘数。
YARN	ResourceManagerPortOpen	ResourceManager的服务端口的可用性： • 1：可用 • 0：不可用
	JobHistoryPortOpen	JobHistory的服务端口的可用性： • 1：可用 • 0：不可用
	yarn.ClusterMetrics.NumUnhealthyNM	Unhealthy的NodeManager个数。
	ProxyServerPortOpen	WebAppProxy端口的可用性： • 1：可用 • 0：不可用
	TimelineServerPortOpen	TimelineServer的服务端口的可用性： • 1：可用 • 0：不可用
Hive	MetastorePortOpen	HiveMetaStore端口的可用性： • 1：可用 • 0：不可用
	HiveServer2PortOpen	HiveServer2的服务端口的可用性： • 1：可用 • 0：不可用

服务名	指标名	指标含义
	ThriftServerPortOpen	ThriftServer的服务端口的可用性： • 1：可用 • 0：不可用
HBase	HMasterIpcPortOpen	HMaster的IPC端口可用性： • 1：可用 • 0：不可用
	HRegionServerIpcPortOpen	HRegionServer的IPC的端口可用性： • 1：可用 • 0：不可用
ZooKeeper	ZKClientPortOpen	ZooKeeper客户端监听端口的可用性： • 1：可用 • 0：不可用
Hue	HuePortOpen	Hue端口的可用性： • 1：可用 • 0：不可用
Storm	StormNimbusThriftPortOpen	StormNimbus的Thrift端口的可用性： • 1：可用 • 0：不可用
HOST	proc_total	总进程数目。
	part_max_used	磁盘分区使用的最大百分比。
	disk_free_percent_mnt_disk1	剩余空间百分比。
	disk_free_percent_rootfs	根文件系统磁盘空间占比。

3.2.2. 创建事件报警规则

本文为您介绍如何创建事件报警规则和调试系统事件，以便在E-MapReduce发生系统异常时，您能及时接收报警通知并处理异常。

前提条件

如果事件报警规则需要作用于指定应用分组的实例上，则请确保您已创建应用分组，且已将资源添加至该应用分组，操作方法请参见创建应用分组和添加资源至应用分组。

创建事件报警规则

1. 登录云监控控制台。
2. 在左侧导航栏，单击**报警服务 > 报警规则**。
3. 在**报警规则列表**页面，单击**事件报警**页签。
4. 在**事件报警**页面，单击**创建事件报警**。
5. 在**创建/修改事件报警**页面，设置事件报警规则和报警方式。
   i. 在**基本信息**区域，填写**时间报警规则**。
   ii. 在**事件报警规则**区域，选择事件类型。
   - 系统事件

     当**事件类型**选择**系统事件**时，您需要根据所需设置**产品类型**、**事件类型**、**事件等级**、**事件名称**、**资源范围**和**报警方式**。

     > **说明** 事件名称详细信息，请参见系统事件。

     系统事件支持的报警方式包括：**报警通知**、**消息服务队列**、**函数服务**、**URL回调**和**日志服务**。
   - 自定义事件。

     当**事件类型**选择**自定义事件**时，您需要根据所需设置**所属应用分组**、**事件名称**、**规则描述**和**通知方式**。
6. 单击**确定**。

调试系统事件

创建事件报警规则后，您可以使用系统事件的调试功能，验证系统事件报警规则中设置的消息服务队列、函数计算、URL回调和日志服务是否能正常被触发。

1. 登录云监控控制台。
2. 在左侧导航栏，单击**报警服务 > 报警规则**。
3. 在**报警规则列表**页面，单击**事件报警**页签。
4. 在**事件报警**页面，单击系统事件报警规则对应**操作**列的**调试**。
5. 在**创建事件调试**页面，选择待调试事件，根据实际情况修改调试内容。
6. 单击**确定**。
   云监控将根据内容发送一个事件，触发报警规则设置的报警通知、消息服务队列、函数计算、URL回调和日志服务。

系统事件

服务名	事件ID	事件名称（中文）	事件名称（英文）	事件含义
	EMR-330200049 EMR-350202005	NameNode发生主备切换	Maintenance:HDFS.NameNode.ActiveStandbySwitch	NameNode发生主备切换。 处理方式：需要判断主节点无法提供服务超时的原因，可能原因包括GC问题和block report storm等。
	EMR-350200012	NameNode发生OOM	Maintenance:HDFS.NameNode.OOM	NameNode发生OOM。 处理方式：您可以在HDFS服务的配置页面，调大**hadoop_namenode_heapsize**的值，以增加内存。
	EMR-350200005	HDFS发生目录格式化	Maintenance:HDFS.NameNode.DirectoryFormatted	NameNode元数据目录被删除。 处理方式：请提交工单或购买专家服务处理。 **警告** 禁止重启HDFS服务，以防数据丢失。
	EMR-350200006	NameNode加载FsImage异常	Maintenance:HDFS.NameNode.LoadFsImageException	NameNode读取FSImage异常。 处理方式：可能是由于云盘数据异常导致的，请提交工单处理。
	EMR-350200008	NameNode异常退出	Maintenance:HDFS.NameNode.ExitUnexpectely	NameNode节点异常退出。 处理方式：查看NameNode节点异常退出的日志，根据日志处理。
	EMR-350200015	HDFS写JournalNode超时	Maintenance:HDFS.NameNode.WriteToJournalNodeTimeout	NameNode写入JournalNode超时。 处理方式：检查JournalNode服务是否正常，同时检查JournalNode所在机器与网络情况是否有异常流量。

服务名	事件ID	事件名称（中文）	事件名称（英文）	事件含义
HDFS	EMR-350200014	NameNode资源不足	Maintenance:HDFS.NameNode.ResourceLow	NameNode节点磁盘空间不足，导致NameNode进入安全模式。 处理方式：检查Master节点*/mnt/disk1*的磁盘空间，是否有日志过大异常，如果没有异常，可以扩容磁盘。
	EMR-350200007	NameNode同步日志失败	Maintenance:HDFS.NameNode.SyncJournalFailed	NameNode操作JournalNode sync超时。 处理方式：检查JournalNode服务是否正常，同时检查JournalNode所在机器与网络情况是否有异常流量。
	EMR-350201008	DataNode OOM不能创建新的Native线程	Maintenance:HDFS.DataNode.OOM.UnableToCreateNewNativeThread	DataNode无法创建新的进程。 处理方式：通常是异常任务在该DataNode创建了大量线程，导致了DataNode无法创建新的线程，因此可以在Master节点通过root用户执行命令 `ps -eo nlwp,pid,args --sort nlwp |tail -n 5` ，找到使用线程数最大的5个进程，并进行分析。
	EMR-350202006	ZKFC监控NameNode健康状态时发生传输层异常事件	Maintenance:HDFS.ZKFC.TransportLevelExceptionInMonitorHealth	ZKFC检查NameNode健康度超时。 处理方式：检查NameNode超时的原因，可能原因包括NameNode内存大小、 GC或者是block report storm。如果无异常，集群规模较大，可以适当调大**ha.health-monitor.rpc-timeout.ms**配置，防止超时。
	EMR-350202002	ZKFC不能连接ZookeeperQuorum	Maintenance:HDFS.ZKFC.UnableToConnectToQuorum	ZKFC无法连接ZooKeeper。 处理方式：查看ZooKeeper服务，检查是否有异常连接没有释放导致的连接过多，以至于ZKFC无法连接ZooKeeper。
	EMR-350202001	ZKFC不能启动	Maintenance:HDFS.ZKFC.UnableToStartZKFC	无法启动ZKFC服务。 处理方式：查看ZKFC的日志，根据日志处理。
	EMR-250201009	JavaHeapSpace引起OOM错误	Maintenance:HDFS.DataNode.OomForJavaHeapSpace	DataNode发生OOM。 处理方式：您可以在HDFS服务的配置页面，调大的**hadoop_datanode_heapsize**值，以增加内存。
	EMR-250201004	DataNode进程异常退出	Maintenance:HDFS.DataNode.ExitUnexpected	DataNode异常退出。 处理方式：查看导致DataNode服务异常退出的日志，根据日志处理。

服务名	事件ID	事件名称（中文）	事件名称（英文）	事件含义
YARN	EMR-330300053	ResourceManager发生主备切换	Maintenance:YARN.ResourceManager.ActiveStandbySwitch	高可用集群ResourceManager发生主备切换。 处理方式：检查ResourceManager主备切换原因，可能原因包括GC问题、Master节点资源不足或内核错误等。
	EMR-350300015	YARN服务中ZKRMStateStore不能连接ZK	Maintenance:YARN.ResourceManager.ZKRMStateStoreCannotConnectZK	ResourceManager ZkClient无法连接ZooKeeper。 处理方式：检查ZooKeeper服务，是否由于异常连接没有释放导致无法连接。
	EMR-350300011	ResourceManager启动错误	Maintenance:YARN.ResourceManager.ErrorInStarting	ResourceManager启动错误。 处理方式：检查RM启动报错的原因，排查是否是由于错误配置导致的。
	EMR-350300010	ResourceManager切换到Active模式发生错误	Maintenance:YARN.ResourceManager.ErrorInTransitionToActiveMode	ResourceManager切换到Active模式时发生错误。 处理方式：查看切换报错的日志，根据日志处理。
	EMR-350300004	ResourceManager无效配置问题:不能找到RM_HA_ID	Maintenance:YARN.ResourceManager.InvalidConf.CannotFoundRM_HA_ID	ResourceManager错误配置导致找不到HA ID。 处理方式：在配置页面检查配置，是否由于修改配置导致了RM ID异常。
	EMR-350300013	ResourceManager异常退出	Maintenance:YARN.ResourceManager.ExitUnexpected	ResourceManager异常退出。 处理方式：查看导致ResourceManager服务异常退出的日志，根据日志处理。
	EMR-350302004	JobHistory服务启动错误	Maintenance:YARN.JobHistory.StartingError	JobHistory服务启动错误。 处理方式：查看导致JobHistory服务启动错误的日志，根据日志处理。
	EMR-350302003	JobHistory服务异常退出	Maintenance:YARN.JobHistory.ExitUnExpectedly	JobHistory服务异常退出。 处理方式：查看导致JobHistory服务异常退出的日志，根据日志处理。
	EMR-350303004	TimelineServer启动错误	Maintenance:YARN.TimelineServer.ErrorInStarting	Timeline Server启动错误。 处理方式：查看导致Timeline Server服务启动错误的日志，根据日志处理。
	EMR-350303003	TimelineServer异常退出	Maintenance:YARN.TimelineServer.ExitUnexpectedly	Timeline Server异常退出。 处理方式：查看导致Timeline Server服务异常退出的日志，根据日志处理。

服务名	事件ID	事件名称（中文）	事件名称（英文）	事件含义
	EMR-250300001	ResourceManager发生UnkownHostException异常	Maintenance:YARN.ResourceManager.UnkownHostException	ResourceManager对某些节点无法解析。 处理方式：检查是否使用了DNS服务，是否由于DNS服务不稳定导致。如果没有使用DNS服务，检查是否/etc/hosts文件配置了对应的host。
	EMR-350301010	磁盘错误导致不健康的NodeManager	Maintenance:YARN.NodeManager.UnHealthyForDiskFailed	磁盘错误导致不健康的NodeManager。 处理方式：检查NodeManager是否存在磁盘写满情况，存在的话请提交工单处理。
	EMR-250301006	NodeManager启动RebootingNodeStatusUpdater失败	Maintenance:YARN.NodeManager.ErrorRebootingNodeStatusUpdater	NodeManager重启NodeStatusUpdater失败。 处理方式：检查NodeManager重连日志，检查ResourceManager服务。
	EMR-250301015	NodeManager发生OOM	Maintenance:YARN.NodeManager.OOM	NodeManager发生OOM。 处理方式：检查NodeManager OOM的原因，排查是否有配置错误导致的内存泄露。
	EMR-350400016	HiveMetaStore发生JDBC通信异常	Maintenance:HIVE.HiveMetaStore.JdbcCommunicationException	Hive MetaStore发生JDBC通信异常。 处理方式：检查元数据连接，使用本地MySql客户端连接进行测试。
	EMR-350400010	HiveMetaStore数据库通信链路失败	Maintenance:HIVE.HiveMetaStore.DataBaseCommunicationLinkFailure	Hive MetaStore数据库连接异常，提示通信链路失败。 处理方式：检查元数据连接，使用本地MySql客户端连接进行测试。
	EMR-350400009	HiveMetaStore数据库连接失败	Maintenance:HIVE.HiveMetaStore.DataBaseConnectionFailed	Hive MetaStore数据库连接失败。 处理方式：检查元数据连接，使用本地MySql客户端连接进行测试。
	EMR-350400014	HiveMetaStore发生OOM	Maintenance:HIVE.HiveMetaStore.OomOccured	Hive MetaStore发生OOM。 处理方式：您可以在Hive服务的配置页面，调大hive_meastore_heapsize的值，以增加内存。
	EMR-350400015	HiveMetastore数据库磁盘空间用尽	Maintenance:HIVE.HiveMetaStore.DataBaseDiskQuotaUsedup	元存储连接超过限制。 处理方式：如果使用本地MySql或者RDS作为元数据存储，需要手动提升限制大小。如果使用统一元数据作为元数据存储，请提交工单处理。

服务名	事件ID	事件名称（中文）	事件名称（英文）	事件含义
HIVE	EMR-350400006	HiveMetastore超过最大查询数	Maintenance:HIVE.HiveMetaStore.MaxQuestionsExceeded	元数据存储连接超过限制。 处理方式：如果使用本地MySql或者RDS作为元数据存储，需要手动提升限制大小。如果使用统一元数据作为元数据存储，请提交工单处理。
	EMR-350400007	HiveMetastore超过最大更新数	Maintenance:HIVE.HiveMetaStore.MaxUpdatesExceeded	元数据存储连接超过限制。 处理方式：如果使用本地MySql或者RDS作为元数据存储，需要手动提升限制大小。如果使用统一元数据作为元数据存储，请提交工单处理。
	EMR-350400005	HiveMetastore超过最大用户连接数	Maintenance:HIVE.HiveMetaStore.MaxUserConnectionExceeded	元数据存储连接超过限制。 处理方式：如果使用本地MySql或者RDS作为元数据存储，需要手动提升限制大小。如果使用统一元数据作为元数据存储，请提交工单处理。
	EMR-350400002	HiveMetastore配置文件解析错误	Maintenance:HIVE.HiveMetaStore.ParseConfError	配置异常导致解析错误。 处理方式：通常是由于配置异常导致解析错误，因此请检查*Hivemetastore-site.xml*里面的配置，排查配置历史信息并修复问题。
	EMR-350400008	HiveMetastore请求的表丢失	Maintenance:HIVE.HiveMetaStore.RequiredTableMissing	Hive MetaStore的元存储表丢失。 处理方式：检查Metastore版本和元数据存储的Hive版本是否一致，并查看元数据是否损坏。
	EMR-350401013	hiveServer2发生OOM	Maintenance:HIVE.HiveServer2.HiveServer2OOM	HiveServer2发生OOM。 处理方式：您可以在Hive服务的配置页面，调大**hive_server2_heapsize**的值，以增加内存。
	EMR-350401007	不能通过提供的URIs连接到hiveServer2	Maintenance:HIVE.HiveServer2.CannotConnectByAnyURIsProvided	HiveServer2无法连接Hive MetaStore。 处理方式：检查MetaStore服务。
	EMR-350401004	hiveServer2连接ZK超时	Maintenance:HIVE.HiveServer2.ConnectToZkTimeout	HiveServer2连接Zookeeper超时。 处理方式：检查ZooKeeper服务。
	EMR-350401008	hiveServer2启动错误	Maintenance:HIVE.HiveServer2.ErrorStartingHiveServer	HiveServer2启动异常。 处理方式：查看启动异常的日志，定位问题原因并处理。

服务名	事件ID	事件名称（中文）	事件名称（英文）	事件含义
	EMR-350401006	hiveServer2初始化MetaStore客户端失败	Maintenance:HIVE.HiveServer2.FailedInitMetaStoreClient	HiveServer2初始化MetaStore客户端失败。 处理方式：检查HiveServer2的异常日志，定位无法初始化问题。
	EMR-350401005	hiveServer2连接MetaStoreServer失败	Maintenance:HIVE.HiveServer2.FailedToConnectToMetaStoreServer	HiveServer2连接不上MetaStoreServer。 处理方式：检查Metastore服务
HOST	EMR-350100001	主机/var/log/message有OOM异常	Maintenance:HOST.OomFoundInVarLogMessage	主机*/var/log/message*中存在OOM异常日志。 处理方式：通常是因为主机内存不足而终止某进程。因此请检查内存分配是否合理，如果内存偏小，请增加内存。
	EMR-250100006	主机CPU卡顿	Maintenance:HOST.CpuStuck	Linux内核报警存在CPU Stuck。 处理方式：检查是否存在异常的任务长时间占据CPU。
SparkHIstory	EMR-350900001	SparkHistory发生OOM	Maintenance:SPARK.SparkHistory.OomOccured	Spark History服务发生OOM。 处理方式：您可以在Spark服务的配置页面，调大**spark_history_daemon_memory**的值，以增加内存。
Zookeeper	EMR-350500001	ZOOKEEPER不能运行QuorumServer	Maintenance:ZOOKEEPER.UnableToRunQuorumServer	Zookeeper无法启动。 处理方式：检查异常日志和配置文件，确认是否ZNode过多导致的OOM。
	EMR-230500059	Zookeeper发生主从切换	Maintenance:ZOOKEEPER.LeaderFollowerSwitch	ZooKeeper发生主备节点切换。 处理方式：请检查切换原因，根据实际情况确认是否重启ZooKeeper服务。

4.运维管理指南
4.1. 元数据管理（旧版功能）
4.1.1. Hive元数据管理
4.1.1.1. Hive统一元数据

EMR-2.4.0之前版本，所有集群采用的是集群本地的MySQL数据库作为Hive元数据库；EMR-2.4.0及后续版本，E-MapReduce（简称EMR）支持统一的高可靠的Hive元数据库。

前提条件

创建集群时，详情请参见创建集群。

背景信息

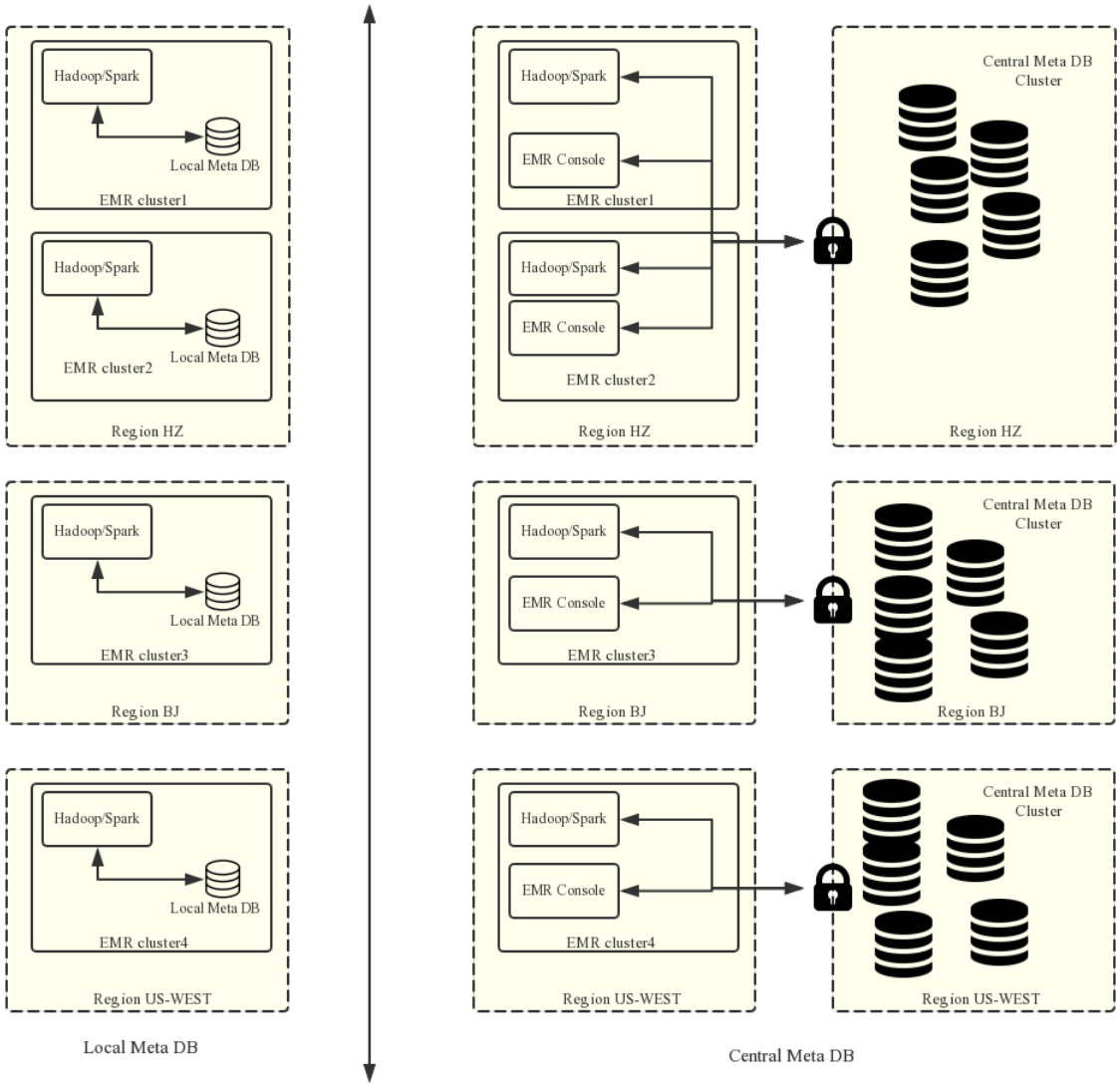
因为元数据库需要使用公网IP来连接，所以集群必须要有公网IP，同时请不要随意的切换公网IP地址，防止对应的数据库白名单失效。

如果是本地的元数据库，您可以使用集群上的Hue工具来管理。

E-MapReduce后台RDS统一管理元数据的方式，仅限小容量的用户使用。对于大容量场景，建议您自建RDS作为统一元数据。默认限制为：

- 总容量：200MiB。
- 小时query数量限制：720000/h。
- 小时update数量限制：144000/h。

介绍



统一的元数据管理，可以实现：

- 持久化的元数据存储。

  支持统一元数据之前，元数据都是在集群内部的MySQL数据库，元数据会随着集群的释放而丢失，特别是EMR提供了灵活按量模式，集群可以按需创建用完就释放。如果您需要保留现有的元数据信息，必须登录集群手动将元数据信息导出。

  支持统一元数据之后，释放集群不会清理元数据信息。所以，在任何时候删除OSS上或者集群HDFS上数据（包括释放集群操作）的时候，需要先确认该数据对应的元数据已经删除（即要删掉数据对应的表和数据库），否则元数据库中可能出现一些脏数据。

- 计算存储分离。

  EMR上可以支持将数据存放在阿里云OSS中，在大数据量的情况下将数据存储在OSS上会大大降低使用的成本，EMR集群主要用来作为计算资源，在计算完成之后可以随时释放，数据在OSS上，同时也不用再考虑元数据迁移的问题。

- 数据共享。

  使用统一的元数据库，如果您的所有数据都存放在OSS之上，则不需要做任何元数据的迁移和重建，所有集群都是可以直接访问数据，这样每个EMR集群可以做不同的业务，但是可以很方便地实现数据的共享。

创建使用统一元数据的集群

支持以下两种方式创建统一元数据的集群：

- 页面方式。

  创建集群时，在**基础配置**页面，选中**统一meta数据库**。

- API方式。

使用CreateClusterV2创建集群，详情请参见CreateClusterV2。

> **说明** 需指定参数：useLocalMetaDb=false。

表管理

详细请参见Hive元数据基本操作。

查看元数据库信息

1. 进入元数据管理页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**元数据管理**页签。
2. 在左侧导航栏，单击**元数据库信息**。

   您可以查看当前RDS的用量和使用限制，如果需要修改，请提交工单。

4.1.1.2. Hive元数据基本操作

本文为您介绍Hive元数据的基本操作，包括新建库、删除库、新建表和删除表。

前提条件

已创建集群，详情请参见创建集群。

新建库

1. 进入元数据管理页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**元数据管理**页签。
2. 在**表管理**页面，单击**新建库**。
3. 在**新建库**对话框中，配置各项参数。

   因为EMR表管理功能仅支持基于OSS文件系统创建数据库和表，所以**数据来源**设置为**OSS**。数据库和表的文件路径不能选择整个OSS bucket，需要选择到OSS bucket的下级目录。
4. 单击**确定**。

   您可以单击**任务列表**，查看执行结果。
   - 当**状态**为**成功**时，表示操作成功。
   - 当**状态**为**失败**时，您可以单击**操作**列的**查看详情**，排查失败的原因。

新建表

> **注意** 当前支持创建外部表和使用分区表。

1. 进入元数据管理页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**元数据管理**页签。
2. 在**表管理**页面的中间区域，单击已创建的元数据库。
3. 单击右上角的**新建表**。
4. 在**新建表**对话框中，配置各项参数。

配置项	操作
表名	表名称。
字段分隔符	从**字段分隔符**中选择分隔符或者自定义。

配置项	操作
外部表	默认不选中。 需要创建外部表时，执行以下操作： i. 选中**外部表**复选框，单击图标，选择文件路径。 ii. 单击**新增列**，设置相关的参数。
使用分区	默认**否**。 需要创建使用分区表时，执行以下操作： i. 单击**是**。 ii. 单击**新建分区列**，设置相关的参数。

5. 单击**确定**。
   您可以单击**任务列表**，查看执行结果。
   - 当**状态**为**成功**时，表示操作成功。
   - 当**状态**为**失败**时，您可以单击**操作**列的**查看详情**，排查失败的原因。

删除表

1. 进入元数据管理页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**元数据管理**页签。
2. 在**表管理**页面的中间区域，单击已创建的元数据库。
3. 单击待删除表**操作**列的**删除**。
4. 在**删除表**对话框中，单击**确认**。
   您可以单击**任务列表**，查看执行结果。
   - 当**状态**为**成功**时，表示操作成功。
   - 当**状态**为**失败**时，您可以单击**操作**列的**查看详情**，排查失败的原因。

删除库

> **注意** 删除数据库之前，必须删除数据库下所有的表。

1. 进入元数据管理页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**元数据管理**页签。
2. 在**表管理**页面的中间区域，单击已创建的元数据库。
3. 单击待删除库**操作**列的**删除**。
4. 在**删除库**对话框中，单击**确认**。
   您可以单击**任务列表**，查看执行结果。
   - 当**状态**为**成功**时，表示操作成功。
   - 当**状态**为**失败**时，您可以单击**操作**列的**查看详情**，排查失败的原因。

4.1.1.3. Hive元数据迁移

4.1.1.3.1. 从统一元数据库迁出到用户自建的RDS实例

为了保证更稳定的大规模Hive元数据服务，您可以从原有的统一元数据库迁出到您自建的RDS实例。

前提条件

已购买RDS，详情请参见创建RDS MySQL实例。

使用限制

- 建议选择**MySQL**的5.7版本；**系列**选择**高可用版**。

- RDS MySQL实例须与E-MapReduce的实例处于同一个安全组，以便RDS与E-MapReduce可以通过内网地址互通。

操作步骤

1. 在RDS中创建一个**database**，名称为**hivemeta**，同时创建一个用户，把**hivemeta**的读写权限赋给这个用户，详情请参见配置独立RDS。
2. 导出统一元数据库的内容（只导出数据，不用导表结构）。
   i. 为保证数据的一致性， 在Hive服务页面停止Hive的MetaStore服务，保证导出期间不会有新的元数据变化，详情请参见停止Hive的MetaStore服务。
   ii. 在Hive服务页面，单击**配置**页签。
   iii. 在配置页面，查找**javax.jdo.option.ConnectionUserName**、**javax.jdo.option.ConnectionPassword**和**javax.jdo.option.ConnectionURL**的值。

   > **说明**
   > - 如果是老版本集群，请在*$HIVE_CONF_DIR/hive-site.xml*中查找。
   > - javax.jdo.option.ConnectionUserName表示对应数据库用户名；javax.jdo.option.ConnectionPassword表示对应数据库访问密码；javax.jdo.option.ConnectionURL对应数据库访问地址和库名。

   iv. 进入集群Master节点，执行以下命令。

   ```
   mysqldump -t DATABASENAME -h HOST -P PORT -u USERNAME -p PASSWORD > /tmp/metastore.sql
   ```

   > **说明** 密码为上一步骤在配置页面获取的密码。

3. 修改集群Master节点上的*/usr/local/emr/emr-agent/run/meta_db_info.json*，把里面的**use_local_meta_db**设置为**false**，meta数据库的链接地址、用户名和密码换成RDS的信息。

   > **说明**
   > - 对于无此文件的集群，直接忽略此步骤。
   > - 如果是HA集群，两个Master节点都需要进行操作。

4. 在Hive配置页面，把元数据库的链接地址、用户名和密码换成新RDS的信息。
   如果是老版本集群，修改*$HIVE_CONF_DIR/hive-site.xml*中对应的配置为需要连接的数据库。
5. 在一台Master节点上，将*hive-site.xml*中的元数据库链接地址、用户名和密码换成RDS的信息，然后根据您集群的Hive版本初始化Schema。
   - 如果Hive是2.3.x版本时，请执行以下命令进行初始化。
     a. 使用ssh方式登录集群Master节点，详情请参见登录集群。
     b. 执行以下命令，进入mysql目录。

     ```
     cd /usr/lib/hive-current/scripts/metastore/upgrade/mysql/
     ```

     c. 执行以下命令，登录MySQL数据库。

     ```
     mysql -h {RDS数据库内网或外网地址} -u{RDS用户名} -p{RDS密码}
     ```

     d. 执行以下命令，进行初始化。

     ```
     use {RDS数据库名称};
     source /usr/lib/hive-current/scripts/metastore/upgrade/mysql/hive-schema-2.3.0.mysql.sql;
     ```

     > **说明** {RDS数据库名称} 为步骤1中创建的数据库，例如hivemeta。

   - 其它Hive版本时，执行以下命令进行初始化。
     a. 使用ssh方式登录集群Master节点，详情请参见登录集群。
     b. 执行以下命令，进入*bin*目录。

     ```
     cd /usr/lib/hive-current/bin
     ```

     c. 执行以下命令，登录MySQL数据库。

     ```
     ./schematool -initSchema -dbType mysql
     ```

6. 把之前导出来的Meta数据导入RDS，您可以通过以下命令行登录MySQL。

   ```
   mysql -h {rds的url} -u {rds的用户名} -p
   ```

进入MySQL的命令行之后，执行命令 `source /tmp/metastore.sql` ，正常情况下可以完全导入，不会报错。

7. 重启Hive所有组件。

   验证Hive功能是否正常，可以在Master节点上，执行 `hive cli` ，确认数据是否和先前一样。

停止Hive的MetaStore服务

1. 进入Hive服务页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务 > Hive**。
2. 在Hive服务页面的**组件列表**区域，单击**Hive MetaStore**所在行的**停止**。
   i. 在**执行集群操作**对话框中，选择**执行范围**、**是否滚动执行**和**失败处理策略**，输入**执行原因**。
   ii. 单击**确定**。

重启Hive所有组件

1. 进入Hive服务页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务 > Hive**。
2. 单击**配置**页签。
3. 在右上角选择**操作 > 重启 All Components**。
   i. 在**执行集群操作**对话框中，选择**执行范围**、**是否滚动执行**和**失败处理策略**，输入**执行原因**。
   ii. 单击**确定**。

4.1.2. Kafka元数据管理

本文为您介绍Kafka元数据的基本操作。例如，添加Topic、删除Topic和查看Topic详情等操作。

添加Topic

1. 进入Kafka数据管理页。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**元数据管理**页签。
   iv. 在左侧导航栏，单击**Kafka数据管理**。
2. 在**Kafka数据管理**页面，单击右上角的**添加Topic**。
3. 在**基本配置**区域，配置各项参数。
4. （可选）在**高级配置**区域，可单击右上角，输入Key和Value增加配置项。
5. 单击**确定**。

   您可以单击**任务列表**，查看执行结果。
   - 当**状态**为**成功**时，表示操作成功。
   - 当**状态**为**失败**时，您可以单击**操作**列的**查看详情**，排查失败的原因。

删除Topic

> 注意
> - 内置的Topic（__consumer_offsets和_schemas）不支持删除。
> - 已经释放的集群不支持删除。

1. 进入Kafka数据管理页。
   i. 登录阿里云E-MapReduce控制台。

ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
iii. 单击上方的**元数据管理**页签。
iv. 在左侧导航栏，单击**Kafka数据管理**。
2. 在**Kafka数据管理**页面，单击对应Topic所在行的**删除**。
3. 单击**确定**。
您可以单击**任务列表**，查看执行结果。
- 当**状态**为**成功**时，表示操作成功。
- 当**状态**为**失败**时，您可以单击**操作**列的**查看详情**，排查失败的原因。

更新配置

> 注意
> - 内置的Topic（__consumer_offsets和_schemas）不支持更新。
> - 已经释放的集群不支持更新。

1. 进入Kafka数据管理页。
i. 登录阿里云E-MapReduce控制台。
ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
iii. 单击上方的**元数据管理**页签。
iv. 在左侧导航栏，单击**Kafka数据管理**。
2. 在**Kafka数据管理**页面，单击对应Topic所在行的**更新配置**。
3. 修改相应的配置信息。
4. 单击**确定**。
您可以单击**任务列表**，查看执行结果。
- 当**状态**为**成功**时，表示操作成功。
- 当**状态**为**失败**时，您可以单击**操作**列的**查看详情**，排查失败的原因。

服务监控

1. 进入Kafka数据管理页。
i. 登录阿里云E-MapReduce控制台。
ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
iii. 单击上方的**元数据管理**页签。
iv. 在左侧导航栏，单击**Kafka数据管理**。
2. 在**Kafka数据管理**页面，单击对应Topic所在行的**监控**。

服务运维

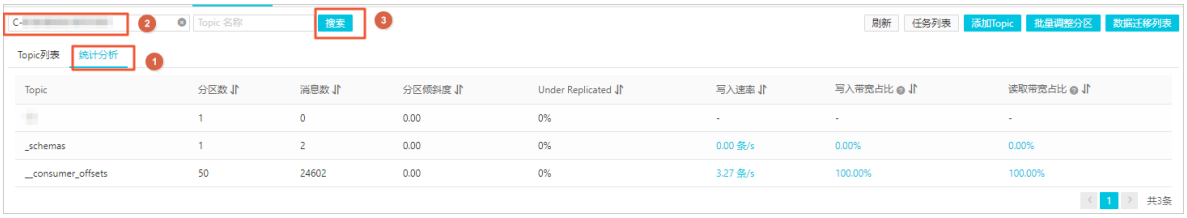
1. 进入Kafka数据管理页。
i. 登录阿里云E-MapReduce控制台。
ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
iii. 单击上方的**元数据管理**页签。
iv. 在左侧导航栏，单击**Kafka数据管理**。
2. 在**Kafka数据管理**页面，单击对应Topic所在行的**运维**。
跳转到Kafka的集群服务页面，可对其组件进行相关的操作。

查看Topic详情

1. 进入Kafka数据管理页。
i. 登录阿里云E-MapReduce控制台。
ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
iii. 单击上方的**元数据管理**页签。
iv. 在左侧导航栏，单击**Kafka数据管理**。
2. 在**Kafka数据管理**页面，单击对应Topic所在行的**详情**。
您可以查看该Topic的监控数据、摘要信息、数据分布和消费组。

查看统计分析

1. 进入Kafka数据管理页。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**元数据管理**页签。
   iv. 在左侧导航栏，单击**Kafka数据管理**。
2. 在**Kafka数据管理**页面，单击**统计分析**页签。
3. 在上方选择集群ID，单击**搜索**。
   统计分析区域展示当前集群下所有Topic的信息。



批量调整分区

1. 进入Kafka数据管理页。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**元数据管理**页签。
   iv. 在左侧导航栏，单击**Kafka数据管理**。
2. 在**Kafka数据管理**页面，单击**批量调整分区**。
3. 在**批量调整分区**对话框中，单击**选择Topic**。
4. 在**选择Topic**对话框中，从**集群 ID**列表选择待调整分区所属的集群。
5. 选中待调整的分区，单击**确定**。
6. 设置调整后的分区数。
7. 单击**确定**。
   您可以单击**任务列表**，查看执行结果。
   - 当**状态**为**成功**时，表示操作成功。
   - 当**状态**为**失败**时，您可以单击**操作**列的**查看详情**，排查失败的原因。

4.1.3. 元数据管理常见问题

本文汇总了使用E-MapReduce的元数据管理时的常见问题。

- 报错提示“oss://yourbucket/***/***/***”或“hdfs://yourhost:9000/***/***/***”路径不存在，该如何处理？
- 删除Hive Database时提示“java.lang.IllegalArgumentException: java.net.UnknownHostException: *****”
- 如何登录内置的MySQL？

报错提示“oss://yourbucket/***/***/***”或“hdfs://yourhost:9000/***/***/***”路径不存在，该如何处理？

问题分析：由于删除OSS上的表数据之前，没有删除数据表对应的元数据，导致表的Schema还存在，但实际的数据已经不存在或已经移动到别的路径。

解决方法：可以先修改表的Location为一个存在的路径，然后再删除表。

```
alter table test set location 'oss://your_bucket/your_folder'
```

> 说明 *oss://your_bucket/your_folder*必须是一个存在的OSS路径。

删除Hive Database时提示“java.lang.IllegalArgumentException: java.net.UnknownHostException: *****”

问题分析：由于在之前的集群上创建了Hive的数据库，并且数据库的位置是落在之前集群的HDFS之上，但是在集群释放的时候，没有清理掉对应的Hive Database，导致新建集群之后，无法访问到之前已经释放集群的HDFS数据。如果在HDFS上的数据库和表是手动创建的，在释放集群时候需要清理。

解决方法：

1. 登录集群的Master节点，在*$HIVE_CONF_DIR/hivemetastore-site.xml*中，找到对应数据库的访问地址和用户名密码信息。

```
javax.jdo.option.ConnectionUserName //对应数据库用户名；
javax.jdo.option.ConnectionPassword //对应数据库访问密码；
javax.jdo.option.ConnectionURL //对应数据库访问地址和库名；
```

```
<property>
  <name>javax.jdo.option.ConnectionUserName</name>
  <value>root</value>
</property>
<property>
  <name>hive.metastore.server.min.threads</name>
  <value>200</value>
</property>
<property>
  <name>hive.service.metrics.file.frequency</name>
  <value>30000</value>
</property>
<property>
  <name>hive.service.metrics.file.location</name>
  <value>/tmp/hivemetastore_metric.json</value>
</property>
<property>
  <name>javax.jdo.option.ConnectionPassword</name>
  <value>EM          </value>
</property>
<property>
  <name>javax.jdo.option.ConnectionURL</name>
  <value>jdbc:mysql://em          /          ?createDatabaseIfNotExist=true&amp;amp;characterEncoding=UTF-8</value>
</property>
<property>
  <name>javax.jdo.option.ConnectionDriverName</name>
  <value>com.mysql.jdbc.Driver</value>
</property>
```

2. 在集群Master节点上输入以下命令，并根据提示信息输入密码登录创建的Hive数据库。

```
mysql -h ${DBConnectionURL} -u ${ConnectionUserName} -p
```

3. 登录Hive数据库后，修改Location为该Region真实存在的OSS路径即可。

```
mysql> show databases;
+-------------------------------------------------+
| Database                                        |
+-------------------------------------------------+
| information_schema                              |
| xxxxxxxxx77ac43c3bd0efae77e0bf1947d45fb4c896fb99 |
+-------------------------------------------------+

mysql> use xxxxxxxxx77ac43c3bd0efae77e0bf1947d45fb4c896fb99;

mysql> select * from dbs;
+-------+------------------------+-------------------------------+---------+------------+------------+
| DB_ID | DESC                   | DB_LOCATION_URI               | NAME    | OWNER_NAME | OWNER_TYPE |
+-------+------------------------+-------------------------------+---------+------------+------------+
|     1 | Default Hive database | oss://mybucket/hive/warehouse | default | public     | ROLE       |
|     6 | NULL                   | hdfs://dirty-hostname/warehouse | dirty_db | NULL       | USER       |
+-------+------------------------+-------------------------------+---------+------------+------------+

mysql> update dbs set DB_LOCATION_URI = 'oss://your-bucket/your-db-folder' where DB_ID = 6;
```

如何登录内置的MySQL？

1. 通过SSH方式连接集群。

   详情请参见登录集群。

2. 执行以下命令，登录内置的MySQL。

```
mysql -uroot -pEMRroot1234
```

   内置MySQL登录的用户名为root，密码为EMRroot1234。

4.2. 监控大盘（Beta）

4.2.1. 概览

监控大盘是供E-MapReduce用户，特别是集群运维人员使用的包含监控集群、服务和作业整体运行状况、排查和解决集群、服务以及作业问题的一套完整工具的产品。

监控大盘包含概述、作业列表、事件中心和日志中心功能。

事件概览

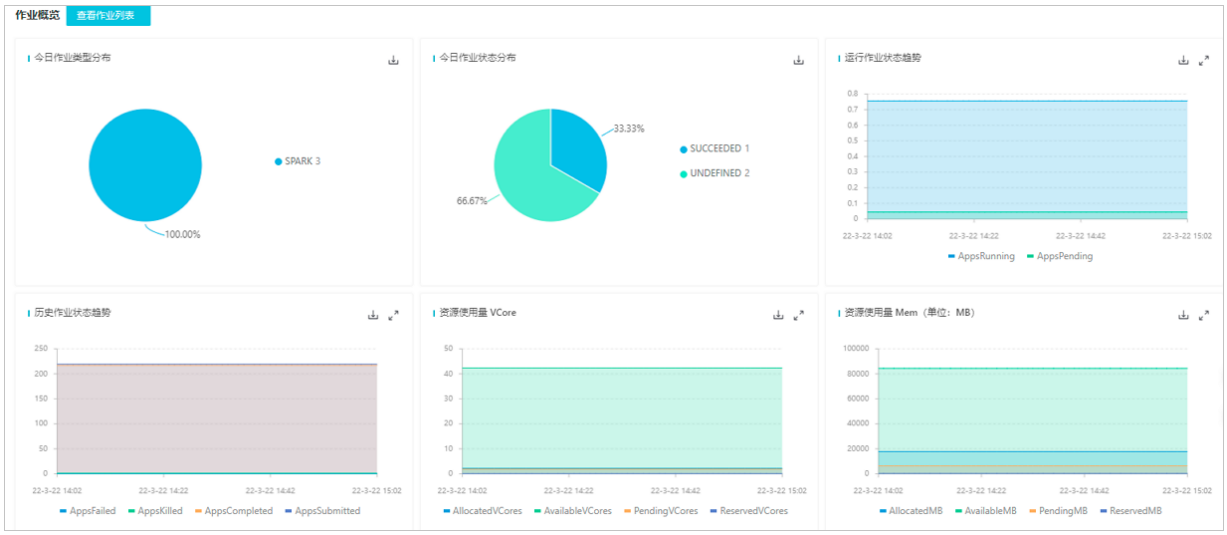
事件概览区域，为您展示所有集群的CRITICAL事件。单击**查看全部事件**，可以查看所有事件的详细信息，详情请参见事件中心。

集群概览

集群概览区域，为您展示集群对应的事件和服务监控状态。单击**查看集群指标**，可以查看主要服务的详细监控指标，详情请参见概述。

作业概览

作业概览区域，为您展示今日作业的类型分布、状态分布、运行作业状态趋势、历史作业状态趋势和资源使用量。单击**查看作业列表**，可以查看作业整体运行状况，详情请参见作业列表。



4.2.2. 集群指标

4.2.2.1. 概述

集群指标提供对集群上安装的主要服务的详细监控功能，包括HDFS 、YARN、Hive、Kafka和Zookeeper等。

前提条件

已创建集群，详情请参见创建集群。

查看集群指标

1. 进入监控大盘。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。

iii. 单击上方的**监控大盘**。

2. 在概览页面，单击上方的**集群指标**页签。
3. 选择集群ID和服务，可以查看服务的指标。

   各服务指标如下：

   - HOST指标
   - HDFS指标
   - YARN指标
   - Hive指标
   - ZooKeeper指标
   - Kafka指标
   - Impala指标
   - HUE指标
   - Kudu指标
   - ClickHouse指标
   - Flink指标
   -

4.2.2.2. HOST指标

本文为您介绍HOST指标的详细信息。

参数	描述
Boot time seconds	运行时间。
CPU num	CPU核数。
MemTotal bytes	总内存。
CPU utilization	CPU使用率。
Root FS Used	根目录使用率。
CPU utilization-system	系统CPU使用率。
CPU utilization-user	用户CPU使用率。
CPU utilization-idle	等待IO完成的CPU使用率。
MemTotal bytes	系统总内存。
Memory used bytes	系统已使用内存。
Average memory usage	平均内存使用率。
Memory Buffers bytes	系统Buffers中的内存。
Memor MemFree bytes	系统空闲内存。
Memor Cached bytes	系统Cached中的内存。
Filesystem size bytes	磁盘大小。
Filesystem avail bytes	磁盘可用空间。
Average disk usage	整体总磁盘与整体平均磁盘使用率。
Disk read time seconds total	每次IO读取耗时。
Disk write time seconds total	每次IO写入耗时。
Disk read rate	磁盘读取速率。
Disk write rate	磁盘写入速率。
Network bandwidth acceptance per second	每秒网络带宽接收量。

参数	描述
Network broadband traffic per second	每秒网络带宽发送量。
node_network_receive_packets_total	网络接收数据包数量。
node_network_transmit_packets_total	网络发送数据包数量。
Network transmit drop total	发送的丢弃的数据包总数。
Networ receive drop total	接收的丢弃的数据包总数。
Networ transmit errs total	发送的错误数据包总数。
Networ receive errs total	接收的错误数据包总数。
Procs running	运行中的进程数目。
Proc blocked	阻塞的进程数目。
Socksta TCP alloc	网络Socket连接信息，已分配的TCP套接字数量。
Sockstat TCP tw	网络Socket连接信息，等待关闭的TCP连接。
Sockstat UDP inuse	网络Socket连接信息，正在使用的UDP套接字数量。
Socksta sockets used	网络Socket连接信息，已使用的所有协议套接字总数。

4.2.2.3. HDFS指标

本文为您介绍HDFS指标的详细信息。

HDFS指标包含以下部分：

- HDFS-HOME
- HDFS-NameNodes
- HDFS-DataNodes
- HDFS-JournanlNodes

HDFS-HOME

参数	指标	描述
NameNodeActive	hdfs_namenode_status	NameNode状态： • 0：Standby • 1：Active
Decomming DataNodes	hdfs_namenode_fsnamesystem_NumDecommissioningDataNodes	NameNode正在执行下线（Decommission）操作的DataNode的个数。
Stale DataNodes	hdfs_namenode_fsnamesystem_StaleDataNodes	NameNode由于Heartbeat延迟被标记为stale的DataNode个数，默认是超过三个心跳时间。
Live DataNodes	hdfs_namenode_fsnamesystem_NumLiveDataNodes	存活的DataNode个数。
Dead DataNodes	hdfs_namenode_fsnamesystem_NumDeadDataNodes	处于Dead状态的DataNode个数。
NameNodeInSafeMode	hdfs_namenode_safemode_status	NameNode是否处于SafeMode状态。
Corrupt Blocks	hdfs_namenode_fsnamesystem_CorruptBlocks	NameNode损坏的块数。
UnderRepl Blocks	hdfs_namenode_fsnamesystem_UnderReplicatedBlocks	NameNode不足副本的块数。

参数	指标	描述
Failed Volumes	hdfs_namenode_fsnamesystem_VolumeFailuresTotal	NameNode存盘算坏的总数。
HDFS Capacity	hdfs_namenode_fsnamesystem_CapacityRemainingGB	NameNode剩余的空间大小，单位GB。
	hdfs_namenode_fsnamesystem_CapacityTotalGB	NameNode配置的总的空间大小，单位GB。
	hdfs_namenode_fsnamesystem_CapacityUsedGB	NameNode已经使用的空间大小，单位GB。
Total Files	hdfs_namenode_fsnamesystem_TotalFiles	NameNode所有的文件和目录个数之和。
HDFS Block Metrics	hdfs_namenode_fsnamesystem_BlocksTotal	NameNode总的块数。
	hdfs_namenode_fsnamesystem_CorruptBlocks	NameNode损坏的块数。
	hdfs_namenode_fsnamesystem_ExcessBlocks	NameNode超过副本数的块数。
	hdfs_namenode_fsnamesystem_MissingBlocks	NameNode丢失的块数。
	hdfs_namenode_fsnamesystem_PendingDeletionBlocks	NameNode等待删除的块数。
	hdfs_namenode_fsnamesystem_PendingReplicationBlocks	NameNode等待复制的块数。
	hdfs_namenode_fsnamesystem_PostponedMisreplicatedBlocks	NameNode启动时被推迟处理的块数。
	hdfs_namenode_fsnamesystem_UnderReplicatedBlocks	NameNode不足副本数。
Number of Files Under Construction	hdfs_namenode_fsnamesystem_NumFilesUnderConstruction	正在写入的文件数量。
HDFS Total Load	hdfs_namenode_fsnamesystem_TotalLoad	整个集群的客户端连接数。
Expired Heartbeats	hdfs_namenode_fsnamesystem_ExpiredHeartbeats	NameNode过期心跳数量。

HDFS-NameNodes

参数	指标	描述
Memory Metrics	hdfs_namenode_jvm_MemHeapMaxM	NameNode设置的JVM最大内存大小，单位为MB。
	hdfs_namenode_jvm_MemMaxM	NameNode可用的最大内存，单位为MB。
	hdfs_namenode_jvm_MemHeapUsedM	NameNode堆使用的内存大小，单位为MB。
	hdfs_namenode_jvm_MemNonHeapUsedM	NameNode非堆使用的内存大小，单位为MB。
	hdfs_namenode_jvm_MemHeapCommittedM	NameNode对内存commit内存大小，单位为MB。
	hdfs_namenode_jvm_MemNonHeapUsedM	NameNode非堆内存使用大小，单位为MB。
	hdfs_namenode_jvm_MemNonHeapCommittedM	NameNode非堆内存commit使用大小，单位为MB。

参数	指标	描述
	hdfs_namenode_jvm_MemNonHeapMaxM	NameNode非堆内存最大值，单位为MB。
Thread Metrics	hdfs_namenode_jvm_ThreadsTerminated	NameNode进程中结束的线程个数。
	hdfs_namenode_jvm_ThreadsTimedding	NameNode进程中等待一定时间后运行的线程个数。
	hdfs_namenode_jvm_ThreadsWaiting	NameNode进程中等待运行的线程个数。
	hdfs_namenode_jvm_ThreadsBlocked	NameNode进程中由于资源或锁原因被组织的线程个数。
GC Count	hdfs_namenode_jvm_GcCount	NameNode GC次数。
Gc Time	hdfs_namenode_jvm_GcTimeMillis	NameNode GC时间。
NameNode Started Time	hdfs_namenode_namenodeinfo_NNStartedTimeInMillis	NameNode启动的时间，单位为毫秒。
NameNode HA Transition Time	hdfs_namenode_ha_transition_time	NameNode上一次HA切换时间。
RPC Queue Num Ops	hdfs_namenode_rpc_client_activity_RpcQueueTimeNumOps	NameNode RPC Client端口被调用次数。
RPC Queue Time	hdfs_namenode_rpc_client_activity_RpcQueueTimeAvgTime	NameNode RPC Client端口队列平均耗时。
RPC Processing Num Ops	hdfs_namenode_rpc_client_activity_RpcProcessingTimeNumOps	NameNode RPC Client端口被调用次数。
RPC Processing Time	hdfs_namenode_rpc_client_activity_RpcProcessingTimeAvgTime	NameNode RPC Client端口平均处理时间。
RPC CallQueueLength	hdfs_namenode_rpc_client_activity_CallQueueLength	NameNode RPC Client端口队列积压长度。
RPC Nework Connections	hdfs_namenode_rpc_service_activity_NumDroppedConnections	NameNode RPC Server端口中断的连接数。
	hdfs_namenode_rpc_service_activity_NumOpenConnections	NameNode RPC Server端口正在连接的连接数。
Rpc Slow Calls	hdfs_namenode_rpc_service_activity_RpcSlowCalls	NameNode RPC Server端口慢连接数。
NameNode Memory Threshold	hdfs_namenode_memory_threshold	NameNode内存水位预估。
GetFileInfo Num Ops	hdfs_namenode_rpc_client_detailed_activity_GetFileInfoNumOps	NameNode RPC Client端口GetFileInfo调用次数。
GetFileInfo Avg Time	hdfs_namenode_rpc_client_detailed_activity_GetFileInfoAvgTime	NameNode RPC Client端口GetFileInfo调用平均时间。
GetBlockLocations Num Ops	hdfs_namenode_rpc_client_detailed_activity_GetBlockLocationsNumOps	NameNode RPC Client端口clientGetBlockLocations调用次数。
GetBlockLocations Avg Time	hdfs_namenode_rpc_client_detailed_activity_GetBlockLocationsAvgTime	NameNode RPC Client端口clientGetBlockLocations调用平均时间。
Complete Num Ops	hdfs_namenode_rpc_client_detailed_activity_CompleteNumOps	NameNode RPC Client端口Complete调用次数。
Complete Avg Time	hdfs_namenode_rpc_client_detailed_activity_CompleteAvgTime	NameNode RPC Client端口Complete调用平均时间。
Create Num Ops	hdfs_namenode_rpc_client_detailed_activity_CreateNumOps	NameNode RPC Client端口Create调用次数。
Create Avg Time	hdfs_namenode_rpc_client_detailed_activity_CreateAvgTime	NameNode RPC Client端口Create调用平均时间。
Mkdirs Num Ops	hdfs_namenode_rpc_client_detailed_activity_MkdirsNumOps	NameNode RPC Client端口Mkdirs调用次数。

参数	指标	描述
Mkdirs Avg Time	hdfs_namenode_rpc_client_detailed_activity_MkdirsAvgTime	NameNode RPC Client端口Mkdirs调用平均时间。
AddBlock Num Ops	hdfs_namenode_rpc_client_detailed_activity_AddBlockNumOps	NameNode RPC Client端口AddBlock调用次数。
AddBlock Avg Time	hdfs_namenode_rpc_client_detailed_activity_AddBlockAvgTime	NameNode RPC Client端口AddBlock调用平均时间。
Rename Num Ops	hdfs_namenode_rpc_client_detailed_activity_RenameNumOps	NameNode RPC Client端口Rename调用次数。
Rename Avg Time	hdfs_namenode_rpc_client_detailed_activity_RenameAvgTime	NameNode RPC Client端口Rename调用平均时间。
Fsync Num Ops	hdfs_namenode_rpc_client_detailed_activity_FsyncNumOps	NameNode RPC Client端口Fsync调用次数。
Fsync Avg Time	hdfs_namenode_rpc_client_detailed_activity_FsyncAvgTime	NameNode RPC Client端口Fsync调用平均时间。
Rename2 Num Ops	hdfs_namenode_rpc_client_detailed_activity_Rename2NumOps	NameNode RPC Client端口Rename2调用次数。
Rename2 Avg Time	hdfs_namenode_rpc_client_detailed_activity_Rename2AvgTime	NameNode RPC Client端口Rename2调用平均时间。
Delete Num Ops	hdfs_namenode_rpc_client_detailed_activity_DeleteNumOps	NameNode RPC Client端口Delete调用次数。
Delete Avg Time	hdfs_namenode_rpc_client_detailed_activity_DeleteAvgTime	NameNode RPC Client端口Delete调用平均时间。
GetListing Num Ops	hdfs_namenode_rpc_client_detailed_activity_GetListingNumOps	NameNode RPC Client端口GetListing调用次数。
GetListing Avg Time	hdfs_namenode_rpc_client_detailed_activity_GetListingAvgTime	NameNode RPC Client端口GetListing调用平均时间。
BlockReceivedAndDeleted Num Ops	hdfs_namenode_rpc_service_detailed_activity_BlockReceivedAndDeletedNumOps	NameNode Server端口块增量汇报次数。
BlockReceivedAndDeleted Avg Time	hdfs_namenode_rpc_service_detailed_activity_BlockReceivedAndDeletedAvgTime	NameNode Server端口块增量汇报平均时间。
SendHeartbeat Num Ops	hdfs_namenode_rpc_service_detailed_activity_SendHeartbeatNumOps	NameNode RPC Server端口心跳汇报次数。
SendHeartbeat Avg Time	hdfs_namenode_rpc_service_detailed_activity_SendHeartbeatAvgTime	NameNode RPC Server端口心跳汇报平均时间。
Block Report Ops	hdfs_namenode_rpc_service_detailed_activity_BlockReportNumOps	NameNode RPC Server端口块汇报次数。
Block Report Avg Time	hdfs_namenode_rpc_service_detailed_activity_BlockReportAvgTime	NameNode RPC Server端口块汇报平均时间。
Syncs Num Ops	hdfs_namenode_rpc_SyncsNumOps	NameNode与Journalnode同步次数。
Syncs Avg Time	hdfs_namenode_rpc_SyncsAvgTime	NameNode与Journalnode同步平均时间。
Block Report Num Ops	hdfs_namenode_rpc_BlockReportNumOps	NameNode块汇报次数。
Block Report Avg Time	hdfs_namenode_rpc_BlockReportAvgTime	NameNode块汇报平均时间。
Transactions Num Ops	hdfs_namenode_rpc_TransactionsNumOps	NameNode写入EditLog的次数。

参数	指标	描述
Transactions Avg Time	hdfs_namenode_rpc_TransactionsAvgTime	NameNode写入EditLog的平均时间。
GetAdditionalDatanodeOps	hdfs_namenode_rpc_GetAdditionalDatanodeOps	NameNode获取新的DataNode用于写Block次数。
GetBlockLocations	hdfs_namenode_rpc_GetBlockLocations	NameNode获取块信息次数。
Files Ops	hdfs_namenode_rpc_FilesCreated	NameNode建立文件次数。
	hdfs_namenode_rpc_FilesDeleted	NameNode删除文件次数。
	hdfs_namenode_rpc_FilesAppended	NameNode追加文件次数。
	hdfs_namenode_rpc_FilesRenamed	NameNode重命名文件次数。
Expired Heartbeats	hdfs_namenode_fsnamesystem_ExpiredHeartbeats	NameNode过期心跳数量。
FsImage Load Time	hdfs_namenode_rpc_FsImageLoadTime	NameNode读取FSImage时间。
SafeModeTime	hdfs_namenode_rpc_SafeModeTime	NameNode安全模式时间。
Transactions Since Last Edit Roll	dfs_FSNamesystem_TransactionsSinceLastLogRoll	从上一次Edit log之后的Transaction。
NameNode HeapUsage of Files Blocks	NameNode_HeapUsage_Files_Blocks	NameNode文件和内存占用的堆内存预估值。

HDFS-DataNodes

参数	指标	描述
GC Time	hdfs_namenode_jvm_GcTimeMillis	NameNode GC时间。
GC Count	hdfs_namenode_jvm_GcCount	NameNode GC次数。
Memory Metrics	hdfs_datanode_jvm_MemHeapMaxM	DataNode设置的JVM最大内存大小，单位为MB。
	hdfs_datanode_jvm_MemMaxM	DataNode可用的最大内存，单位为MB。
	hdfs_datanode_jvm_MemHeapUsedM	DataNode堆使用的内存大小，单位为MB。
	hdfs_datanode_jvm_MemNonHeapUsedM	DataNode非堆使用的内存大小，单位为MB。
	hdfs_datanode_jvm_MemHeapCommittedM	DataNode堆内存commit大小，单位为MB。
Thread Metrics	hdfs_datanode_jvm_ThreadsTerminated	DataNode进程中结束的线程个数。
	hdfs_datanode_jvm_ThreadsTimedWaiting	DataNode进程中等待一定时间后运行的线程个数。
	hdfs_datanode_jvm_ThreadsWaiting	DataNode进程中等待运行的线程个数。
	hdfs_datanode_jvm_ThreadsBlocked	DataNode进程中由于资源或锁原因被组织的线程个数。
Rpc CallQueueLength	hdfs_datanode_rpc_activity_CallQueueLength	DataNode RPC队列积压长度。
RPC Network Connections	hdfs_datanode_rpc_activity_NumDroppedConnections	DataNode RPC中断的连接数。
	hdfs_datanode_rpc_activity_NumOpenConnections	DataNode RPC正在连接的连接数。
Rpc Slow Calls	hdfs_datanode_rpc_activity_RpcSlowCalls	DataNode RPC慢连接数。
	hdfs_datanode_rpc_BlocksRead	DataNode RPC读取块次数。

参数	指标	描述
Blocks Ops	hdfs_datanode_rpc_BlocksRemoved	DataNode RPC移除块次数。
	hdfs_datanode_rpc_BlocksReplicated	DataNode RPC复制块次数。
	hdfs_datanode_rpc_BlocksVerified	DataNode RPC验证块次数。
	hdfs_datanode_rpc_BlocksWritten	DataNode RPC块写入次数。
Rpc QueueTime Num Ops	hdfs_datanode_rpc_service_activity_RpcQueueTimeNumOps	DataNode RPC Server端口被调用次数。
Rpc QueueTime Avg Time	hdfs_datanode_rpc_service_activity_RpcQueueTimeAvgTime	DataNode RPC Server端口队列平均耗时。
Rpc ProcessingTime Num Ops	hdfs_datanode_rpc_service_activity_RpcProcessingTimeNumOps	DataNode RPC Server端口被调用次数。
Rpc ProcessingTime Avg Time	hdfs_datanode_rpc_service_activity_RpcProcessingTimeAvgTime	DataNode RPC Server端口平均处理时间。
WriteBlockOpNumOps	hdfs_datanode_rpc_WriteBlockOpNumOps	DataNode RPC WriteBlock调用次数。
WriteBlockOp Avg Time	hdfs_datanode_rpc_WriteBlockOpAvgTime	DataNode RPC WriteBloc调用平均时间。
ReadBlockOp Num Ops	hdfs_datanode_rpc_ReadBlockOpNumOps	DataNode RPC ReadBlock调用次数。
ReadBlockOp Avg Time	hdfs_datanode_rpc_ReadBlockOpAvgTime	DataNode RPC ReadBlock调用平均时间。
Packet Transfer Num Ops	dfs_datanode_SendDataPacketTransferNanosNumOps	RPC包传输的次数。
Packet Transfer Time	dfs_datanode_SendDataPacketTransferNanosAvgTime	RPC包传输的时间。
Writes Local Remote	hdfs_datanode_rpc_WritesFromLocalClient	DataNode RPC本地Client写入次数。
	hdfs_datanode_rpc_WritesFromRemoteClient	DataNode RPC远程Client写入次数。
Reads Local Remote	hdfs_datanode_rpc_ReadsFromLocalClient	DataNode RPC本地Client读取次数。
	hdfs_datanode_rpc_ReadsFromRemoteClient	DataNode RPC远程Client读取次数。
Data Packet Blocked Num Ops	dfs_datanode_SendDataPacketBlockedOnNetworkNanosNumOps	数据包被阻塞的次数。
Data Packet Blocked Time	dfs_datanode_SendDataPacketBlockedOnNetworkNanosAvgTime	数据包被阻塞的时间。
FlushNanosNumOps	hdfs_datanode_rpc_FlushNanosNumOps	DataNode RPC Flush调用次数。
FlushNanosAvgTime	hdfs_datanode_rpc_FlushNanosAvgTime	DataNode RPC Flush调用平均时间。
HeartbeatsNumOps	hdfs_datanode_rpc_HeartbeatsNumOps	DataNode RPC Heartbeats调用次数。
HeartbeatsAvgTime	hdfs_datanode_rpc_ReadBlockOpAvgTime	DataNode RPC ReadBloc调用平均时间。
ReplaceBlockOpNumOps	hdfs_datanode_rpc_ReplaceBlockOpNumOps	DataNode RPC ReplaceBlock调用次数。
ReplaceBlockOpAvgTime	hdfs_datanode_rpc_ReplaceBlockOpAvgTime	DataNode RPC ReplaceBlock调用平均时间。

参数	指标	描述
IncrementalBlockReportsNumOps	hdfs_datanode_rpc_IncrementalBlockReportsNumOps	DataNode RPC IncrementalBlockReports调用次数。
IncrementalBlockReportsAvgTime	hdfs_datanode_rpc_IncrementalBlockReportsAvgTime	DataNode RPC IncrementalBlockReports调用平均时间。
BlockReportsNumOps	hdfs_datanode_rpc_BlockReportsNumOps	DataNode RPC BlockReports调用次数。
BlockReportsAvgTime	hdfs_datanode_rpc_BlockReportsAvgTime	DataNode RPC BlockReports调用平均时间。
ReplaceBlockOpNumOps	hdfs_datanode_rpc_ReplaceBlockOpNumOps	DataNode RPC ReplaceBlock调用次数。
ReplaceBlockOpAvgTime	hdfs_datanode_rpc_ReplaceBlockOpAvgTime	DataNode RPC ReplaceBlock调用平均时间。
CopyBlockOpNumOps	hdfs_datanode_rpc_CopyBlockOpNumOps	DataNode RPC CopyBlock调用次数。
CopyBlockOpAvgTime	hdfs_datanode_rpc_CopyBlockOpAvgTime	DataNode RPC CopyBlock调用平均时间。
ReadBlockOpNumOps	hdfs_datanode_rpc_ReadBlockOpNumOps	DataNode RPC ReadBloc调用次数。
ReadBlockOpAvgTime	hdfs_datanode_rpc_ReadBlockOpAvgTime	DataNode RPC ReadBloc调用平均时间。
Network Errors	dfs_datanode_DatanodeNetworkErrors	网络错误。
PacketAckRoundTripTimeNanosAvgTime	hdfs_datanode_rpc_PacketAckRoundTripTimeNanosAvgTime	DataNode RPC副本写入ACK时间。

HDFS-JournanlNodes

参数	指标	描述
Memory Metrics	hdfs_journalnode_jvm_MemHeapMaxM	JournalNode设置的JVM最大内存大小，单位为MB。
	hdfs_journalnode_jvm_MemMaxM	JournalNode可用的最大内存，单位为MB。
	hdfs_journalnode_jvm_MemHeapUsedM	JournalNode堆使用的内存大小，单位为MB。
	hdfs_journalnode_jvm_MemNonHeapUsedM	JournalNode非堆使用的内存大小，单位为MB。
Threads Metrics	hdfs_journalnode_jvm_ThreadsTerminated	JournalNode进程中结束的线程个数。
	hdfs_journalnode_jvm_ThreadsTimedWaiting	JournalNode进程中等待一定时间后运行的线程个数。
	hdfs_journalnode_jvm_ThreadsWaiting	JournalNode进程中等待运行的线程个数。
	hdfs_journalnode_jvm_ThreadsBlocked	JournalNode进程中由于资源或锁原因被组织的线程个数。
GC Count	hdfs_journalnode_jvm_GcCount	JournalNode GC次数。
GC Time	hdfs_journalnode_jvm_GcTimeMillis	JournalNode GC时间。
RPC QueueTime Num Ops	hdfs_journalnode_rpc_activity_RpcQueueTimeNumOps	JournalNode RPC被调用次数。
RPC QueueTime Avg Time	hdfs_journalnode_rpc_activity_RpcQueueTimeAvgTime	JournalNode RPC队列平均耗时。
RPC ProcessingTime Num Ops	hdfs_journalnode_rpc_activity_RpcProcessingTimeNumOps	JournalNode RPC被调用次数。

参数	指标	描述
RPC ProcessingTime Avg Time	hdfs_journalnode_rpc_activity_RpcProcessingTimeAvgTime	JournalNode RPC平均处理时间。
RPC CallQueueLength	hdfs_datanode_rpc_activity_CallQueueLength	DataNode RPC队列积压长度。
RPC Connections	hdfs_journalnode_rpc_activity_NumDroppedConnections	JournalNode RPC中断的连接数。
	hdfs_journalnode_rpc_activity_NumOpenConnections	JournalNode RPC正在连接的连接数。
FinalizeLogSegment Num Ops	hdfs_journalnode_rpc_detailed_activity_FinalizeLogSegmentNumOps	JournalNode RPC FinalizeLogSegment调用的次数。
FinalizeLogSegment Avg Time	hdfs_journalnode_rpc_detailed_activity_FinalizeLogSegmentAvgTime	JournalNode FinalizeLogSegment调用的平均时间。
Journal Num Ops	hdfs_journalnode_rpc_detailed_activity_JournalNumOps	JournalNode RPC Journal调用的次数。
Journal Avg Time	hdfs_journalnode_rpc_detailed_activity_JournalAvgTime	JournalNode RPC Journal调用的平均时间。
StartLogSegment Num Ops	hdfs_journalnode_rpc_detailed_activity_StartLogSegmentNumOps	JournalNode RPC StartLogSegment调用的次数。
StartLogSegment Avg Time	hdfs_journalnode_rpc_detailed_activity_StartLogSegmentAvgTime	JournalNode RPC StartLogSegment调用的平均时间。
New Epoch Num Ops	hdfs_journalnode_rpc_detailed_activity_NewEpochNumOps	JournalNode RPC NewEpoch调用的次数。
NewEpoch Avg Time	hdfs_journalnode_rpc_detailed_activity_NewEpochAvgTime	JournalNode RPC NewEpoch调用的平均时间。
RPC Slow Calls	hdfs_journalnode_rpc_activity_RpcSlowCalls	JournalNode RPC慢连接数。

4.2.2.4. YARN指标

本文为您介绍YARN指标的详细信息。

YARN指标包含以下部分：

- YARN-HOME
- YARN-Queues
- YARN-ResourceManager
- YARN-NodeManagers
- YARN-TimeLineServer
- YARN-JobHistory

YARN-HOME

指标	描述
yarn_cluster_status	集群状态： • 0：Standby • 1：Active
yarn_cluster_availableMB	集群可用内存大小，单位为MB。
yarn_cluster_appsSubmitted	集群已经提交的任务数。
yarn_cluster_appsCompleted	集群已经提交的任务数。
yarn_cluster_appsPending	集群正在阻塞的任务数。

指标	描述
yarn_cluster_appsRunning	集群正在运行的任务数。
yarn_cluster_appsFailed	集群运行失败的任务数。
yarn_cluster_appsKilled	集群运行中止的任务数。
yarn_cluster_reservedMB	集群被预留调度的内存大小，单位为MB。
yarn_cluster_availableMB	集群可用内存大小，单位为MB。
yarn_cluster_allocatedMB	集群已分配的内存大小，单位为MB。
yarn_cluster_totalMB	集群总内存大小，单位为MB。
yarn_cluster_reservedVirtualCores	集群被预留调度的虚拟核数。
yarn_cluster_availableVirtualCores	集群可用虚拟核数。
yarn_cluster_allocatedVirtualCores	集群已分配虚拟核数。
yarn_cluster_totalVirtualCores	集群总虚拟核数。
yarn_cluster_containersAllocated	集群已经分配的Container个数。
yarn_cluster_containersReserved	集群预留调度的Container个数。
yarn_cluster_containersPending	集群阻塞调度的Container个数。
yarn_cluster_totalNodes	集群总节点数。
yarn_cluster_activeNodes	集群存活的节点数。
yarn_cluster_lostNodes	集群丢失的节点数。
yarn_cluster_unhealthyNodes	集群不健康的节点数。
yarn_cluster_decommissioningNodes	集群正在下线的节点数。
yarn_cluster_decommissionedNodes	集群已经下线的节点数。
yarn_cluster_rebootedNodes	集群重启的节点数。
yarn_cluster_shutdownNodes	集群关闭的节点数。

YARN-Queues

当您使用节点标签功能时，以下指标仅统计特定队列在default分区的对应值。

指标	描述
yarn_resourcemanager_queue_AppsSubmitted	ResourceManager调度器特定队列提交的任务数。
yarn_resourcemanager_queue_AppsRunning	ResourceManager调度器特定队列提正在运行的任务数。
yarn_resourcemanager_queue_AppsPending	ResourceManager调度器特定队列阻塞的任务数。
yarn_resourcemanager_queue_AppsCompleted	ResourceManager调度器特定队列完成的任务数。
yarn_resourcemanager_queue_AppsKilled	ResourceManager调度器特定队列完成的任务数。
yarn_resourcemanager_queue_AppsFailed	ResourceManager调度器特定队列终止的任务数。
yarn_resourcemanager_queue_AllocatedMB	ResourceManager调度器特定队列分配的内存大小，单位为MB。
yarn_resourcemanager_queue_AllocatedVCores	ResourceManager调度器特定队列分配的虚拟核数。
yarn_resourcemanager_queue_AllocatedContainers	ResourceManager调度器特定队列分配的container数。
yarn_resourcemanager_queue_AggregateContainersAllocated	ResourceManager调度器特定队列累积的container分配数。

指标	描述
yarn_resourcemanager_queue_AggregateContainersReleased	ResourceManager调度器特定队列累积的container释放数。
yarn_resourcemanager_queue_AvailableMB	ResourceManager调度器特定队可用内存，单位为MB。
yarn_resourcemanager_queue_AvailableVCores	ResourceManager调度器特定队列可用核数。
yarn_resourcemanager_queue_PendingMB	ResourceManager调度器特定队列阻塞调度内存，单位为MB。
yarn_resourcemanager_queue_PendingVCores	ResourceManager调度器特定队列阻塞调度核数。
yarn_resourcemanager_queue_PendingContainers	ResourceManager调度器特定队列阻塞调度container个数。
yarn_resourcemanager_queue_ReservedMB	ResourceManager调度器特定队列预留内存，单位为MB。
yarn_resourcemanager_queue_ReservedVCores	ResourceManager调度器特定队列预留核数。
yarn_resourcemanager_queue_ReservedContainers	ResourceManager调度器特定队列预留container数。

YARN-ResourceManager

指标	描述
yarn_resourcemanager_rpc_RpcProcessingTimeAvgTime	ResourceManager RPC队列平均处理时间。单位为ms（毫秒）。
yarn_resourcemanager_rpc_CallQueueLength	ResourceManager RPC队列积压长度。
yarn_resourcemanager_jvm_MemNonHeapCommittedM	ResourceManager JVM当前非堆内存大小已提交大小，单位为MB。
yarn_resourcemanager_jvm_MemNonHeapUsedM	ResourceManager JVM当前已使用的非堆内存大小，单位为MB。
yarn_resourcemanager_jvm_MemHeapUsedM	ResourceManager JVM当前已使用堆内存大小，单位为MB。
yarn_resourcemanager_jvm_MemNonHeapMaxM	ResourceManager JVM非堆最大可用内存，单位为MB。
yarn_resourcemanager_jvm_MemHeapMaxM	ResourceManager JVM堆内存最大可用内存，单位为MB。
yarn_resourcemanager_jvm_MemHeapCommittedM	ResourceManager JVM当前堆内存已提交的内存，单位为MB。
yarn_resourcemanager_jvm_MemMaxM	ResourceManager JVM配置的最大内存，单位为MB。
yarn_resourcemanager_jvm_GcTimeMillis	ResourceManager JVM GC时间。
yarn_resourcemanager_jvm_GcCount	ResourceManager JVM GC次数。

YARN-NodeManagers

指标	描述
yarn_nodemanager_AvailableGB	NodeManager可用的内存大小，单位为GB。
yarn_nodemanager_AllocatedGB	NodeManager使用的内存大小，单位为GB。
yarn_nodemanager_AllocatedVCores	NodeManager使用的虚拟核数。
yarn_nodemanager_AvailableVCores	NodeManager可用的虚拟核数。
yarn_nodemanager_ContainersLaunched	NodeManager Container启动的个数。
yarn_nodemanager_ContainersRunning	NodeManager Container正在运行的个数。
yarn_nodemanager_ContainersFailed	NodeManager Container失败的个数。
yarn_nodemanager_ContainersCompleted	NodeManager Container完成的个数。
yarn_nodemanager_ContainersIniting	NodeManager Container初始化的个数。
yarn_nodemanager_ContainersKilled	NodeManager Container被中止的个数。

指标	描述
yarn_nodemanager_BadLocalDirs	NodeManager磁盘损坏个数。
yarn_nodemanager_jvm_MemNonHeapUsedM	NodeManager JVM已使用的非堆内存大小，单位为MB。
yarn_nodemanager_GoodLocalDirsDiskUtilizationPerc	NodeManager磁盘利用率。
yarn_nodemanager_jvm_MemNonHeapMaxM	NodeManager JVM非堆最大可用内存，单位为MB。
yarn_nodemanager_jvm_MemNonHeapCommittedM	NodeManager JVM非堆内存已提交大小，单位为MB。
yarn_nodemanager_jvm_MemHeapCommittedM	NodeManager JVM堆内存已提交大小，单位为MB。
yarn_nodemanager_jvm_MemHeapUsedM	NodeManager JVM当前已使用堆内存大小，单位为MB。
yarn_nodemanager_jvm_MemMaxM	NodeManager JVM配置的最大内存大小，单位为MB。
yarn_nodemanager_jvm_MemHeapMaxM	NodeManager JVM堆内存最大可用内存，单位为MB。
yarn_nodemanager_jvm_GcTimeMillis	NodeManager JVM GC时间。
yarn_nodemanager_jvm_GcCount	NodeManager JVM GC次数。
yarn_nodemanager_shuffle_ShuffleOutputsFailed	NodeManager Shuffle输出失败的个数。
yarn_nodemanager_shuffle_ShuffleOutputBytes	NodeManager Shuffle输出的字节数。
yarn_nodemanager_shuffle_ShuffleConnections	NodeManager Shuffle连接数。
yarn_nodemanager_shuffle_ShuffleOutputsOK	NodeManager Shuffle输出成功的个数。

YARN-TimeLineServer

指标	描述
yarn_timelineserver_jvm_MemNonHeapUsedM	TimelineServer JVM当前已使用的非堆内存大小，单位为MB。
yarn_timelineserver_jvm_MemNonHeapCommittedM	TimelineServer JVM当前非堆内存大小已提交大小，单位为MB。
yarn_timelineserver_jvm_MemNonHeapMaxM	TimelineServer JVM配置的最大非堆内存大小，单位为MB。
yarn_timelineserver_jvm_MemHeapUsedM	TimelineServer JVM当前已使用的堆内存大小，单位为MB。
yarn_timelineserver_jvm_MemHeapCommittedM	TimelineServer JVM当前堆内存已提交的内存，单位为MB。
yarn_timelineserver_jvm_MemHeapMaxM	TimelineServer JVM堆内存最大可用内存，单位为MB。
yarn_timelineserver_jvm_MemMaxM	TimelineServer JVM配置的最大内存，单位为MB。
yarn_timelineserver_jvm_GcCount	TimelineServer JVM GC次数。
yarn_timelineserver_jvm_GcTimeMillis	TimelineServer JVM GC时间。
yarn_timeline_GetEntitiesOps	TimelineServer获取批量entities操作数。
yarn_timeline_GetEntitiesTimeAvgTime	TimelineServer获取批量entities平均时间。
yarn_timeline_GetEntityOps	TimelineServer获取entity操作数。
yarn_timeline_GetEntityTimeAvgTime	TimelineServer获取entity平均时间。
yarn_timeline_GetEventsOps	TimelineServer获取批量events操作数。
yarn_timeline_GetEventsTimeAvgTime	TimelineServer获取批量events平均时间。
yarn_timeline_PostEntitiesOps	TimelineServer更新批量entities操作数。
yarn_timeline_PostEntitiesTimeAvgTime	TimelineServer更新批量entities平均时间。
yarn_timeline_PutDomainOps	TimelineServer更新Domain操作数。

指标	描述
yarn_timeline_PutDomainTimeAvgTime	TimelineServer更新Domain平均时间。
yarn_timeline_GetDomainOps	TimelineServer获取Domain操作数。
yarn_timeline_GetDomainTimeAvgTime	TimelineServer获取Domain平均时间。
yarn_timeline_GetDomainsOps	TimelineServer获取批量Domain操作数。
yarn_timeline_GetDomainsTimeAvgTime	TimelineServer获取批量Domain平均时间。

YARN-JobHistory

指标	描述
yarn_jobhistory_jvm_MemNonHeapUsedM	JobHistory JVM当前已使用的非堆内存大小，单位为MB。
yarn_jobhistory_jvm_MemNonHeapCommittedM	JobHistory JVM当前非堆内存大小已提交大小，单位为MB。
yarn_jobhistory_jvm_MemNonHeapMaxM	JobHistory JVM配置的最大非堆内存大小，单位为MB。
yarn_jobhistory_jvm_MemHeapUsedM	JobHistory JVM当前已使用的堆内存大小，单位为MB。
yarn_jobhistory_jvm_MemHeapCommittedM	JobHistory JVM当前堆内存已提交的内存，单位为MB。
yarn_jobhistory_jvm_MemHeapMaxM	JobHistory JVM堆内存最大可用内存，单位为MB。
yarn_jobhistory_jvm_MemMaxM	JobHistory JVM配置的最大内存，单位为MB。

4.2.2.5. Hive指标

本文为您介绍Hive指标的详细信息。

Hive指标包含以下部分：

- HIVE-HiveMetaStore
- HIVE-HiveServer2

HIVE-HiveMetaStore

参数	指标	描述
memory_heap_max	hive_memory_heap_max	JVM最大可用堆内存，单位：Byte。
memory_heap_used	hive_memory_heap_used	JVM已使用堆内存，单位：Byte。
memory_non_heap_used	hive_memory_non_heap_used	JVM已使用堆外内存量，单位：Byte。
active_calls_api_alter_table	hive_active_calls_api_alter_table	当前活跃的alter table请求数。
active_calls_api_create_table	hive_active_calls_api_create_table	当前活跃的create table请求数。
active_calls_api_drop_table	hive_active_calls_api_drop_table	当前活跃的drop table请求数。
api_alter_table	hive_api_alter_table	alter table请求平均时间，单位：ms。
api_alter_table_with_environment_context	hive_api_alter_table_with_environment_context	alter table with env context请求平均时间，单位：ms。
api_create_table	hive_api_create_table	create table请求平均时间，单位：ms。
api_create_table_with_environment_context	hive_api_create_table_with_environment_context	create table with env context请求平均时间，单位：ms。
api_drop_table	api_drop_table	drop table请求平均时间，单位：ms。
api_drop_table_with_environment_context	hive_api_drop_table_with_environment_context	drop table with env context请求平均时间，单位：ms。
api_get_all_databases	hive_api_get_all_databases	get all databases请求平均时间，单位：ms。

参数	指标	描述
api_get_all_functions	hive_api_get_all_functions	get all functions请求平均时间，单位：ms。
api_get_database	hive_api_get_database	get database请求平均时间，单位：ms。
api_get_databases	hive_api_get_databases	get databases请求平均时间，单位：ms。
api_get_multi_table	hive_api_get_multi_table	get multi table请求平均时间，单位：ms。
api_get_table	hive_api_get_tables_by_type	get table请求平均时间，单位：ms。
api_get_table_objects_by_name_req	hive_api_get_table_objects_by_name_req	get table objects by name请求平均时间，单位：ms。
api_get_table_req	hive_api_get_table_req	get table req请求平均时间，单位：ms。
api_get_table_statistics_req	hive_api_get_table_statistics_req	get table statistics请求平均时间，单位：ms。
api_get_tables	hive_api_get_tables	get tables请求平均时间，单位：ms。
api_get_tables_by_type	hive_api_get_tables_by_type	get tables by type请求平均时间，单位：ms。

HIVE-HiveServer2

参数	指标	描述
hs2_active_sessions	hive_metrics_hs2_active_sessions	当前活跃的session个数。
memory_total_init	hive_metrics_memory_total_init	JVM初始化总内存，单位：Byte。
memory_total_committed	hive_metrics_memory_total_committed	JVM已预留总内存，单位：Byte。
memory_total_max	hive_metrics_memory_total_max	JVM最大可用总内存，单位：Byte。
memory_heap_committed	hive_metrics_memory_heap_committed	JVM已预留堆内存，单位：Byte。
memory_heap_init	hive_metrics_memory_heap_inithive_metrics_memory_heap_committed	JVM初始化堆内存，单位：Byte。
memory_non_heap_committed	hive_metrics_memory_non_heap_committed	JVM已预留堆外内存，单位：Byte。
memory_non_heap_init	hive_metrics_memory_non_heap_init	JVM初始化堆外内存，单位：Byte。
memory_non_heap_max	hive_metrics_memory_non_heap_max	JVM最大可用堆外内存，单位：Byte。
gc_PS_MarkSweep_count	hive_metrics_gc_PS_MarkSweep_count	JVM PS MarkSweep GC次数。
gc_PS_MarkSweep_time	hive_metrics_gc_PS_MarkSweep_time	JVM PS MarkSweep GC时间，单位：ms。
gc_PS_Scavenge_time	hive_metrics_gc_PS_Scavenge_time	JVM PS Scavenge GC时间，单位：ms。
threads_daemon_count	hive_metrics_threads_daemon_count	JVM daemon线程数。
threads_count	hive_metrics_threads_count	JVM线程数。
threads_blocked_count	hive_metrics_threads_blocked_count	JVM blocked线程数。
threads_deadlock_count	hive_metrics_threads_deadlock_count	JVM deadlock线程数。
threads_new_count	hive_metrics_threads_new_count	JVM new状态线程数。
threads_runnable_count	hive_metrics_threads_runnable_count	JVM runnable线程数。
threads_terminated_count	hive_metrics_threads_terminated_count	JVM terminated线程数。
threads_waiting_count	hive_metrics_threads_waiting_count	JVM waiting线程数。
threads_timed_waiting_count	hive_metrics_threads_timed_waiting_count	JVM timed_waiting线程数。
memory_heap_max	hive_metrics_memory_heap_max	JVM最大可用堆内存，单位：Byte。

参数	指标	描述
memory_heap_used	hive_metrics_memory_heap_used	JVM已使用堆内存，单位：Byte。
memory_non_heap_used	hive_metrics_memory_non_heap_used	JVM已使用堆外内存量，单位：Byte。
hs2_open_sessions	hive_metrics_hs2_open_sessions	当前打开的session数。
hive_mapred_tasks	hive_metrics_hive_mapred_tasks	提交的Hive on MR作业总数。
hive_tez_tasks	hive_metrics_hive_tez_tasks	提交的Hive on Tez作业总数。
cumulative_connection_count	hive_metrics_cumulative_connection_count	累计连接数。
active_calls_api_runTasks	hive_metrics_active_calls_api_runTasks	当前runtask请求数。
hs2_completed_sql_operation_FINISHED	hive_metrics_hs2_completed_sql_operation_FINISHED	已结束的SQL总数。
hs2_sql_operation_active_user	hive_metrics_hs2_sql_operation_active_user	当前活跃用户数。
open_connections	hive_metrics_open_connections	当前打开的连接数。
api_PostHook_com_aliyun_emr_meta_hive_hook_LineageLoggerHook	hive_metrics_api_PostHook_com_aliyun_emr_meta_hive_hook_LineageLoggerHook	执行LineageLoggerHook的平均时间，单位：ms。
api_hs2_sql_operation_PENDING	hive_metrics_api_hs2_sql_operation_PENDING	SQL任务处于PEEDING状态的平均时间，单位：ms。
api_hs2_sql_operation_RUNNING	hive_metrics_api_hs2_sql_operation_RUNNING	运SQL任务处于RUNNING状态的平均时间，单位：ms。
hs2_submitted_queries	hive_metrics_hs2_submitted_queries	提交查询的平均时间，单位：ms。
hs2_executing_queries	hive_metrics_hs2_executing_queries	执行查询的平均时间，单位：ms。
hs2_succeeded_queries	hive_metrics_hs2_succeeded_queries	服务启动后成功的查询数。
hs2_failed_queries	hive_metrics_hs2_failed_queries	服务启动后失败的查询数。

4.2.2.6. ZooKeeper指标

本文为您介绍ZooKeeper指标的详细信息。

指标	描述
zk_packets_received	ZooKeeper接收的包的数量。
zk_packets_sent	ZooKeeper发送的包的数量。
zk_avg_latency	ZooKeeper平均请求延迟，单位：ms。
zk_min_latency	ZooKeeper最小请求延迟，单位：ms。
zk_max_latency	ZooKeeper最大请求延迟，单位：ms。
zk_watch_count	ZooKeeper watch的数量。
zk_znode_count	ZooKeeper znode的数量。
zk_num_alive_connections	ZooKeeper存活的连接数。
zk_outstanding_requests	ZooKeeper排队请求的数量。当ZooKeeper超过了它的处理能力时，该值会增大。
zk_approximate_data_size	ZooKeeper的数据大小（近似值），单位：Byte。
zk_open_file_descriptor_count	ZooKeeper打开文件的数量。
zk_max_file_descriptor_count	ZooKeeper最大允许打开的文件数量。

指标	描述
zk_node_status	ZooKeeper节点状态： • -1：节点不可用。 • 0：作为follower节点。 • 1：作为leader节点。
zk_synced_followers	同步的ZooKeeper服务数量。

4.2.2.7. Kafka指标

本文为您介绍Kafka指标的详细信息。

Kafka指标包含以下部分：

- Kafka-HOME
- Kafka-Broker
- Kafka-Topic

Kafka-HOME

参数	指标	描述
ActiveControllerCount	Kafka_Broker_kafka_controller_KafkaController_ActiveControllerCount	当某个Broker上该指标为1时，该Broker为Controller节点。
GlobalTopicCount	Kafka_Broker_kafka_controller_KafkaController_GlobalTopicCount	集群总Topic个数。
GlobalPartitionCount	Kafka_Broker_kafka_controller_KafkaController_GlobalPartitionCount	集群总Partition个数。
UnderMinIsrPartitionCount	Kafka_Broker_kafka_server_ReplicaManager_UnderMinIsrPartitionCount	低于Min Isr partition的个数。
OfflineLogDirectoryCount	Kafka_Broker_kafka_log_LogManager_OfflineLogDirectoryCount	Directory Offline个数。
OfflinePartitionsCount	Kafka_Broker_kafka_controller_KafkaController_OfflinePartitionsCount	Offline的Partition个数。
UnderReplicatedPartitions	Kafka_Broker_kafka_server_ReplicaManager_UnderReplicatedPartitions	处于未同步状态Partition个数。
BytesOutPerSec	Kafka_Broker_kafka_server_BrokerTopicMetrics_BytesOutPerSec_OneMinuteRate	每秒流量出口字节数。
BytesInPerSec	Kafka_Broker_kafka_server_BrokerTopicMetrics_BytesInPerSec_OneMinuteRate	每秒流量入口字节数。
MessagesInPerSec	Kafka_Broker_kafka_server_BrokerTopicMetrics_MessagesInPerSec_OneMinuteRate	每秒流量入口条数。

Kafka-Broker

Status

参数	指标	描述
LeaderCount	Kafka_Broker_kafka_server_ReplicaManager_LeaderCount	Leader个数。

参数	指标	描述
PartitionCount	Kafka_Broker_kafka_server_ReplicaManager_PartitionCount	Partition个数。
OfflineLogDirectoryCount	Kafka_Broker_kafka_log_LogManager_OfflineLogDirectoryCount	Directory Offline个数。
OfflineReplicaCount	Kafka_Broker_kafka_server_ReplicaManager_OfflineReplicaCount	Offline Replica个数。
UnderReplicatedPartitions	Kafka_Broker_kafka_server_ReplicaManager_UnderReplicatedPartitions	处于未同步状态的Partition个数。
UnderMinIsrPartitionCount	Kafka_Broker_kafka_server_ReplicaManager_UnderMinIsrPartitionCount	低于Min Isr partition的个数。

Throughput

参数	指标	描述
BytesInPerSec	Kafka_Broker_kafka_server_BrokerTopicMetrics_BytesInPerSec_OneMinuteRate	Broker每秒流量入口字节数。
BytesOutPerSec	Kafka_Broker_kafka_server_BrokerTopicMetrics_BytesOutPerSec_OneMinuteRate	Broker每秒流量出口字节数。
MessagesInPerSec	Kafka_Broker_kafka_server_BrokerTopicMetrics_MessagesInPerSec_OneMinuteRate	Broker每秒流量入口条数。

Performance

参数	指标	描述
IsrShrinksPerSec	Kafka_Broker_kafka_server_ReplicaManager_IsrShrinksPerSec_OneMinuteRate	ISR缩减频率。
IsrExpandsPerSec	Kafka_Broker_kafka_server_ReplicaManager_IsrExpandsPerSec_OneMinuteRate	ISR膨胀频率。
RequestHandlerAvgIdlePercent	Kafka_Broker_kafka_server_KafkaRequestHandlerPool_RequestHandlerAvgIdlePercent_OneMinuteRate	Request请求处理线程的空闲比。
NetworkProcessorAvgIdlePercent	Kafka_Broker_kafka_network_SocketServer_NetworkProcessorAvgIdlePercent	网络处理线程的空闲比。
RequestQueueUsagePercent	Kafka_Broker_kafka_network_RequestChannel_RequestQueueUsagePercent	RequestQueue队列的使用率。
LogFlushRateAndTimeMs	Kafka_Broker_kafka_log_LogFlushStats_LogFlushRateAndTimeMs_OneMinuteRate	磁盘刷盘的速率。
	Kafka_Broker_kafka_log_LogFlushStats_LogFlushRateAndTimeMs_99thPercentile	磁盘刷盘消耗的时间，单位：ms。

Storage

参数	指标	描述
Size	Kafka_Broker_kafka_log_Log_Size	Partition存储的大小。
disk-usages-max-percent	Kafka_Broker_kafka_server_ReplicaManager_disk_usages_max_percent	该Broker各个磁盘使用率的最大值。
disk-usages-mean-percent	Kafka_Broker_kafka_server_ReplicaManager_disk_usages_mean_percent	该Broker各个磁盘使用率的平均值。
disk-usages-min-percent	Kafka_Broker_kafka_server_ReplicaManager_disk_usages_min_percent	该Broker各个磁盘使用率的最小值。

Request Rate

参数	指标	描述
TotalFetchRequestsPerSec	Kafka_Broker_kafka_server_BrokerTopicMetrics_TotalFetchRequestsPerSec_OneMinuteRate	Broker每秒Fetch请求次数。
TotalProduceRequestsPerSec	Kafka_Broker_kafka_server_BrokerTopicMetrics_TotalProduceRequestsPerSec_OneMinuteRate	Broker每秒Produce请求次数。
FailedFetchRequestsPerSec	Kafka_Broker_kafka_server_BrokerTopicMetrics_FailedFetchRequestsPerSec_OneMinuteRate	Broker每秒失败的Fetch请求次数。
FailedProduceRequestsPerSec	Kafka_Broker_kafka_server_BrokerTopicMetrics_FailedProduceRequestsPerSec_OneMinuteRate	Broker每秒失败的Produce请求次数。
RequestsPerSec-Produce	Kafka_Broker_kafka_network_RequestMetrics_RequestsPerSec_OneMinuteRate	Produce每秒请求的字节数。
RequestsPerSec-Fetch	Kafka_Broker_kafka_network_RequestMetrics_RequestsPerSec_OneMinuteRate	Fetch请求数。
RequestsPerSec-FetchFollower	Kafka_Broker_kafka_network_RequestMetrics_RequestsPerSec_OneMinuteRate	Follower Fetch请求数。
RequestsPerSec-FetchConsumer	Kafka_Broker_kafka_network_RequestMetrics_RequestsPerSec_OneMinuteRate	Consumer Fetch请求数。

Request Time

参数	指标	描述
RequestMetrics-TotalTimeMs-Produce-99th	Kafka_Broker_kafka_network_RequestMetrics_TotalTimeMs_99thPercentile	Produce Request处理时长，单位：ms。
RequestMetrics-TotalTimeMs-Fetch-99th	Kafka_Broker_kafka_network_RequestMetrics_TotalTimeMs_99thPercentile	Fetch Request处理时长，单位：ms。

MessageConversion

参数	指标	描述
FetchMessageConversionsPerSec	Kafka_Broker_kafka_server_BrokerTopicMetrics_FetchMessageConversionsPerSec_OneMinuteRate	Broker每秒Fetch Request导致的消息版本转换的条数。

参数	指标	描述
ProduceMessageConversionsPerSec	Kafka_Broker_kafka_server_BrokerTopicMetrics_ProduceMessageConversionsPerSec_OneMinuteRate	Broker每秒Produce Request导致的消息版本转换的条数。

ZK session

参数	指标	描述
ZooKeeperDisconnectsPerSec	Kafka_Broker_kafka_server_SessionExpireListener_ZooKeeperDisconnectsPerSec_OneMinuteRate	Zookeeper客户端断开频率。
ZooKeeperExpiresPerSec	Kafka_Broker_kafka_server_SessionExpireListener_ZooKeeperExpiresPerSec_OneMinuteRate	Zookeeper客户端Session过期频率。

JVM

参数	指标	描述
G1-Old-CollectionTime	Kafka_Broker_java_lang_GarbageCollector_G1_Old_Generation_CollectionTime	老年代垃圾回收器收集时长。
G1-Old-CollectionCount	Kafka_Broker_java_lang_GarbageCollector_G1_Old_Generation_CollectionCount	老年代垃圾回收器收集次数。
G1-Young-CollectionTime	Kafka_Broker_java_lang_GarbageCollector_G1_Young_Generation_CollectionTime	年轻代垃圾回收器收集时长。
G1-Young-CollectionCount	Kafka_Broker_java_lang_GarbageCollector_G1_Young_Generation_CollectionCount	年轻代垃圾回收器收集次数。

Kafka-Topic

Status

参数	指标	描述
UnderReplicated	Kafka_Broker_kafka_cluster_Partition_UnderReplicated	该Topic-Partition是否在非同步状态。
UnderMinIsr	Kafka_Broker_kafka_cluster_Partition_UnderMinIsr	该Topic-Partition是否处于小于MinIsr状态。

Throughput

参数	指标	描述
BytesInPerSec	kafka_server_BrokerTopicMetrics_BytesInPerSec_OneMinuteRate	Topic每秒流量入口字节数。
BytesOutPerSec	kafka_server_BrokerTopicMetrics_BytesOutPerSec_OneMinuteRate	Topic每秒流量出口字节数。
MessagesInPerSec	kafka_server_BrokerTopicMetrics_MessagesInPerSec_OneMinuteRate	Topic每秒流量入口条数。

Request Rate

参数	指标	描述

参数	指标	描述
TotalFetchRequestsPerSec	Kafka_Broker_kafka_server_BrokerTopicMetrics_TotalFetchRequestsPerSec_OneMinuteRate	Topic每秒Fetch请求次数。
TotalProduceRequestsPerSec	Kafka_Broker_kafka_server_BrokerTopicMetrics_TotalProduceRequestsPerSec_OneMinuteRate	Topic每秒Produce请求次数。
FailedFetchRequestsPerSec	Kafka_Broker_kafka_server_BrokerTopicMetrics_FailedFetchRequestsPerSec_OneMinuteRate	Topic每秒失败Fetch请求次数。
FailedProduceRequestsPerSec	Kafka_Broker_kafka_server_BrokerTopicMetrics_FailedProduceRequestsPerSec_OneMinuteRate	Topic每秒失败Produce请求次数。

MessageConversion

参数	指标	描述
FetchMessageConversionsPerSec	Kafka_Broker_kafka_server_BrokerTopicMetrics_FetchMessageConversionsPerSec_OneMinuteRate	Topic每秒Fetch Request导致的消息版本转换的条数。
ProduceMessageConversionsPerSec	Kafka_Broker_kafka_server_BrokerTopicMetrics_ProduceMessageConversionsPerSec_OneMinuteRate	Topic每秒Produce Request导致的消息版本转换的条数。

Storage

参数	指标	描述
Size	Kafka_Broker_kafka_log_Log_Size	Partition存储的大小。

4.2.2.8. Impala指标

本文为您介绍Impala指标的详细信息。

指标	描述
impala_impala_server_resultset_cache_total_bytes	结果集缓存大小，单位：Byte。
impala_num_executing_queries	当前正在执行的查询数量。
impala_num_waiting_queries	当前正在等待的查询数量。
impala_impala_server_query_durations_ms_95th	95%的查询耗时时间，单位：ms。
impala_num_in_flight_queries	集群正在in fight状态的查询数量。
impala_impala_server_query_durations_ms_75th	75%的查询耗时时间，单位：ms。
impala_impala_thrift_server_CatalogService_svc_thread_wait_time_99_9th	Catalog Service的客户端对服务线程的等待时间，单位：ms。
impala_impala_thrift_server_CatalogService_connection_setup_time_99_9th	99%的Catalog Service客户端等待建立连接所花费的时间，单位：ms。
impala_impala_server_query_durations_ms_99_9th	99%的查询耗时时间，单位：ms。
impala_impala_server_ddl_durations_ms_99_9th	99%的DDL操作耗时时间，单位：ms。

指标	描述
impala_impala_server_query_durations_ms_90th	90%的查询耗时时间，单位：ms。
impala_impala_server_ddl_durations_ms_90th	90%的DDL操作耗时时间，单位：ms。
impala_impala_server_query_durations_ms_50th	50%的查询耗时时间，单位：ms。
impala_impala_server_ddl_durations_ms_50th	50%的DDL操作耗时时间，单位：ms。
impala_impala_server_ddl_durations_ms_95th	95%的DDL操作耗时时间，单位：ms。
impala_impala_server_scan_ranges_num_missing_volume_id	在进程生命周期内缺失volume id的scan range总数。
impala_impala_server_ddl_durations_ms_75th	75%的DDL操作耗时时间，单位：ms。
impala_impala_server_num_queries_spilled	任何运算符溢出的查询数。
impala_impala_server_scan_ranges_total	在进程生命周期内读取的扫描范围总数。
impala_impala_server_num_queries_expired	由于不活动而过期的查询数。
impala_impala_server_resultset_cache_total_num_rows	结果集缓存记录数。
impala_impala_server_num_open_hiveserver2_sessions	打开的HiveServer2会话数。
impala_impala_server_num_sessions_expired	由于不活动而过期的会话数。
impala_impala_server_num_fragments_in_flight	当前正在执行的查询片段实例的数量。
impala_impala_server_num_queries_registered	在此Impala服务器实例上注册的查询总数。包括正在进行中并等待关闭的查询。
impala_impala_server_num_files_open_for_insert	当前为写入而打开的HDFS文件数。
impala_impala_server_num_queries	在进程生命周期内处理的查询总数。
impala_impala_server_hedged_read_ops	在进程生命周期内尝试的hedged reads总数。
impala_impala_server_num_open_beeswax_sessions	打开Beeswax会话的数量。
impala_impala_server_backend_num_queries_executed	在进程的生命周期内在此后端执行的查询总数。
impala_impala_server_num_fragments	在进程生命周期内处理的查询片段总数。
impala_rpc_impala_ControlService_rpcs_queue_overflow	ControlService由于服务队列溢出而被拒绝的传入RPC总数。
impala_impala_server_hedged_read_ops_win	Hedged read比常规读取操作快的总次数。
impala_mem_tracker_DataStreamService_current_usage_bytes	Memtracker DataStreamService当前使用的字节数。
impala_impala_server_backend_num_queries_executing	当前在此后端上执行的查询数。
impala_cluster_membership_executor_groups_total_healthy	处于健康状态的执行器组总数。
impala_rpc_impala_DataStreamService_rpcs_queue_overflow	DataStreamService由于服务队列溢出而被拒绝的传入RPC总数。
impala_cluster_membership_backends_total	向statestore注册的后端总数。

指标	描述
impala_mem_tracker_DataStreamService_peak_usage_bytes	Memtracker DataStreamService峰值使用的字节数。
impala_total_senders_blocked_on_recvr_creation	已被阻止等待接收片段初始化的发件人总数。
impala_mem_tracker_ControlService_peak_usage_bytes	Memtracker ControlService峰值使用字节数。
impala_simple_scheduler_local_assignments_total	本地作业数。
impala_mem_tracker_ControlService_current_usage_bytes	Memtracker ControlService当前使用字节数。
impala_memory_total_used	已使用内存，单位：Byte。
impala_cluster_membership_executor_groups_total	至少有一个执行程序的执行程序组总数。
impala_memory_rss	RSS的内存大小，包括TCMalloc、缓冲池和JVM，单位：Byte。
impala_total_senders_timedout_waiting_for_recvr_creation	超时等待接收片段初始化的发送者总数。
impala_senders_blocked_on_recvr_creation	等待接收片段初始化的发送者数量。
impala_simple_scheduler_assignments_total	作业数。
impala_memory_mapped_bytes	进程中内存映射的总字节数（虚拟内存大小），单位：Byte。

4.2.2.9. HUE指标

本文为您介绍HUE指标的详细信息。

开启HUE指标采集

1. 登录阿里云E-MapReduce控制台。
2. 在HUE组件中添加如下参数，详情请参见添加组件参数。

Key	Value	说明
desktop.metrics.enable_web_metrics	True	是否开启Web服务指标采集： ○ True：开启。 ○ False：不开启。
desktop.metrics.location	/var/log/hue/metrics.json	设置指标采集的存储路径，默认为/var/log/hue/metrics.json，建议不要修改。
desktop.metrics.collection_interval	30000	指标采集的时间间隔，单位：ms。

3. 重启HUE组件，详情请参见重启服务。
4. 登录header-1，执行如下命令，变更metrics.json文件权限，详情请参见登录集群。

```
chmod 755 /var/log/hue/metrics.json
```

开启HUE指标采集后，您可以参见查看集群指标，进入HUE服务监控页面，查看监控详情。关于HUE指标的详细说明，请参见HUE指标说明。

HUE指标说明

指标	描述
hue_requests_response_time_avg	请求响应时间平均值。
hue_requests_response_time_95_percentile	95%的请求响应时间。
hue_requests_response_time_std_dev	请求响应时间标准差。

指标	描述
hue_requests_response_time_median	50%的请求响应时间。
hue_requests_response_time_75_percentile	75%的请求响应时间。
hue_requests_response_time_count	请求响应时间计数。
hue_requests_response_time_5m_rate	最近5分钟的请求响应速率。
hue_requests_response_time_min	请求响应时间最小值。
hue_requests_response_time_sum	请求响应时间总和。
hue_requests_response_time_max	请求响应时间的最大值。
hue_requests_response_time_mean_rate	请求响应速率平均值。
hue_requests_response_time_99_percentile	99%的最近一小时请求响应时间。
hue_requests_response_time_15m_rate	最近15分钟请求响应速率。
hue_requests_response_time_999_percentile	99.9%的请求响应时间。
hue_requests_response_time_1m_rate	最近1分钟的请求响应速率。
hue_users_active_total	活跃用户总数。
hue_users_active	最近1小时的活跃用户数。
hue_users	用户总数。
hue_threads_total	当前线程总数。
hue_threads_daemon	常驻线程数量。
hue_queries_number	查询数量总和。
hue_requests_exceptions	当前异常请求数。
hue_requests_active	当前活跃请求数。

4.2.2.10. Kudu指标

本文为您介绍Kudu指标的详细信息。

参数	指标	描述
op_apply_queue_length（99）	kudu_op_apply_queue_length_percentile_99	99%的操作队列的长度。
op_apply_queue_length（75）	kudu_op_apply_queue_length_percentile_75	75%的操作队列的长度。
op_apply_queue_length（mean）	kudu_op_apply_queue_length_mean	操作队列的长度的平均值。
rpc_incoming_queue_time（99）	kudu_rpc_incoming_queue_time_percentile_99	99%的RPC队列的等待时间，单位：μs。
rpc_incoming_queue_time（75）	kudu_rpc_incoming_queue_time_percentile_75	75%的RPC队列的等待时间，单位：μs。
rpc_incoming_queue_time（mean）	kudu_rpc_incoming_queue_time_mean	RPC队列的等待时间的平均值，单位：μs。
reactor_load_percent（99）	kudu_reactor_load_percent_percentile_99	99%的Reactor线程的负载。
reactor_load_percent（75）	kudu_reactor_load_percent_percentile_75	75%的Reactor线程的负载。

参数	指标	描述
reactor_load_percent（mean）	kudu_reactor_load_percent_mean	Reactor线程的负载的平均值。
op_apply_run_time（99）	kudu_op_apply_run_time_percentile_99	99%的操作执行时间，单位：μs。
op_apply_run_time（75）	kudu_op_apply_run_time_percentile_75	75%的操作执行时间，单位：μs。
op_apply_run_time（mean）	kudu_op_apply_run_time_mean	操作执行时间的平均值，单位：μs。
op_prepare_run_time（99）	kudu_op_prepare_run_time_percentile_99	99%的操作准备时间，单位：μs。
op_prepare_run_time（75）	kudu_op_prepare_run_time_percentile_75	75%的操作准备时间，单位：μs。
op_prepare_run_time（mean）	kudu_op_prepare_run_time_mean	操作准备时间的平均值，单位：μs。
flush_mrs_duration（99）	kudu_flush_mrs_duration_percentile_99	99%的MemRowSet flush时间，单位：ms。
flush_mrs_duration（75）	kudu_flush_mrs_duration_percentile_75	75%的MemRowSet flush时间，单位：ms。
flush_mrs_duration（mean）	kudu_flush_mrs_duration_mean	MemRowSet flush时间的平均值，单位：ms。
log_append_latency（99）	kudu_log_append_latency_percentile_99	99%的日志的append时间，单位：μs。
log_append_latency（75）	kudu_log_append_latency_percentile_75	75%的日志的append时间，单位：μs。
log_append_latency（mean）	kudu_log_append_latency_mean	日志的append时间的平均值，单位：μs。
flush_dms_duration（99）	kudu_flush_dms_duration_percentile_99	99%的DeltaMemStore flush时间，单位：ms。
flush_dms_duration（75）	kudu_flush_dms_duration_percentile_75	75%的DeltaMemStore flush时间，单位：ms。
flush_dms_duration（mean）	kudu_flush_dms_duration_mean	DeltaMemStore flush时间的平均值，单位：ms。
op_prepare_queue_length（99）	kudu_op_prepare_queue_length_percentile_99	99%的准备队列的长度。
op_prepare_queue_length（75）	kudu_op_prepare_queue_length_percentile_75	75%的准备队列的长度。
op_prepare_queue_length（mean）	kudu_op_prepare_queue_length_mean	准备队列的长度的平均值。
log_gc_duration（99）	kudu_log_gc_duration_percentile_99	99%的日志GC的时间，单位：ms。
log_gc_duration（75）	kudu_log_gc_duration_percentile_75	75%的日志GC的时间，单位：ms。
log_gc_duration（mean）	kudu_log_gc_duration_mean	日志GC的时间的平均值，单位：ms。
log_sync_latency（99）	kudu_log_sync_latency_percentile_99	99%的日志Sync的时间，单位：μs。
log_sync_latency(75)	kudu_log_sync_latency_percentile_75	75%的日志Sync的时间，单位：μs。
log_sync_latency（mean）	kudu_log_sync_latency_mean	日志Sync的时间的平均值，单位：μs。
prepare_queue_time（99）	kudu_op_prepare_queue_time_percentile_99	99%的操作在准备队列的等待时间，单位：μs。

参数	指标	描述
prepare_queue_time（75）	kudu_op_prepare_queue_time_percentile_75	75%的操作在准备队列的等待时间，单位：μs。
prepare_queue_time（mean）	kudu_op_prepare_queue_time_mean	操作在准备队列的等待时间的平均值，单位：μs。
rpc_connections_accepted	kudu_rpc_connections_accepted	RPC请求接收的数量。
block_cache_usage	kudu_block_cache_usage	Tserver Block缓存的使用量，单位：Byte。
active_scanners	kudu_active_scanners	处于Active状态的Scanner数量。
data_dirs_full	kudu_data_dirs_full	Full状态的数据目录个数。
rpcs_queue_overflow	kudu_rpcs_queue_overflow	RPC队列溢出次数。
cluster_replica_skew	kudu_cluster_replica_skew	服务器上承载的最多的tablet数量与最少的tablet数量的差值。
log_gc_running	kudu_log_gc_running	正在GC的日志数量。
data_dirs_failed	kudu_data_dirs_failed	失效的数据目录个数。
leader_memory_pressure_rejections	kudu_leader_memory_pressure_rejections	内存压力拒绝的请求个数。
transaction_memory_pressure_rejections	kudu_transaction_memory_pressure_rejections	内存压力拒绝的事务个数。

4.2.2.11. ClickHouse指标

本文为您介绍ClickHouse指标的详细信息。

指标	描述
clickhouse_server_events_ReplicatedPartFailedFetches	数据无法从Replicated*MergeTree表中任一副本获取的次数。
clickhouse_server_events_ReplicatedPartChecksFailed	Replicated*MergeTree表中数据检查失败的次数。
clickhouse_server_events_ReplicatedDataLoss	Replicated*MergeTree表中数据不在任何一个副本中的次数。
clickhouse_server_events_ReplicatedMetaDataChecksFailed	Replicated*MergeTree表检查元数据失败的次数。
clickhouse_server_events_ReplicatedMetaDataLoss	Replicated*MergeTree表中元数据丢失的次数。
clickhouse_server_events_DuplicatedInsertedBlocks	写入Replicated*MergeTree表中的Block重复的次数。
clickhouse_server_events_ZooKeeperUserExceptions	Zookeeper中与ClickHouse状态相关错误出现的次数。
clickhouse_server_events_ZooKeeperHardwareExceptions	ZooKeeper网络或类似的错误出现的次数。
clickhouse_server_events_ZooKeeperOtherExceptions	ZooKeeper中非硬件或状态错误出现的次数。
clickhouse_server_events_DistributedConnectionFailTry	分布式连接重试出错的次数。
clickhouse_server_events_DistributedConnectionMissingTable	分布式连接无法找到表的次数。
clickhouse_server_events_DistributedConnectionStaleReplica	分布式连接得到的副本不新鲜的次数。

指标	描述
clickhouse_server_events_DistributedConnectionFailAtAll	在所有次重试结束后分布式连接失败的次数。
clickhouse_server_events_SlowRead	Slow Read的次数。
clickhouse_server_events_ReadBackoff	由于Slow Read导致的线程减少的次数。
clickhouse_server_metrics_BackgroundPoolTask	background_pool中的任务个数。
clickhouse_server_metrics_BackgroundMovePoolTask	background_move_pool中的任务个数。
clickhouse_server_metrics_BackgroundSchedulePoolTask	schedule_pool中的任务个数。
clickhouse_server_metrics_BackgroundBufferFlushSchedulePoolTask	buffer_flush_schedule_pool中的任务个数。
clickhouse_server_metrics_BackgroundDistributedSchedulePoolTask	distributed_schedule_pool中的任务个数。
clickhouse_server_metrics_BackgroundTrivialSchedulePoolTask	trivial_schedule_pool中的任务个数。
clickhouse_server_metrics_TCPConnection	TCP连接个数。
clickhouse_server_metrics_HTTPConnection	HTTP连接个数。
clickhouse_server_metrics_InterserverConnection	用于从其他副本上获取数据的连接个数。
clickhouse_server_metrics_MemoryTracking	Server使用的总内存，单位：Byte。
clickhouse_server_metrics_MemoryTrackingInBackgroundProcessingPool	background_pool中任务执行所使用的内存，单位：Byte。
clickhouse_server_metrics_MemoryTrackingInBackgroundMoveProcessingPool	background_move_pool中任务执行所使用的内存，单位：Byte。
clickhouse_server_metrics_MemoryTrackingInBackgroundBufferFlushSchedulePool	buffer_flush_schedule_pool中任务执行所使用的内存，单位：Byte。
clickhouse_server_metrics_MemoryTrackingInBackgroundSchedulePool	schedule_pool中任务执行所使用的内存，单位：Byte。
clickhouse_server_metrics_MemoryTrackingInBackgroundDistributedSchedulePool	distributed_schedule_pool中任务执行所使用的内存，单位：Byte。
clickhouse_server_metrics_MemoryTrackingInBackgroundTrivialSchedulePool	trivial_schedule_pool中任务执行所使用的内存，单位：Byte。
clickhouse_server_metrics_MemoryTrackingForMerges	后台执行Merge时使用的内存，单位：Byte。

4.2.2.12. Flink指标

本文为您介绍Flink指标的详细信息。

注意事项

以下几个指标参数需要Flink作业中配置了上下游的source和sink才会有输出：

- currentEmitEventTimeLag
- currentFetchEventTimeLag
- OperatorCurrentSendTime
- sourceIdleTime
- watermarkLag

Flink指标说明

- Overview

参数	指标	描述
Num Of RunningJobs	numRunningJobs	JM中正在运行的作业数。
Job Uptime	job_uptime	作业已运行时间，单位：ms。仅支持返回单个系列或表的查询。
TaskSlots Available	taskSlotsAvailable	当前可用的TaskSlots数量。
TaskSlots Total	taskSlotsTotal	TaskSlots的总数量。
Num of TM	numRegisteredTaskManagers	已注册的TM数量。
sourceIdleTime	sourceIdleTime	源没有处理任何记录的时间，单位：ms。
currentFetchEventTimeLag	currentFetchEventTimeLag	业务延时（fetch=数据发生时间与数据进入Flink Source时间之间的差值）。
currentEmitEventTimeLag	currentEmitEventTimeLag	业务延时（emit=数据发生时间与数据离开Flink Source时间之间的差值）。

- Checkpoint

参数	指标	描述
Num of Checkpoints	totalNumberOfCheckpoints	检查点总数。
	numberOfFailedCheckpoints	失败的检查点数量。
	numberOfCompletedCheckpoints	已完成的检查点数量。
	numberOfInProgressCheckpoints	正在进行的检查点数量。
lastCheckpointDuration	lastCheckpointDuration	最近一个检查点完成时间，单位：ms。
lastCheckpointSize	lastCheckpointSize	最近一个检查点的大小，单位：Byte。
lastCheckpointRestoreTimestamp	lastCheckpointRestoreTimestamp	协调器上最近一个检查点的恢复时间，单位：ms。

- Network

参数	指标	描述
InPool Usage	inPoolUsage	输入缓冲区使用量。
OutPool Usage	outPoolUsage	输出缓冲区使用量。
OutputQueue Length	outputQueueLength	输出缓冲区排队数量。
InputQueue Length	inputQueueLength	输入缓冲区排队数量。

- IO

参数	指标	描述
numBytesIn PerSecond	numBytesInLocalPerSecond	每秒本地读取数据的字节数。
	numBytesInRemotePerSecond	每秒远端读取数据的字节数。
	numBuffersInLocalPerSecond	每秒本地读取网络缓冲区的数量。
	numBuffersInRemotePerSecond	每秒远端读取网络缓冲区的数量。
numBytesOut PerSecond	numBytesOutPerSecond	每秒发出字节数。
	numBuffersOutPerSecond	每秒发出网络缓冲区的数量。
	numRecordsInPerSecond	每秒接收的记录数。

Task numRecords I/O PerSecond 参数	指标	描述
	numRecordsOutPerSecond	每秒发出的记录数。
Task numRecords I/O	numRecordsIn	接收的记录数。
	numRecordsOut	发出的记录数。
Operator CurrentSendTime	currentSendTime	发送最新一条记录的耗时时间，单位：ms。

- Watermark

参数	指标	描述
Task InputWatermark	currentInputWatermark	任务收到最后一个水印的时间，单位：ms。
Operator In/Out Watermark	currentInputWatermark	算子收到最后一个水印的时间，单位：ms。
	currentOutputWatermark	算子发出最后一个水印的时间，单位：ms。
watermarkLag	watermarkLag	Watermark滞后时间，单位：ms。

- CPU

参数	指标	描述
JM CPU Load	CPU_Load	JM CPU使用率。
TM CPU Load	CPU_Load	TM CPU使用率。
CPU Usage	CPU_Usage	TM CPU使用率（基于ProcessTree）。

- Memory

参数	指标	描述
JM Heap Memory	Memory_Heap_Used	JM Heap Memory已使用量，单位：Byte。
	Memory_Heap_Committed	JM Heap Memory已申请量，单位：Byte。
	Memory_Heap_Max	JM Heap Memory最大可用量，单位：Byte。
JM NonHeap Memory	Memory_NonHeap_Used	JM NonHeap Memory已使用量，单位：Byte。
	Memory_NonHeap_Committed	JM NonHeap Memory已申请量，单位：Byte。
	Memory_NonHeap_Max	JM NonHeap Memory最大可用量，单位：Byte。
TM Heap Memory	Memory_Heap_Used	TM Heap Memory已使用量，单位：Byte。
	Memory_Heap_Committed	TM Heap Memory已申请量，单位：Byte。
	Memory_Heap_Max	TM Heap Memory最大可用量，单位：Byte。
TM NonHeap Memory	Memory_NonHeap_Used	TM NonHeap Memory已使用量，单位：Byte。
	Memory_NonHeap_Committed	TM NonHeap Memory已申请量，单位：Byte。
	Memory_NonHeap_Max	TM NonHeap Memory最大可用量，单位：Byte。
Memory RSS	Memory_RSS	TM当前已使用的堆内存量，单位：Byte。

- JVM

参数	指标	描述
JM Threads	Threads_Count	JM活跃线程总数。
TM Threads	Threads_Count	TM活跃线程总数。

参数	指标	描述
JM GC Time	GarbageCollector_PS_Scavenge_Time	JM年轻代垃圾回收器运行时间。
	GarbageCollector_PS_MarkSweep_Time	JM老年代“标记-清除”垃圾回收器的运行时间。
JM GC Count	GarbageCollector_PS_Scavenge_Count	JM年轻代垃圾回收器运行次数。
	GarbageCollector_PS_MarkSweep_Count	JM老年代“标记-清除”垃圾回收器的运行次数。
TM GC Count	GarbageCollector_PS_Scavenge_Count	TM年轻代垃圾回收器运行次数。
	GarbageCollector_PS_MarkSweep_Count	TM老年代“标记-清除”垃圾回收器的运行次数。
TM GC Time	GarbageCollector_PS_Scavenge_Time	TM年轻代垃圾回收器运行时间。
	GarbageCollector_PS_MarkSweep_Time	TM老年代“标记-清除”垃圾回收器的运行时间。
TM ClassLoader	ClassLoader_ClassesLoaded	TM自JVM启动以来已加载的类总数。
	ClassLoader_ClassesUnloaded	TM自JVM启动以来已卸载的类总数。
JM ClassLoader	ClassLoader_ClassesLoaded	JM自JVM启动以来已加载的类总数。
	ClassLoader_ClassesUnloaded	JM自JVM启动以来已卸载的类总数。

4.2.3. 作业列表

作业执行完成后，您可以在作业列表页面查看作业整体运行状况。作业列表页面展示了作业基础状态统计和作业类型统计，同时对于失败的作业提供了智能诊断。

前提条件

已创建集群，详情请参见创建集群。

查看作业

1. 进入监控大盘。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**监控大盘**。
2. 在监控大盘页面，单击上方的**作业列表**页签。
   - 查看全部作业

     默认展示全部作业。您可以通过集群ID、作业名称、作业ID、作业类型和作业状态等条件过滤查看作业。
   - 查看最近一周失败作业

     在作业大盘页面，单击**最近一周失败作业**。

     作业列表页面展示最近一周失败作业列表。
3. 单击目标作业操作列的**查看详情**。

   可以看到作业的摘要信息，详细信息以及诊断建议。

4.2.4. 事件中心

事件中心页面展示了所有集群产生的事件，事件等级包括CRITICAL、WARN和INFO三类。

前提条件

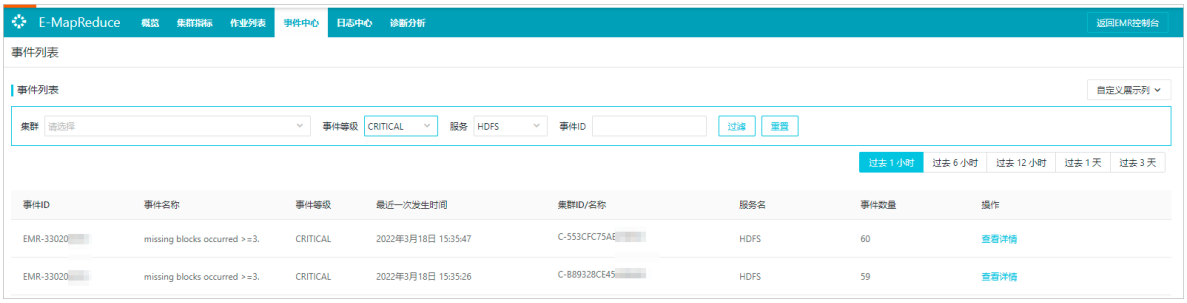
已创建集群，详情请参见创建集群。

查看事件

1. 进入监控大盘。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**监控大盘**。
2. 在监控大盘页面，单击上方的**事件中心**页签。
3. 您可以通过配置过滤条件，查询事件。

   默认展示过去1个小时所有集群，HDFS服务CRITICAL等级的事件。

   > **说明** 所有类型的事件，都支持根据集群、服务、发生的时间范围和告警级别进行过滤。



4. 单击目标事件操作列的**查看详情**。

   可以看到事件的详细信息。

4.2.5. 日志中心

日志中心功能支持对账号下所有集群核心服务日志和主机系统日志进行关键词搜索，可以在不登录主机的情况下快速查看服务关键日志，在集群主机日志轮转清理之后仍然可以搜到日志。

前提条件

已创建集群，详情请参见创建集群。

查询日志

1. 进入监控大盘。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**监控大盘**。
2. 在监控大盘页面，单击上方的**日志中心**页签。
3. 通过配置过滤条件，查询日志。

   日志搜索支持根据集群、日志文件、主机和时间范围进行过滤，日志结果包含日志所在集群的主机名、主机 IP、日志在主机上的路径以及日志内容。

   日志搜索支持的日志文件。

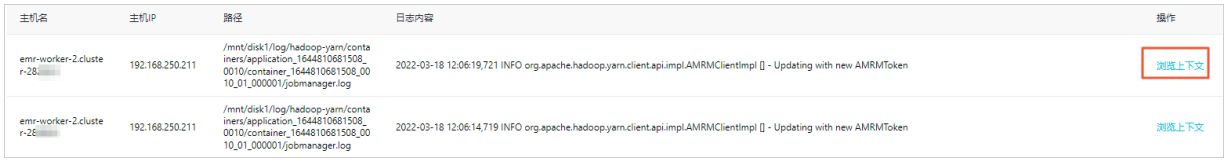
服务	日志文件
YARN	◦ yarn-container-log ◦ nodemanager-out ◦ nodemanager-log ◦ proxyserver-out ◦ proxyserver-log ◦ resourcemanager-out ◦ resourcemanager-log ◦ timelineserver-out ◦ timelineserver-log ◦ historyserver-out ◦ historyserver-log

服务	日志文件
SPARK	◦ spark-history-gc ◦ spark-thriftserver-out ◦ sparkhistory-out
PRESTO	◦ presto-server-log ◦ presto-launcher-log ◦ presto-http-request-log
FLUME	◦ flume-gc ◦ flume-log
HIVE	◦ hiveserver2-err ◦ metastore-log ◦ hiveserver2-log ◦ hive-log
HBASE	◦ thriftserver-gc ◦ thriftserver-out ◦ thriftserver-log ◦ hregionserver-gc-log ◦ hregionserver-out ◦ hregionserver-log ◦ hmaster-gc-log ◦ hmaster-out ◦ hmaster-log ◦ hbase-audit-log ◦ hbase-gc-log
HDFS	◦ namenode-gc ◦ namenode-out ◦ namenode-log ◦ hdfs-audit-log ◦ datanode-log ◦ datanode-out ◦ zkfc-out ◦ zkfc-log ◦ journalnode-out ◦ journalnode-log

服务	日志文件
HOST	var-log-messages
ZEPPELIN	○ zeppelin-log ○ zeppelin-out
HUE	○ hue-supervisor-log ○ hue-runcpserver-log ○ hue-access-log ○ hue-error-log
ZOOKEEPER	○ zookeeper-gc ○ zookeeper-out
IMPALA	○ impala-statestored-launch-log ○ impala-statestored-error-log ○ impala-statestored-warning-log ○ impala-statestored-info-log ○ impala-catalogd-launch-log ○ impala-catalogd-error-log ○ impala-catalogd-warning-log ○ impala-catalogd-info-log ○ impala-impalad-launch-log ○ impala-impalad-error-log ○ impala-impalad-warning-log ○ impala-impalad-info-log
OOZIE	○ oozie-audit-log ○ oozie-instrumentation-log ○ oozie-error-log ○ oozie-server-catalina-log ○ oozie-ops-log ○ oozie-jpa-log ○ oozie-log

浏览日志上下文

在日志搜索结果列表中，单击目标日志操作列的**浏览上下文**，可以查看日志的上下文信息。



4.3. 集群告警

4.3.1. 创建阈值报警规则

云监控（CloudMonitor）是阿里云的一种监控告警服务，当您需要监控E-MapReduce资源的使用和运行情况时，可以通过创建阈值报警规则，实现监控项超过设定阈值后自动发送报警通知的功能，帮助您及时了解监控数据异常并快速进行处理。

前提条件

已创建集群，详情请参见创建集群。

操作步骤

1. 登录云监控控制台。

2. 在左侧导航栏，选择**报警服务 > 报警规则**。
3. 在**阈值报警**页面，单击**创建报警规则**。
4. 在**创建报警规则**页面，设置报警规则相关信息，具体操作请参见创建报警规则。

参数	说明
产品	从**产品**列表中选择**E-MapReduce**。
资源范围	报警规则的作用范围。取值： ○ **全部资源**：表示该规则作用在用户名下E-MapReduce的全部集群上。 ○ **应用分组**：表示该规则作用在用户名下E-MapReduce的指定应用分组内的全部集群上。 ○ **实例**：表示该规则只作用在指定的集群上。
规则描述	报警规则的主体，定义在监控数据满足指定条件时，触发报警规则。例如：CPU使用率5分钟平均值>=90%，持续3个周期，则报警服务5分钟检查一次数据是否满足平均值>=90%，只检测3次。 **说明** 规则详细信息，请参见报警规则。 规则描述的设置方法如下： i. 单击**添加规则**。 ii. 在**添加规则描述**面板，设置规则名称、监控指标类型、监控指标、阈值、报警级别和报警方式等。 iii. 单击**确定**。
通道沉默周期	报警发生后未恢复正常，间隔多久重复发送一次报警通知。 **说明** 单击**高级设置**，可设置该参数。
生效时间	报警规则的生效时间，报警规则只在生效时间内才会检查监控数据是否需要报警。 **说明** 单击**高级设置**，可设置该参数。
报警联系人组	发送报警的联系人组。 如果您需要新建联系人组，创建详情请参见创建报警联系人或报警联系组。
报警回调	填写公网可访问的URL，云监控会将报警信息通过POST请求推送到该地址，目前仅支持HTTP协议。
弹性伸缩	如果您选中**弹性伸缩**，当报警发生时，会触发相应的伸缩规则。您需要设置弹性伸缩的**地域**、**弹性伸缩组**和**弹性伸缩规则**。 ○ 创建弹性伸缩组的操作方法，请参见创建伸缩组。 ○ 创建弹性伸缩规则的操作方法，请参见配置伸缩规则。
日志服务	如果您选中**日志服务**，当报警发生时，会将报警信息写入日志服务。您需要设置日志服务的**地域**、**Project**和**Logstore**。 创建Project和Logstore的操作方法，请参见快速入门。
消息服务MNS-Topic	如果您打开消息服务MNS-Topic开关，当报警发生时，会将报警信息写入消息服务的主题。您需要设置消息服务的地域和主题。 关于如何创建主题，请参见创建主题。

5. 单击**确定**。

报警规则

服务名	指标名	指标含义
	NameNodeIpcPortOpen	NameNode的IPC端口的可用性： ● 1：可用 ● 0：不可用
	TotalDFSUsedPercent	集群的HDFS总容量使用百分比。
	DataNodeDfsUsedPercent	DataNode节点的DFS使用率。

服务名	指标名	指标含义
HDFS	DataNodeIpcPortOpen	DataNode中IPC端口的可用性： • 1：可用 • 0：不可用
	JournalNodeRpcPortOpen	JournalNode的RPC端口的可用性： • 1：可用 • 0：不可用
	ZKFCPortOpen	ZKFC端口的可用性： • 1：可用 • 0：不可用
	dfs.FSNamesystem.MissingBlocks	丢失的块数。
	dfs.datanode.VolumeFailures	HDFS检测出的坏盘数。
YARN	ResourceManagerPortOpen	ResourceManager的服务端口的可用性： • 1：可用 • 0：不可用
	JobHistoryPortOpen	JobHistory的服务端口的可用性： • 1：可用 • 0：不可用
	yarn.ClusterMetrics.NumUnhealthyNM	Unhealthy的NodeManager个数。
	ProxyServerPortOpen	WebAppProxy端口的可用性： • 1：可用 • 0：不可用
	TimelineServerPortOpen	TimelineServer的服务端口的可用性： • 1：可用 • 0：不可用
Hive	MetastorePortOpen	HiveMetaStore端口的可用性： • 1：可用 • 0：不可用
	HiveServer2PortOpen	HiveServer2的服务端口的可用性： • 1：可用 • 0：不可用
	ThriftServerPortOpen	ThriftServer的服务端口的可用性： • 1：可用 • 0：不可用
HBase	HMasterIpcPortOpen	HMaster的IPC端口可用性： • 1：可用 • 0：不可用
	HRegionServerIpcPortOpen	HRegionServer的IPC的端口可用性： • 1：可用 • 0：不可用
ZooKeeper	ZKClientPortOpen	ZooKeeper客户端监听端口的可用性： • 1：可用 • 0：不可用

服务名	指标名	指标含义
Hue	HuePortOpen	Hue端口的可用性： • 1：可用 • 0：不可用
Storm	StormNimbusThriftPortOpen	StormNimbus的Thrift端口的可用性： • 1：可用 • 0：不可用
HOST	proc_total	总进程数目。
	part_max_used	磁盘分区使用的最大百分比。
	disk_free_percent_mnt_disk1	剩余空间百分比。
	disk_free_percent_rootfs	根文件系统磁盘空间占比。

4.3.2. 创建事件报警规则

本文为您介绍如何创建事件报警规则和调试系统事件，以便在E-MapReduce发生系统异常时，您能及时接收报警通知并处理异常。

前提条件

如果事件报警规则需要作用于指定应用分组的实例上，则请确保您已创建应用分组，且已将资源添加至该应用分组，操作方法请参见创建应用分组和添加资源至应用分组。

创建事件报警规则

1. 登录云监控控制台。
2. 在左侧导航栏，单击**报警服务 > 报警规则**。
3. 在**报警规则列表**页面，单击**事件报警**页签。
4. 在**事件报警**页面，单击**创建事件报警**。
5. 在**创建/修改事件报警**页面，设置事件报警规则和报警方式。
   i. 在**基本信息**区域，填写**时间报警规则**。
   ii. 在**事件报警规则**区域，选择事件类型。
   - 系统事件

     当**事件类型**选择**系统事件**时，您需要根据所需设置**产品类型**、**事件类型**、**事件等级**、**事件名称**、**资源范围**和**报警方式**。

     > **说明** 事件名称详细信息，请参见系统事件。

     系统事件支持的报警方式包括：**报警通知**、**消息服务队列**、**函数服务**、**URL回调**和**日志服务**。
   - 自定义事件。

     当**事件类型**选择**自定义事件**时，您需要根据所需设置**所属应用分组**、**事件名称**、**规则描述**和**通知方式**。
6. 单击**确定**。

调试系统事件

创建事件报警规则后，您可以使用系统事件的调试功能，验证系统事件报警规则中设置的消息服务队列、函数计算、URL回调和日志服务是否能正常被触发。

1. 登录云监控控制台。
2. 在左侧导航栏，单击**报警服务 > 报警规则**。
3. 在**报警规则列表**页面，单击**事件报警**页签。
4. 在**事件报警**页面，单击系统事件报警规则对应**操作**列的**调试**。
5. 在**创建事件调试**页面，选择待调试事件，根据实际情况修改调试内容。
6. 单击**确定**。
   云监控将根据内容发送一个事件，触发报警规则设置的报警通知、消息服务队列、函数计算、URL回调和日志服务。

系统事件

服务名	事件ID	事件名称（中文）	事件名称（英文）	事件含义
HDFS	EMR-330200049 EMR-350202005	NameNode发生主备切换	Maintenance:HDFS.NameNode.ActiveStandbySwitch	NameNode发生主备切换。 处理方式：需要判断主节点无法提供服务超时的原因，可能原因包括GC问题和block report storm等。
	EMR-350200012	NameNode发生OOM	Maintenance:HDFS.NameNode.OOM	NameNode发生OOM。 处理方式：您可以在HDFS服务的配置页面，调大**hadoop_namenode_heapsize**的值，以增加内存。
	EMR-350200005	HDFS发生目录格式化	Maintenance:HDFS.NameNode.DirectoryFormatted	NameNode元数据目录被删除。 处理方式：请提交工单或购买专家服务处理。 警告 禁止重启HDFS服务，以防数据丢失。
	EMR-350200006	NameNode加载FsImage异常	Maintenance:HDFS.NameNode.LoadFsImageException	NameNode读取FSImage异常。 处理方式：可能是由于云盘数据异常导致的，请提交工单处理。
	EMR-350200008	NameNode异常退出	Maintenance:HDFS.NameNode.ExitUnexpectely	NameNode节点异常退出。 处理方式：查看NameNode节点异常退出的日志，根据日志处理。
	EMR-350200015	HDFS写JournalNode超时	Maintenance:HDFS.NameNode.WriteToJournalNodeTimeout	NameNode写入JournalNode超时。 处理方式：检查JournalNode服务是否正常，同时检查JournalNode所在机器与网络情况是否有异常流量。
	EMR-350200014	NameNode资源不足	Maintenance:HDFS.NameNode.ResourceLow	NameNode节点磁盘空间不足，导致NameNode进入安全模式。 处理方式：检查Master节点*/mnt/disk1*的磁盘空间，是否有日志过大异常，如果没有异常，可以扩容磁盘。
	EMR-350200007	NameNode同步日志失败	Maintenance:HDFS.NameNode.SyncJournalFailed	NameNode操作JournalNode sync超时。 处理方式：检查JournalNode服务是否正常，同时检查JournalNode所在机器与网络情况是否有异常流量。

服务名	事件ID	事件名称（中文）	事件名称（英文）	事件含义
	EMR-350201008	DataNode OOM不能创建新的Native线程	Maintenance:HDFS.DataNode.OOM.UnableToCreateNewNativeThread	DataNode无法创建新的进程。 处理方式：通常是异常任务在该DataNode创建了大量线程，导致了DataNode无法创建新的线程，因此可以在Master节点通过root用户执行命令 `ps -eo nlwp,pid,args --sort nlwp |tail -n 5` ，找到使用线程数最大的5个进程，并进行分析。
	EMR-350202006	ZKFC监控NameNode健康状态时发生传输层异常事件	Maintenance:HDFS.ZKFC.TransportLevelExceptionInMonitorHealth	ZKFC检查NameNode健康度超时。 处理方式：检查NameNode超时的原因，可能原因包括NameNode内存大小、 GC或者是block report storm。如果无异常，集群规模较大，可以适当调大**ha.health-monitor.rpc-timeout.ms**配置，防止超时。
	EMR-350202002	ZKFC不能连接ZookeeperQuorum	Maintenance:HDFS.ZKFC.UnableToConnectToQuorum	ZKFC无法连接ZooKeeper。 处理方式：查看ZooKeeper服务，检查是否有异常连接没有释放导致的连接过多，以至于ZKFC无法连接ZooKeeper。
	EMR-350202001	ZKFC不能启动	Maintenance:HDFS.ZKFC.UnableToStartZKFC	无法启动ZKFC服务。 处理方式：查看ZKFC的日志，根据日志处理。
	EMR-250201009	JavaHeapSpace引起OOM错误	Maintenance:HDFS.DataNode.OomForJavaHeapSpace	DataNode发生OOM。 处理方式：您可以在HDFS服务的配置页面，调大的**hadoop_datanode_heapsize**值，以增加内存。
	EMR-250201004	DataNode进程异常退出	Maintenance:HDFS.DataNode.ExitUnexpected	DataNode异常退出。 处理方式：查看导致DataNode服务异常退出的日志，根据日志处理。
	EMR-330300053	ResourceManager发生主备切换	Maintenance:YARN.ResourceManager.ActiveStandbySwitch	高可用集群ResourceManager发生主备切换。 处理方式：检查ResourceManager主备切换原因，可能原因包括GC问题、Master节点资源不足或内核错误等。
	EMR-350300015	YARN服务中ZKRMStateStore不能连接ZK	Maintenance:YARN.ResourceManager.ZKRMStateStoreCannotConnectZK	ResourceManager ZkClient无法连接ZooKeeper。 处理方式：检查ZooKeeper服务，是否由于异常连接没有释放导致无法连接。

服务名	事件ID	事件名称（中文）	事件名称（英文）	事件含义
YARN	EMR-350300011	ResourceManager启动错误	Maintenance:YARN.ResourceManager.ErrorInStarting	ResourceManager启动错误。 处理方式：检查RM启动报错的原因，排查是否是由于错误配置导致的。
	EMR-350300010	ResourceManager切换到Active模式发生错误	Maintenance:YARN.ResourceManager.ErrorInTransitionToActiveMode	ResourceManager切换到Active模式时发生错误。 处理方式：查看切换报错的日志，根据日志处理。
	EMR-350300004	ResourceManager无效配置问题:不能找到RM_HA_ID	Maintenance:YARN.ResourceManager.InvalidConf.CannotFoundRM_HA_ID	ResourceManager错误配置导致找不到HA ID。 处理方式：在配置页面检查配置，是否由于修改配置导致了RM ID异常。
	EMR-350300013	ResourceManager异常退出	Maintenance:YARN.ResourceManager.ExitUnexpected	ResourceManager异常退出。 处理方式：查看导致ResourceManager服务异常退出的日志，根据日志处理。
	EMR-350302004	JobHistory服务启动错误	Maintenance:YARN.JobHistory.StartingError	JobHistory服务启动错误。 处理方式：查看导致JobHistory服务启动错误的日志，根据日志处理。
	EMR-350302003	JobHistory服务异常退出	Maintenance:YARN.JobHistory.ExitUnExpectedly	JobHistory服务异常退出。 处理方式：查看导致JobHistory服务异常退出的日志，根据日志处理。
	EMR-350303004	TimelineServer启动错误	Maintenance:YARN.TimelineServer.ErrorInStarting	Timeline Server启动错误。 处理方式：查看导致Timeline Server服务启动错误的日志，根据日志处理。
	EMR-350303003	TimelineServer异常退出	Maintenance:YARN.TimelineServer.ExitUnexpectedly	Timeline Server异常退出。 处理方式：查看导致Timeline Server服务异常退出的日志，根据日志处理。
	EMR-250300001	ResourceManager发生UnkownHostException异常	Maintenance:YARN.ResourceManager.UnkownHostException	ResourceManager对某些节点无法解析。 处理方式：检查是否使用了DNS服务，是否由于DNS服务不稳定导致。如果没有使用DNS服务，检查是否*/etc/hosts*文件配置了对应的host。
	EMR-350301010	磁盘错误导致不健康的NodeManager	Maintenance:YARN.NodeManager.UnHealthyForDiskFailed	磁盘错误导致不健康的NodeManager。 处理方式：检查NodeManager是否存在磁盘写满情况，存在的话请提交工单处理。

服务名	事件ID	事件名称（中文）	事件名称（英文）	事件含义
	EMR-250301006	NodeManager启动RebootingNodeStatusUpdater失败	Maintenance:YARN.NodeManager.ErrorRebootingNodeStatusUpdater	NodeManager重启NodeStatusUpdater失败。 处理方式：检查NodeManager重连日志，检查ResourceManager服务。
	EMR-250301015	NodeManager发生OOM	Maintenance:YARN.NodeManager.OOM	NodeManager发生OOM。 处理方式：检查NodeManager OOM的原因，排查是否有配置错误导致的内存泄露。
	EMR-350400016	HiveMetaStore发生JDBC通信异常	Maintenance:HIVE.HiveMetaStore.JdbcCommunicationException	Hive MetaStore发生JDBC通信异常。 处理方式：检查元数据连接，使用本地MySql客户端连接进行测试。
	EMR-350400010	HiveMetaStore数据库通信链路失败	Maintenance:HIVE.HiveMetaStore.DataBaseCommunicationLinkFailure	Hive MetaStore数据库连接异常，提示通信链路失败。 处理方式：检查元数据连接，使用本地MySql客户端连接进行测试。
	EMR-350400009	HiveMetaStore数据库连接失败	Maintenance:HIVE.HiveMetaStore.DataBaseConnectionFailed	Hive MetaStore数据库连接失败。 处理方式：检查元数据连接，使用本地MySql客户端连接进行测试。
	EMR-350400014	HiveMetaStore发生OOM	Maintenance:HIVE.HiveMetaStore.OomOccured	Hive MetaStore发生OOM。 处理方式：您可以在Hive服务的配置页面，调大hive_meastore_heapsize的值，以增加内存。
	EMR-350400015	HiveMetastore数据库磁盘空间用尽	Maintenance:HIVE.HiveMetaStore.DataBaseDiskQuotaUsedup	元存储连接超过限制。 处理方式：如果使用本地MySql或者RDS作为元数据存储，需要手动提升限制大小。如果使用统一元数据作为元数据存储，请提交工单处理。
	EMR-350400006	HiveMetastore超过最大查询数	Maintenance:HIVE.HiveMetaStore.MaxQuestionsExceeded	元数据存储连接超过限制。 处理方式：如果使用本地MySql或者RDS作为元数据存储，需要手动提升限制大小。如果使用统一元数据作为元数据存储，请提交工单处理。
	EMR-350400007	HiveMetastore超过最大更新数	Maintenance:HIVE.HiveMetaStore.MaxUpdatesExceeded	元数据存储连接超过限制。 处理方式：如果使用本地MySql或者RDS作为元数据存储，需要手动提升限制大小。如果使用统一元数据作为元数据存储，请提交工单处理。

服务名	事件ID	事件名称（中文）	事件名称（英文）	事件含义
HIVE	EMR-350400005	HiveMetastore超过最大用户连接数	Maintenance:HIVE.HiveMetaStore.MaxUserConnectionExceeded	元数据存储连接超过限制。 处理方式：如果使用本地MySql或者RDS作为元数据存储，需要手动提升限制大小。如果使用统一元数据作为元数据存储，请提交工单处理。
	EMR-350400002	HiveMetastore配置文件解析错误	Maintenance:HIVE.HiveMetaStore.ParseConfError	配置异常导致解析错误。 处理方式：通常是由于配置异常导致解析错误，因此请检查*Hivemetastore-site.xml*里面的配置，排查配置历史信息并修复问题。
	EMR-350400008	HiveMetastore请求的表丢失	Maintenance:HIVE.HiveMetaStore.RequiredTableMissing	Hive MetaStore的元存储表丢失。 处理方式：检查Metastore版本和元数据存储的Hive版本是否一致，并查看元数据是否损坏。
	EMR-350401013	hiveServer2发生OOM	Maintenance:HIVE.HiveServer2.HiveServer2OOM	HiveServer2发生OOM。 处理方式：您可以在Hive服务的配置页面，调大**hive_server2_heapsize**的值，以增加内存。
	EMR-350401007	不能通过提供的URIs连接到hiveServer2	Maintenance:HIVE.HiveServer2.CannotConnectByAnyURIsProvided	HiveServer2无法连接Hive MetaStore。 处理方式：检查MetaStore服务。
	EMR-350401004	hiveServer2连接ZK超时	Maintenance:HIVE.HiveServer2.ConnectToZkTimeout	HiveServer2连接Zookeeper超时。 处理方式：检查ZooKeeper服务。
	EMR-350401008	hiveServer2启动错误	Maintenance:HIVE.HiveServer2.ErrorStartingHiveServer	HiveServer2启动异常。 处理方式：查看启动异常的日志，定位问题原因并处理。
	EMR-350401006	hiveServer2初始化MetaStore客户端失败	Maintenance:HIVE.HiveServer2.FailedInitMetaStoreClient	HiveServer2初始化MetaStore客户端失败。 处理方式：检查HiveServer2的异常日志，定位无法初始化问题。
	EMR-350401005	hiveServer2连接MetaStoreServer失败	Maintenance:HIVE.HiveServer2.FailedToConnectToMetaStoreServer	HiveServer2连接不上MetaStoreServer。 处理方式：检查Metastore服务

服务名	事件ID	事件名称（中文）	事件名称（英文）	事件含义
HOST	EMR-350100001	主机/var/log/message有OOM异常	Maintenance:HOST.OomFoundInVarLogMessage	主机*/var/log/message*中存在OOM异常日志。 处理方式：通常是因为主机内存不足而终止某进程。因此请检查内存分配是否合理，如果内存偏小，请增加内存。
	EMR-250100006	主机CPU卡顿	Maintenance:HOST.CpuStuck	Linux内核报警存在CPU Stuck。 处理方式：检查是否存在异常的任务长时间占据CPU。
SparkHIstory	EMR-350900001	SparkHistory发生OOM	Maintenance:SPARK.SparkHistory.OomOccured	Spark History服务发生OOM。 处理方式：您可以在Spark服务的配置页面，调大**spark_history_daemon_memory**的值，以增加内存。
Zookeeper	EMR-350500001	ZOOKEEPER不能运行QuorumServer	Maintenance:ZOOKEEPER.UnableToRunQuorumServer	Zookeeper无法启动。 处理方式：检查异常日志和配置文件，确认是否ZNode过多导致的OOM。
	EMR-230500059	Zookeeper发生主从切换	Maintenance:ZOOKEEPER.LeaderFollowerSwitch	ZooKeeper发生主备节点切换。 处理方式：请检查切换原因，根据实际情况确认是否重启ZooKeeper服务。

5.开发指南（新版控制台）
5.1. 组件操作指南
5.1.1. Hive
5.1.1.1. Hive概述

Hive是一个基于Hadoop的数据仓库框架，在大数据业务场景中，主要用来进行数据提取、转化和加载（ETL）以及元数据管理。

背景信息

E-MapReduce（简称EMR）版本中，Hadoop、Hive版本和EMR集群的配套情况，请参见版本概述。

Hive结构

名称	说明
HiveServer2	HiveQL查询服务器，可以配置为Thrift或者HTTP协议，接收来自JDBC客户端提交的SQL请求，支持多客户端并发以及身份验证。
Hive MetaStore	元数据管理模块，此模块被其他引擎所依赖，用于存储Database和Table等元信息。例如，Spark和Presto均依赖此模块作为其元数据管理。
Hive Client	Hive客户端，直接利用该客户端提交SQL作业，根据其设置运行引擎配置，可以将SQL转换成MR作业、Tez作业和Spark作业，该模块在所有EMR节点上均有安装。

Hive语法

EMR产品最大程度的保持了开源社区的语法以及体验，在Hive语法上保持与开源社区Hive语法100%的兼容性。

关于Apache Hive的更多介绍，请参见Apache Hive官网。

入门指导

- Hive的使用请参见Hive作业配置。
- 在Zeppelin中使用Hive的详情请参见Zeppelin概述。

5.1.1.2. EMR Hive功能增强

本文为您介绍E-MapReduce（简称EMR）各版本对应的Hive组件版本，以及各版本中Hive相对开源增强的功能。

Hive针对开源功能增强的功能如下表。

EMR版本	组件版本	功能增强
EMR-5.2.1	Hive 3.1.2	• 修复使用DLF元数据执行 `show create table` 命令，结果显示不正确的问题。 • 优化Hive默认参数，以提升作业性能。 • 修改E-MapReduce控制台上，Hive服务**配置**页面的**hive-env**页签的配置项名称为大写，便于用户使用。 • 修复UDF（User Define Function）导致HiveServer2内存泄露的问题。 • 优化文件系统与MetaStore不一致时写Hive表的报错信息。
EMR-4.8.0	Hive 3.1.2	• 优化了部分默认配置。 • 性能优化：增强CBO。 • 支持一键开启或关闭LDAP功能。 开启或关闭LDAP功能详情，请参见管理LDAP认证。
EMR-4.6.0	Hive 3.1.2	• HCatalog支持Data Lake Formation。 • 支持Hive元数据和作业运行信息输出至DataWorks。
EMR-4.5.0	Hive 3.1.2	• 支持数据湖构建（DLF）元数据。 • 支持Ranger Ownership权限。
EMR-4.4.1	Hive 3.1.2	优化默认的参数配置。

EMR版本	组件版本	功能增强
EMR-4.4.0	Hive 3.1.2	• 升级至3.1.2版本。 • 优化JindoFS。 • 优化MSCK。 • HCatalog支持JindoCommitter。 • 升级HAS依赖。
EMR-4.3.0	Hive 3.1.1	支持自定义部署。
EMR-3.36.1	Hive 2.3.8	• 升级Hive至2.3.8版本。 • 修复使用DLF（DataLakeFormation）元数据执行`show create table`命令时，结果显示不正确的问题。 • 优化Hive默认参数，以提升作业性能。 • 修改E-MapReduce控制台上，Hive服务**配置**页面的**hive-env**页签的配置项名称为大写，便于用户使用。 • 优化文件系统与MetaStore不一致时写Hive表的报错信息。
EMR-3.35.0	Hive 2.3.7	修复Fetch Task相关的社区问题。
EMR-3.34.0	Hive 2.3.7	• 优化了部分默认配置。 • 性能优化：增强CBO。 • 支持一键开启或关闭LDAP功能。 开启或关闭LDAP功能详情，请参见管理LDAP认证。 • 升级Calcite版本至1.12.0。 • 增加参数**hive.security.authorization.sqlstd.confwhitelist.append**。
EMR-3.33.0	Hive 2.3.7	• 升级至2.3.7版本。 • HCatalog支持Data Lake Formation。 • 支持Hive元数据和作业运行信息输出至DataWorks。
EMR-3.32.0	Hive 2.3.5	• 修复了HiveServer连接池泄漏的问题。 • JindoTable支持打开或关闭数据采集功能。 • 优化`ADD COLUMN`的性能。 • 修复了读取HUDI表时数据不正确的问题。 • 默认的参数配置，可以根据集群节点大小调整。
EMR-3.30.0	Hive 2.3.5	• 支持阿里云DLF（Data Lake Formation）元数据。 • 解决了读Delta表空目录时写DUMMY文件问题。 • 升级HAS依赖至2.0.1。
EMR-3.29.0	Hive 2.3.5	• Hive升级至2.3.5.6.0。 • 支持第三方Metastore的功能。 • 增加datalake metastore-client。
EMR-3.28.0	Hive 2.3.5	支持Delta 0.6.0版本。
EMR-3.27.2	Hive 2.3.5	• hcatalog表支持magic committer。 • 移除一些过时的默认配置。
EMR-3.26.3	Hive 2.3.5	hcatalog表支持direct committer。
EMR-3.25.0	Hive 2.3.5	修复自动LOCAL模式下MR任务执行失败的问题

EMR版本	组件版本	功能增强
EMR-3.24.0	Hive 2.3.5	• 增加SQL兼容性检查功能逻辑。 • Hive2.3.5+Hadoop2.8.5组合发布。 • 重启组件时不同步*hiveserver2-site.xml*中的内容至spark-conf下的*hive-site.xml*。 • 支持使用MSCK命令添加增量目录。 • 修复Hive复用tez container时出现的bug。 • 支持使用MSCK命令优化列目录。
EMR-3.23.0	Hive 2.3.5	• 删除老版本的hive hook。 • 添加支持多个count distinct字段的数据倾斜处理优化。 • 解决join不同bucketversion的表时丢数据的问题。
EMR-3.23.0之前版本	Hive 2.x	外部统一数据库保存至Hive Meta，所有使用外部Hive Meta的集群共享同一份Meta信息。

5.1.1.3. 基础使用

5.1.1.3.1. Hive基础操作

本文介绍如何通过Hive在E-MapReduce集群上创建库和表等操作。

前提条件

已创建集群，详情请参见创建集群。

进入Hive命令行

1. 使用SSH方式登录集群，详情请参见登录集群。
2. 执行以下命令，切换为hadoop用户。

   ```
   su hadoop
   ```

3. 执行以下命令，进入Hive命令行。

   ```
   hive
   ```

库操作

本文示例中的数据库以testdb为例介绍。

1. 创建库

   ```
   create database if not exists testdb;
   ```

   当返回信息包含OK时，表示创建库testdb成功。

2. 查看库

   ```
   desc database testdb;
   ```

3. 使用数据库

   ```
   use testdb;
   ```

4. 删除库

   ```
   drop database if exists testdb;
   ```

   当返回信息包含OK时，表示删除库成功。

表操作

本文示例中的表以t为例介绍。

1. 创建表

   ```
   create table if not exists t (id bigint, value string);
   ```

   当返回信息包含OK时，表示创建表t成功。

2. 查看表信息

```
desc formatted t;
```

3. 查看所有表

```
show tables;
```

返回信息如下所示。

```
OK
t
```

4. 删除表

```
drop table if exists t;
```

当返回信息包含OK时，表示删除表成功。

SQL操作

1. 插入记录

```
insert into table t select 1, 'value-1';
```

当返回信息包含OK时，表示插入信息成功。

```
OK
Time taken: 14.73 seconds
```

2. 查询表中的前10条信息

```
select * from t limit 10;
```

返回信息如下所示。

```
OK
1       value-1
Time taken: 11.48 seconds, Fetched: 1 row(s)
```

3. 聚合操作

```
select value, count(id) from t group by value;
```

返回信息如下所示。

```
OK
value-1 1
Time taken: 20.11 seconds, Fetched: 1 row(s)
```

5.1.1.3.2. Hive连接方式

本文为您介绍在E-MapReduce集群提交Hive SQL的两种方式。

前提条件

已创建集群，详情请参见创建集群。

注意事项

本文示例中需替换的参数：

- `<主节点的节点名称>`：您可以在EMR控制台目标集群的**节点管理**页面获取，具体操作步骤请参见获取主节点公网IP地址和节点名称。
- `cluster-xxx@EMR.xxx.COM` 中的 `xxx`：您可以在主节点上通过命令 `hostname` 获取，获取到的数字即为 `xxx`。

方式一：通过Hive客户端

- 普通集群，提交方式如下所示。

```
hive
```

- 当集群有高安全选项时，提交方式如下所示。
  i. 执行如下命令进行认证。

```
kinit -kt /etc/ecm/hive-conf/hive.keytab hive/<主节点的节点名称>.cluster-xxx@EMR.xxx.COM
```

您也可以通过用户管理功能创建用户，在连接Beeline前使用 `kinit 用户名` 并输入密码，即可通过新建用户使用Beeline客户端。创建用户详情请参见管理用户。

a. 执行如下命令进行认证。

```
kinit 用户名
```

b. 根据提示输入用户的密码。

ii. 连接Hive。

```
hive
```

方式二：通过Beeline

- 普通集群，提交方式如下所示。

```
beeline -u jdbc:hive2://<主节点的节点名称>:10000
```

根据您的集群类型，对应命令如下：

- DataLake集群

```
beeline -u jdbc:hive2://master-1-1:10000
```

- Hadoop集群

```
beeline -u jdbc:hive2://emr-header-1:10000
```

- 当集群有高可用选项时，提交方式如下所示。

根据您集群的类型，对应命令如下：

- DataLake集群

```
beeline -u 'jdbc:hive2://master-1-1:2181,master-1-2:2181,master-1-3:2181/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=hiveserver2'
```

- Hadoop集群

```
beeline -u 'jdbc:hive2://emr-header-1:2181,emr-header-2:2181,emr-header-3:2181/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=hiveserver2'
```

- 当集群有高安全选项时，通过以下步骤提交。

i. 执行如下命令进行认证。

```
kinit -kt /etc/ecm/hive-conf/hive.keytab hive/<主节点的节点名称>.cluster-xxx@EMR.xxx.COM
```

您也可以通过用户管理功能创建用户，在连接Beeline前使用 `kinit 用户名` 并输入密码，即可通过新建用户使用Beeline客户端。创建用户详情请参见管理用户。

a. 执行如下命令进行认证。

```
kinit 用户名
```

b. 根据提示输入用户的密码。

ii. 连接Beeline。

```
beeline -u "jdbc:hive2://<主节点的节点名称>:10000/;principal=hive/<主节点的节点名称>.cluster-xxx@EMR.xxx.COM"
```

> **说明** `jdbc` 链接串需要用双引号括起来。

5.1.1.4. 高阶使用

5.1.1.4.1. Hive作业调优

您可以通过调整内存、CPU和Task个数等，实现对Hive作业的调优。本文为您介绍如何调优Hive作业。

作业调优方案

作业调优方向	调优方案

作业调优方向	调优方案
参数调优	• 内存参数 • CPU参数 • Task数量优化 • 并行运行 • Fetch task • 开启向量化 • 合并小文件
代码优化	代码优化

代码优化

- 数据清洗
  i. 读取表时分区过滤，避免全表扫描。
  ii. 数据过滤之后再JOIN。
  iii. 重复使用数据时，避免重复计算，构建中间表，重复使用中间表。
- 多Distinct优化
  - 优化前代码

    多个Distinct时，数据会出现膨胀。

```
select k,count(distinct case when a > 1 then user_id) user1,
       count(distinct case when a > 2 then user_id) user2,
       count(distinct case when a > 3 then user_id) user3,
       count(distinct case when a > 4 then user_id) user4
from t
group by k
```

  - 优化后代码

    通过两次Group By的方式代替Distinct操作，通过内层的Group By去重并降低数据量，通过外层的Group By取sum，即可实现Distinct的效果。

```
select k,sum(case when user1 > 0 then 1 end) as user1,
       sum(case when user2 > 0 then 1 end) as user2,
       sum(case when user3 > 0 then 1 end) as user3,
       sum(case when user4 > 0 then 1 end) as user4
from
        (select k,user_id,count(case when a > 1 then user_id) user1,
                count(case when a > 2 then user_id) user2,
                count(case when a > 3 then user_id) user3,
                count(case when a > 4 then user_id) user4
         from t
         group by k,user_id
         ) tmp
group by k
```

- 数据倾斜

  热点Key处理方式如下：

  - 如果是Group By出现热点，请按照以下方法操作：

    a. 先开启Map端聚合。

```
set hive.map.aggr=true
hive.groupby.mapaggr.checkinterval=100000（用于设定Map端进行聚合操作的条目数）
```

    b. 可以对Key随机化打散，多次聚合，或者直接设置。

```
set hive.groupby.skewindata=true;
```

    当选项设定为true时，生成的查询计划有两个MapReduce任务。在第一个MapReduce中，Map的输出结果集合会随机分布到Reduce中， 每个部分进行聚合操作，并输出结果。这样处理的结果是，相同的Group By Key有可能分发到不同的Reduce中，从而达到负载均衡的目的；第二个MapReduce任务再根据预处理的数据结果按照Group By Key分布到Reduce中（这个过程可以保证相同的Group By Key分布到同一个Reduce中），最后完成最终的聚合操作。

- 如果两个大表进行JOIN操作时，出现热点，则使用热点Key随机化。

  例如，log表存在大量user_id为null的记录，但是表bmw_users中不会存在user_id为空，则可以把null随机化再关联，这样就避免null值都分发到一个Reduce Task上。代码示例如下。

  ```
  SELECT * FROM log a LEFT OUTER
  JOIN bmw_users b ON
  CASE WHEN a.user_id IS NULL THEN CONCAT('dp_hive',RAND()) ELSE a.user_id=b.user_id END;
  ```

- 如果大表和小表进行JOIN操作时，出现热点，则使用MAP JOIN。

内存参数

您可以通过设置以下参数，对Map和Reduce阶段的内存进行调优：

- Map阶段

参数	描述	示例
mapreduce.map.java.opts	默认参数，表示JVM堆内存。	-Xmx2048m
mapreduce.map.memory.mb	默认参数，表示整个JVM进程占用的内存，计算方法为 **堆内存+堆外内存**=2048+256 。	2304

- Reduce阶段

参数	描述	示例
mapreduce.reduce.java.opts	默认参数，表示JVM堆内存。	-Xmx2048m
mapreduce.reduce.memory.mb	默认参数，表示整个JVM进程占用的内存，计算方法为 **堆内存+堆外内存**=2048+256 。	2304

CPU参数

您可以通过设置以下参数，对Map和Reduce任务的CPU进行调优。

参数	描述
mapreduce.map.cpu.vcores	每个Map任务可用的最多的CPU Core数目。
mapreduce.reduce.cpu.vcores	每个Reduce任务可用的最多的CPU Core数目。 **说明** 此设置在公平队列是不生效的，通常vCores用于较大的集群，以限制不同用户或应用程序的CPU。

Task数量优化

- Map Task数量优化

  在分布式计算系统中，决定Map数量的一个因素就是原始数据，在不加干预的情况下，原始数据有多少个块，就可能有多少个起始的Task，因为每个Task对应读取一个块的数据；当然这个也不是绝对的，当文件数量特别多，并且每个文件的大小特别小时，您就可以限制减少初始Map对相应的Task的数量，以减少计算资源的浪费，如果文件数量较少，但是单个文件较大，您可以增加Map的Task的数量，以减小单个Task的压力。

  通常，Map Task数量是由mapred.map.tasks、mapred.min.split.size和dfs.block.size决定的。

  i. Hive的文件基本上都是存储在HDFS上，而HDFS上的文件，都是分块的，所以具体的Hive数据文件在HDFS上分多少块，可能对应的是默认Hive起始的Task的数量，使用default_mapper_num参数表示。使用数据总大小除以dfs默认的最大块大小来决定初始默认数据分区数。

     初始默认的Map Task数量，具体公式如下。

     ```
     default_mapper_num = total_size/dfs.block.size
     ```

  ii. 计算Split的size，具体公式如下。

     ```
     default_split_size = max(mapred.min.split.size, min(mapred.max.split.size, dfs.block.size))
     ```

     上述公式中的mapred.min.split.size和mapred.max.split.size，分别为Hive计算的时Split的最小值和最大值。

  iii. 将数据按照计算出来的size划分为数据块，具体公式如下。

     ```
     split_num = total_size/default_split_size;
     ```

  iv. 计算的Map Task数量，具体公式如下。

```
map_task_num = min(split_num, max(mapred.map.tasks, default_mapper_num))
```

从上面的过程来看，Task的数量受各个方面的限制，不至于Task的数量太多，也不至于Task的数量太少。如果需要提高Task数量，就要降低mapred.min.split.size的数值，在一定的范围内可以减小default_split_size的数值，从而增加split_num的数量，也可以增大mapred.map.tasks的数量。

> **注意** Hive on TEZ和Hive on MR使用是有差异的。例如，在Hive中执行一个Query时，可以发现Hive的执行引擎在使用Tez与MR时，两者生成的mapper数量差异较大。主要原因在于Tez中对inputSplit做了grouping操作，可以将多个inputSplit组合成更少的groups，然后为每个group生成一个mapper任务，而不是为每个inputSplit生成一个mapper任务。

- Reduce Task数量优化
  - 通过hive.exec.reducers.bytes.per.reducer参数控制单个Reduce处理的字节数。

    Reduce的计算方法如下。

    ```
    reducer_num = min(total_size/hive.exec.reducers.bytes.per.reducers, hive.exec.reducers.max)。
    ```

  - 通过mapred.reduce.tasks参数来设置Reduce Task的数量。

    > **说明** 在TEZ引擎模式下，通过命令 `set hive.tez.auto.reducer.parallelism = true;` ，TEZ将会根据vertice的输出大小动态预估调整Reduce Task的数量。

    同Map一样，启动和初始化Reduce也会消耗时间和资源。另外，有多少个Reduce，就会有多少个输出文件，如果生成了很多个小文件，并且这些小文件作为下一个任务的输入，则会出现小文件过多的问题。

并行运行

并行运行表示同步执行Hive的多个阶段。Hive在执行过程中，将一个查询转化成一个或者多个阶段。某个特定的Job可能包含多个阶段，而这些阶段可能并非完全相互依赖的，也就是可以并行运行的，这样可以使得整个Job的运行时间缩短。

您可以通过设置以下参数，控制不同的作业是否可以同时运行。

参数	描述
hive.exec.parallel	默认值为false。设置true时，表示允许任务并行运行。
hive.exec.parallel.thread.number	默认值为8。表示允许同时运行线程的最大值。

Fetch task

您可以通过设置以下参数，在执行查询等语句时，不执行MapReduce程序，以减少等待时间。

参数	描述
hive.fetch.task.conversion	默认值为none。参数取值如下： • none：关闭Fetch task优化。 在执行语句时，执行MapReduce程序。 • minimal：只在SELECT、FILTER和LIMIT的语句上进行优化。 • more：在minimal的基础上更强大，SELECT不仅仅是查看，还可以单独选择列，FILTER也不再局限于分区字段，同时支持虚拟列（别名）。

开启向量化

您可以通过设置以下参数，在执行查询等语句时，不执行MapReduce程序，以减少等待时间。

参数	描述
hive.vectorized.execution.enabled	默认值为true。开启向量化查询的开关。
hive.vectorized.execution.reduce.enabled	默认值为true。表示是否启用Reduce任务的向量化执行模式。

合并小文件

大量小文件容易在文件存储端造成瓶颈，影响处理效率。对此，您可以通过合并Map和Reduce的结果文件来处理。

您可以通过设置以下参数，合并小文件。

参数	描述
hive.merge.mapfiles	默认值为true。表示是否合并Map输出文件。

参数	描述
hive.merge.mapredfiles	默认值为false。表示是否合并Reduce输出文件。
hive.merge.size.per.task	默认值为256000000，单位字节。表示合并文件的大小。

5.1.1.5. 开发指南

5.1.1.5.1. 自定义函数（UDF）

Hive提供了很多内建函数来满足您的计算需求，您也可以通过创建自定义函数（UDF）来满足不同的计算需求。UDF在使用上与普通的内建函数类似。本文为您介绍自定义函数的开发和使用流程。

背景信息

UDF分类如下表。

UDF分类	描述
UDF（User Defined Scalar Function）	自定义标量函数，通常称为UDF。其输入与输出是一对一的关系，即读入一行数据，写出一条输出值。
UDTF（User Defined Table-valued Function）	自定义表值函数，用来解决一次函数调用输出多行数据场景的，也是唯一一个可以返回多个字段的自定义函数。
UDAF（User Defined Aggregation Function）	自定义聚合函数，其输入与输出是多对一的关系，即将多条输入记录聚合成一条输出值，可以与SQL中的Group By语句联合使用。

开发UDF

1. 使用IDE，创建Maven工程。

   工程基本信息如下，您可以自定义groupId和artifactId。

   ```
   <groupId>org.example</groupId>
   <artifactId>hiveudf</artifactId>
   <version>1.0-SNAPSHOT</version>
   ```

2. 添加pom依赖。

   ```
   <dependency>
         <groupId>org.apache.hive</groupId>
         <artifactId>hive-exec</artifactId>
         <version>2.3.7</version>
         <exclusions>
            <exclusion>
              <groupId>org.pentaho</groupId>
              <artifactId>*</artifactId>
            </exclusion>
         </exclusions>
   </dependency>
   ```

3. 创建一个类，继承Hive UDF类。

   类名您可以自定义，本文示例中类名为MyUDF。

   ```
   package org.example;
   import org.apache.hadoop.hive.ql.exec.UDF;
   /**
    * Hello world!
    *
    */
   public class MyUDF extends UDF
   {
       public String evaluate(final String s) {
           if (s == null) { return null; }
           return s + ":HelloWorld";
       }
   }
   ```

4. 将自定义的代码打成JAR包。

   在*pom.xml*所在目录，执行如下命令制作JAR包。

```
mvn clean package -DskipTests
```

*target*目录下会出现*hiveudf-1.0-SNAPSHOT.jar*的JAR包，即代表完成了UDF开发工作。

使用UDF

1. 使用文件传输工具，上传生成的JAR包至集群root目录。

   本示例使用了文件传输工具（SSH Secure File Transfer Client）。
2. 上传JAR包至HDFS。
   i. 通过SSH方式登录集群，详情请参见登录集群。
   ii. 执行以下命令，上传JAR包到HDFS。

   ```
   hadoop fs -put hiveudf-1.0-SNAPSHOT.jar /user/hive/warehouse/
   ```

   您可以通过 `hadoop fs -ls /user/hive/warehouse/` 命令，查看是否上传成功。待返回信息如下所示表示上传成功。

   ```
   Found 1 items
   -rw-r--r--   1 xx xx 2668 2021-06-09 14:13 /user/hive/warehouse/hiveudf-1.0-SNAPSHOT.jar
   ```
3. 创建UDF函数。
   i. 执行以下命令，进入Hive命令行。

   ```
   hive
   ```

   ii. 执行以下命令，应用生成的JAR包创建函数。

   ```
   create function myfunc as "org.example.MyUDF" using jar "hdfs:///user/hive/warehouse/hiveudf-1.0-SNAPSHOT.jar";
   ```

   代码中的 `myfunc` 是UDF函数的名称，`org.example.MyUDF` 是开发UDF中创建的类，`dfs:///user/hive/warehouse/hiveudf-1.0-SNAPSHOT.jar` 为上传JAR包到HDFS的路径。

   当出现以下信息时，表示创建成功。

   ```
   Added [/private/var/folders/2s/wzzsgpn13rn8rl_0fc4xxkc00000gp/T/40608d4a-a0e1-4bf5-92e8-b875fa6a1e53_resources/hiveudf-1.0-SNAPSHOT.jar] to class path
   Added resources: [hdfs:///user/hive/warehouse/myfunc/hiveudf-1.0-SNAPSHOT.jar]
   ```

   > **说明** 您也可以通过命令 `SHOW FUNCTIONS LIKE '*myfunc* '`，验证函数是否创建成功。
4. 执行以下命令，使用UDF函数。

   该函数与内置函数使用方式一样，直接使用函数名称即可访问。

   ```
   select myfunc("abc");
   ```

   返回如下信息。

   ```
   OK
   abc:HelloWorld
   ```

相关文档

- UDF
- UDAF
- UDTF

5.1.1.6. 最佳实践

5.1.1.6.1. Hive访问EMR HBase数据

Hive支持通过内表或外表两种方式访问E-MapReduce（简称EMR）HBase数据。本文通过示例为您介绍，如何使用EMR上的Hive处理EMR HBase数据。

前提条件

- 已创建Hadoop集群，并且选择了HBase和Zookeeper服务，详情请参见创建集群。
- 已登录集群，详情请参见登录集群。

Hive通过内表访问HBase

如果HBase中没有已经创建好的表，则可以在Hive中创建表，Hive会自动把表结构和数据写入到HBase中。本示例是在Hive中新建表访问HBase。

1. 执行以下命令，进入Hive命令行。

```
hive
```

2. 在Hive中创建并查询表数据。
   i. 执行以下命令，在Hive中创建HBase表。

```
create table hive_hbase_table(key int, value string)
stored by 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
with serdeproperties("hbase.columns.mapping" = ":key,cf1:val")
tblproperties("hbase.table.name" = "hive_hbase_table", "hbase.mapred.output.outputtable" = "hive_hbase_table");
```

   > **说明** 表的存储方式是HBaseStorageHandler，可以存储和读取HBase数据。

   ii. 执行以下命令，向表中插入数据。

```
insert into hive_hbase_table values(212,'bab');
```

   iii. 执行以下命令，查看表数据。

```
select * from hive_hbase_table;
```

   返回信息如下。

```
OK
212 bab
Time taken: 0.337 seconds, Fetched: 1 row(s)
```

3. 退出Hive命令行后，执行以下命令，进入HBase命令行。

```
hbase shell
```

4. 执行以下命令，查看是否已经通过Hive在HBase中创建了表。

```
describe 'hive_hbase_table'
```

   返回信息如下。

```
hbase(main):001:0> describe 'hive_hbase_table'
Table hive_hbase_table is ENABLED
hive_hbase_table
COLUMN FAMILIES DESCRIPTION
{NAME => 'cf1', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEE
65536', REPLICATION_SCOPE => '0'}
1 row(s) in 0.2320 seconds
```

   > **说明** 查看表已存在，说明Hive已经在HBase中创建了表。

5. 执行以下命令，在HBase中查看Hive写的数据是否已存在。

```
scan 'hive_hbase_table'
```

   返回信息如下。

```
ROW                                     COLUMN+CELL

 212                                    column=cf1:val, timestamp=1624513121062, value=bab

1 row(s) in 0.2320 seconds
```

   > **说明** 查看数据已存在，并且与在Hive中插入的数据一致，说明Hive已经成功访问了HBase的数据。

Hive通过外表访问HBase

如果已经在HBase中创建了表，想通过Hive访问，则可以使用Hive外表的方式与HBase中的表建立映射关系，进而通过Hive访问HBase中已经存在的表。

1. 退出Hive命令行后，执行以下命令，进入HBase命令行。

```
hbase shell
```

2. 在HBase中创建并查询表数据。

i. 执行以下命令，在HBase中创建表。

```
create 'hbase_table','f'
```

ii. 执行以下命令，向表中插入数据。

```
put 'hbase_table','1122','f:col1','hello'
```

iii. 执行以下命令，查看表数据。

```
scan 'hbase_table'
```

返回信息如下。

```
ROW                                                 COLUMN+CELL
 1122                                               column=f:col1, timestamp=1627027165760, value=hello
1 row(s) in 0.0170 seconds
```

3. 退出HBase命令行后，执行以下命令，进入Hive命令行。

```
hive
```

4. 执行以下命令，在Hive中创建外表，并与HBase中的表建立映射关系。

```
create external table hbase_table(key int,col1 string,col2 string)
stored by 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
with serdeproperties("hbase.columns.mapping" = "f:col1,f:col2")
tblproperties("hbase.table.name" = "hbase_table", "hbase.mapred.output.outputtable" = "hbase_table");
```

5. 执行以下命令，在Hive中查看hbase_table数据。

```
select * from hbase_table;
```

返回信息如下。

```
OK
1122  hello NULL
Time taken: 2.201 seconds, Fetched: 1 row(s)
```

> **说明** 查看数据已存在，并且与在HBase中插入的数据一致，说明Hive已经成功访问了HBase的数据。

5.1.1.6.2. Hive访问EMR Phoenix数据

Hive支持通过内表或外表两种方式访问E-MapReduce（简称EMR）中的Phoenix数据。本文通过示例为您介绍如何使用EMR上的Hive处理EMR Phoenix数据。

前提条件

- 已创建Hadoop集群，并且选择了HBase、Zookeeper和Phoenix服务，详情请参见创建集群。

> **说明** 因为当前EMR-4.x和EMR-5.x系列版本未支持Phoenix服务，所以此文档仅适用于EMR-3.x系列版本。

- 已登录集群，详情请参见登录集群。

Hive通过内表访问Phoenix

如果Phoenix中没有已经创建好的表，则可以在Hive中创建表，存储到Phoenix中。本示例是在Hive中新建表访问Phoenix。

1. 执行以下命令，进入Hive命令行。

```
hive
```

2. 在Hive中创建并查询表数据。

i. 执行以下命令，在Hive中创建Phoenix表。

```
create table phoenix_hive_create_internal(s1 string,i1 int,f1 float,d1 double)
stored by 'org.apache.phoenix.hive.PhoenixStorageHandler'
tblproperties(
"phoenix.table.name" = "phoenix_hive_create_internal",
"phoenix.rowkeys" = "s1,i1",
"phoenix.column.mapping" = "s1:s1,i1:i1,f1:f1,d1:d1",
"phoenix.table.options" = "SALT_BUCKETS=10,DATA_BLOCK_ENCODING='DIFF'"
);
```

> **说明** 表的存储方式是PhoenixStorageHandler，可以存储和读取Phoenix数据。

ii. 执行以下命令，向表中插入数据。

```
insert into phoenix_hive_create_internal values('wyk',1,2.3412,3.14);
```

iii. 执行以下命令，查看表数据。

```
select * from phoenix_hive_create_internal;
```

返回信息如下。

```
OK
wyk     1       2.3412  3.14
Time taken: 0.569 seconds, Fetched: 1 row(s)
```

3. 进入Phoenix命令行。

退出Hive命令行后，执行以下命令，进入Phoenix命令行。

i. 执行以下命令，进入*phoenix-current*目录。

```
cd /usr/lib/phoenix-current/
```

ii. 执行以下命令，进入Phoenix命令行。

```
python ./bin/sqlline.py
```

4. 执行以下命令，在Phoenix中查看Hive写的数据是否已存在。

```
select * from phoenix_hive_create_internal;
```

返回信息如下。

```
+------+-----+---------+-------+
|  s1  | i1  |   f1    |  d1   |
+------+-----+---------+-------+
| wyk  | 1   | 2.3412  | 3.14  |
+------+-----+---------+-------+
```

> **说明** 查看数据已存在，并且与在Hive中插入的数据一致，说明Hive已经成功访问了Phoenix的数据。

Hive通过外表访问Phoenix

如果已经在Phoenix中创建了表*phoenix_hive_create_internal*，想通过Hive访问，则可以使用Hive外表的方式与Phoenix中的表建立映射关系，进而通过Hive访问Phoenix中已经存在的表。

1. 执行以下命令，进入Hive命令行。

```
hive
```

2. 执行以下命令，在Hive中创建外表，建立与Phoenix表的映射关系。

```
create external table ext_table(
  s1 string,
  i1 int,
  f1 float,
  d1 double
)
stored by 'org.apache.phoenix.hive.PhoenixStorageHandler'
tblproperties(
  "phoenix.table.name" = "phoenix_hive_create_internal",
  "phoenix.rowkeys" = "s1, i1",
  "phoenix.column.mapping" = "s1:s1, i1:i1, f1:f1, d1:d1"
);
```

3. 执行以下命令，在Hive中查看Phoenix表的数据。

```
select * from ext_table;
```

如果可以正常查询数据，说明Hive已经成功访问了Phoenix的数据。

相关文档

- Phoenix相关的介绍，请参见Phoenix。
- Phoenix接入Hive的内容，请参见Phoenix Storage Handler for Apache Hive。

5.1.1.6.3. Hive访问Delta Lake和Hudi数据

Hive不支持写入数据到Delta Lake和Hudi，但是可以通过外部表的方式查询Delta Lake和Hudi中的数据。本文通过示例为您介绍如何使用EMR上的Hive访问Delta Lake和Hudi数据。

前提条件

- 已创建DataLake或Hadoop集群，详情请参见创建集群。

  > **说明** 如果是DataLake集群，需选择Hive、Delta Lake和Hudi服务。

- 已登录集群，详情请参见登录集群。

使用限制

EMR-3.36.0及后续版本和EMR-5.2.0及后续版本，支持Hive对Hudi进行读操作。

Hive访问Delta Lake数据

1. 执行以下命令，进入Spark命令行。

```
spark-sql
```

2. 在Spark中创建并查询表数据。
   i. 执行以下命令，在Spark中创建Delta表。

```
create table delta_table (id int) using delta location "/tmp/delta_table";
```

   ii. 执行以下命令，向表中插入数据。

```
insert into delta_table values 0,1,2,3,4;
```

   iii. 执行以下命令，查看表数据。

```
select * from delta_table;
```

   返回包含如下的信息。

```
2
3
4
0
1
Time taken: 1.847 seconds, Fetched 5 row(s)
```

3. 在Hive中查看Delta Lake数据。
   i. 执行以下命令，进入Hive命令行。

```
hive
```

ii. 执行以下命令，在Hive中查看Delta Lake表。

```
desc formatted delta_table;
```

iii. 执行以下命令，在Hive中查看Delta Lake表的数据。

```
select * from delta_table;
```

返回如下信息。

```
OK
2
3
4
0
1
Time taken: 1.897 seconds, Fetched: 5 row(s)
```

> **说明** 查看数据与在Spark中插入的数据一致，说明Hive已经成功访问了Delta Lake的数据。

Hive访问Hudi数据

> **注意** EMR-3.36.0及后续版本和EMR-5.2.0及后续版本，支持Hive对Hudi进行读操作。

1. 执行以下命令，进入Spark命令行。

```
spark-sql --conf 'spark.serializer=org.apache.spark.serializer.KryoSerializer' \
--conf 'spark.sql.extensions=org.apache.spark.sql.hudi.HoodieSparkSessionExtension'
```

2. 在Spark中创建并查询表数据。
   i. 执行以下命令，在Spark中创建Hudi表。

```
create table h0 (
              id bigint,
              name string,
              price double
            ) using hudi;
```

您可以执行以下命令，查看表结构属性。

```
desc formatted h0;
```

返回信息中会多出以下字段，这些字段是Hudi的默认字段，会同步到元数据中。

```
_hoodie_commit_time string  NULL
_hoodie_commit_seqno  string  NULL
_hoodie_record_key  string  NULL
_hoodie_partition_path  string  NULL
_hoodie_file_name string  NULL
```

   ii. 执行以下命令，向表中插入数据。

```
insert into h0 select 1, 'a1', 10;
```

   iii. 执行以下命令，查询表数据。

```
select * from h0;
```

返回如下信息。

```
OK
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
20210728180336  20210728180336_0_1      74641e17-75d6-4243-be71-a5aa98b5c1af           49062904-52da-402c-82c1-84c
04d3c2a4c-0_0-6-6_20210728180336.parquet     1       a1      10.0
Time taken: 2.001 seconds, Fetched: 1 row(s)
```

3. 在Hive中查看Hudi数据。
   i. 退出Spark命令后，执行以下命令，进入Hive命令行。

```
hive
```

ii. 执行以下命令，查询表数据。

```
select * from h0;
```

返回如下信息。

```
OK
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
20210728180336  20210728180336_0_1      74641e17-75d6-4243-be71-a5aa98b5c1af            49062904-52da-402c-82c1-84c
04d3c2a4c-0_0-6-6_20210728180336.parquet     1       a1      10.0
Time taken: 2.001 seconds, Fetched: 1 row(s)
```

> **说明** 查看的数据与在Spark中插入的数据一致，说明Hive已经成功访问了Hudi的数据。如果在Hive中不想看到Hudi的默认系统字段，可以在Hive中创建一个外表指向Hudi的目录。

您也可以在Hive中查看Hudi表结构的信息。

```
desc formatted h0;
```

5.1.1.6.4. 在EMR集群运行TPC-DS Benchmark

TPC-DS是大数据领域最为知名的Benchmark标准。阿里云E-MapReduce多次刷新TPC-DS官方最好成绩，并且是唯一一个通过认证的可运行TPC-DS 100 TB的大数据系统。本文介绍如何在EMR集群完整运行TPC-DS的99个SQL，并得到最佳的性能体验。

背景信息

TPC-DS是全球最知名的数据管理系统评测基准标准化组织TPC（事务性管理委员会）制定的标准规范，并由TPC管理测试结果的发布。TPC-DS官方工具只包含SQL生成器以及单机版数据生成工具，并不适合大数据场景，所以本文教程中使用的工具和集群信息如下：

- Hive TPC-DS Benchmark测试工具。

  该工具是业界最常用的测试工具，是由Hortonworks公司开发，支持使用Hive和Spark运行TPC-DS以及TPC-H等Benchmark。

- EMR集群版本为EMR-4.8.0。

  Hive TPC-DS Benchmark测试工具是基于Hortonworks HDP 3版本开发的，对应的Hive版本是3.1。本文教程使用的是EMR-4.8.0版本，EMR-4.8.0及之后的版本、EMR-5.1.0以及之后的版本均可运行该教程。

使用限制

EMR-4.8.0及之后的版本、EMR-5.1.0以及之后的版本均可运行该教程。

注意事项

本文示例使用的是Hadoop集群，所以主节点名称为emr-header-1。如果您使用的是其他类型的集群，请修改节点名称，具体请参见获取主节点公网IP地址和节点名称。

集群类型不同，对应的主节点名称不同：

- DataLake集群：master-1-1。
- Hadoop集群：emr-header-1。

步骤一：创建EMR集群和下载TPC-DS Benchmark工具

1. 创建EMR-4.8.0集群，具体操作步骤，请参见创建集群。

   在创建集群时，请关注如下配置信息：

   - **集群类型**：选择**Hadoop**。
   - **实例规格**：如果想获得最佳性能，Core实例推荐使用大数据型或本地SSD。如果想能用小规模数据快速完成所有流程，Core实例也可以选择4 vCPU 16 GiB规格的通用型实例。

     > **注意** 根据您选择运行的数据集确定集群规模，确保Core实例的数据盘总容量大于数据集规模的三倍。数据集相关信息，请参见步骤三：生成并加载数据。

   - **元数据选择**：推荐使用**数据湖元数据**。
   - **挂载公网**：开启。
2. 通过SSH方式连接集群，具体操作请参见登录集群。
3. 安装Git和Maven。

   ```
   sudo yum install -y git maven
   ```

4. 下载TPC-DS Benchmark工具。
   - 通过Git下载。

     ```
     git clone https://github.com/hortonworks/hive-testbench.git
     ```

     > **注意** 中国内地（大陆）地域的节点访问GitHub较慢，如果下载失败，可直接本地下载，将ZIP文件上传到EMR集群，并解压缩。

   - 本地下载，将ZIP文件上传到EMR集群，并解压缩。

     具体操作步骤如下：

     a. 下载hive-testbench-hdp3.zip文件。

     b. 修改下载的*hive-testbench-hdp3.zip*文件名称为*hive-testbench.zip*。

     c. 上传ZIP文件到EMR集群。本步骤以本地操作系统为Linux为例介绍，操作命令如下：

     ```
     scp hive-testbench.zip root@xx.xx.xx.xx:/root/
     ```

     > **说明** xx.xx.xx.xx为EMR集群公网IP地址，获取方式请参见获取主节点公网IP地址和节点名称。

     d. 在EMR集群解压缩上传的ZIP文件。

     ```
     unzip hive-testbench.zip
     ```

步骤二：编译并打包数据生成器

1. （可选）配置阿里云镜像。

   在中国内地（大陆）地区加速Maven编译，可以使用阿里云镜像。使用阿里云镜像，编译并打包数据生成器的耗时为2min~3min。

   i. 执行如下命令，新建文件目录。

   ```
   mkdir -p ~/.m2/
   ```

   ii. 执行如下命令，将Maven配置文件拷贝到新文件目录下。

   ```
   cp /usr/share/maven/conf/settings.xml ~/.m2/
   ```

   iii. 在*~/.m2/settings.xml*文件中添加镜像信息，具体内容如下：

   ```
   <mirror>
       <id>aliyun</id>
       <mirrorOf>central</mirrorOf>
       <name>Nexus aliyun</name>
       <url>http://maven.aliyun.com/nexus/content/groups/public</url>
   </mirror>
   ```

2. 切换到*hive-testbench*目录。

   ```
   cd hive-testbench
   ```

3. 使用TPC-DS工具集进行编译并打包数据生成器。

   ```
   ./tpcds-build.sh
   ```

步骤三：生成并加载数据

1. 设置数据规模SF（Scale Factor）。

   SF单位相当于GB，所以SF=1相当于1 GB，SF=100相当于100 GB，SF=1000相当于1 TB，以此类推。本步骤示例采用小规模数据集，推荐使用SF=3。具体命令如下：

   ```
   SF=3
   ```

   > **注意** 请确保数据盘总大小是数据集规模的3倍以上，否则后续流程中会出现报错情况。

2. 检查并清理Hive数据库。

   i. 检查Hive数据库是否存在。

   ```
   hive -e "desc database tpcds_bin_partitioned_orc_$SF"
   ```

ii. （可选）清理已经存在的Hive数据库。

> **注意** 如果Hive数据库tpcds_bin_partitioned_orc_$SF已经存在，需要执行下面的命令清理数据库，否则后续流程会报错。如果不存在，则跳过该步骤。

```
hive -e "drop database tpcds_bin_partitioned_orc_$SF cascade"
```

3. 配置Hive服务地址。

   *tpcds-setup.sh*脚本默认配置的Hive服务地址与EMR集群环境不一致，所以需要将脚本中HiveSever的地址替换为EMR集群中的Hive服务地址。具体命令如下：

   ```
   sed -i 's/localhost:2181\/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=hiveserver2?tez.queue.name=default/emr-header-1:10000\//' tpcds-setup.sh
   ```

   脚本默认配置的Hive服务地址为：`jdbc:hive2://localhost:2181/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=hiveserver2?tez.queue.name=default`。通过上述命令替换后的Hive服务地址为：`jdbc:hive2://emr-header-1:10000/`。

4. 生成并加载数据。

   在SF=3时，该步骤耗时为40min~50min。如果运行正常，TPC-DS数据表将会加载到tpcds_bin_partitioned_orc_$SF数据库中。生成的数据可以保存在HDFS和OSS，针对这两种场景具体的操作如下：

   - 使用HDFS存储数据的场景。

     EMR集群默认将TPC-DS Table数据保存在HDFS上。执行如下命令生成并加载数据：

     ```
     ./tpcds-setup.sh $SF
     ```

     ```
     TPC-DS text data generation complete.
     Loading text data into external tables.
     Optimizing table date_dim (1/24).
     Optimizing table time_dim (2/24).
     Optimizing table item (3/24).
     Optimizing table customer (4/24).
     Optimizing table customer_demographics (5/24).
     Optimizing table household_demographics (6/24).
     Optimizing table customer_address (7/24).
     Optimizing table store (8/24).
     Optimizing table promotion (9/24).
     Optimizing table warehouse (10/24).
     Optimizing table ship_mode (11/24).
     Optimizing table reason (12/24).
     Optimizing table income_band (13/24).
     Optimizing table call_center (14/24).
     Optimizing table web_page (15/24).
     Optimizing table catalog_page (16/24).
     Optimizing table web_site (17/24).
     Optimizing table store_sales (18/24).
     Optimizing table store_returns (19/24).
     Optimizing table web_sales (20/24).
     Optimizing table web_returns (21/24).
     Optimizing table catalog_sales (22/24).
     Optimizing table catalog_returns (23/24).
     Optimizing table inventory (24/24).
     Loading constraints
     Data loaded into database tpcds_bin_partitioned_orc_4.
     ```

   - 使用OSS存储数据的场景。

     通过EMR存储和计算分离的架构能力，可以很方便的做到将数据保存在OSS。具体操作步骤如下：

     a. 使用Hive命令修改数据库路径。

     ```
     hive --hivevar SF=$SF
     create database if not exists tpcds_bin_partitioned_orc_${SF};
     alter database tpcds_bin_partitioned_orc_${SF} set location 'oss://<bucket-name>/warehouse/tpcds_bin_partitioned_orc_${SF}.db';
     ```

     <bucket-name>需要改成和EMR集群在同一地域的OSS Bucket名称。

     b. 生成并加载据。

     ```
     cd ~/hive-testbench
     ./tpcds-setup.sh $SF
     ```

     > **说明** 执行以上命令后，生成的数据就直接保存在OSS上了。如果您同时使用了数据湖构建（DLF）来保存Hive表的元数据，数据生成后，您可以随时释放当前的EMR集群，并在同一地域的其他EMR集群上再次查询当前生成的TPC-DS数据集。

5. 获取Hive表统计信息。

推荐使用Hive SQL ANALYZE命令获取Hive表统计信息，可以加快后续SQL的查询速度。此步骤在SF=3时，耗时为20min~30min。

```
hive -f ./ddl-tpcds/bin_partitioned/analyze.sql \
    --hiveconf hive.execution.engine=tez \
    --database tpcds_bin_partitioned_orc_$SF
```

步骤四：运行TPC-DS SQL

本步骤分别介绍如何使用Hive和Spark运行TPC-DS SQL。

- 使用Hive运行TPC-DS SQL。
  i. 通过以下命令执行单SQL。

  TPC-DS SQL共有99个文件都放在*sample-queries-tpcds*工作目录下（包括*query10.sql*和*query11.sql*等文件）。在SF=3时，所有的SQL都可以在5min内返回结果。

  > **注意** 因为TPC-DS Query和数据都是随机生成，所以部分SQL查询返回结果数为0属于正常现象。

  ```
  cd sample-queries-tpcds
  hive --database tpcds_bin_partitioned_orc_$SF
  set hive.execution.engine=tez;
  source query10.sql;
  ```

  ii. 利用工具包中的脚本顺序执行99个完整SQL。具体命令如下：

  ```
  cd ~/hive-testbench
  # 生成一个Hive配置文件，并指定Hive执行引擎为Tez。
  echo 'set hive.execution.engine=tez;' > sample-queries-tpcds/testbench.settings
  ./runSuite.pl tpcds $SF
  ```

  ```
  [root@emr-header-1 hive-testbench]# ./runSuite.pl tpcds $SF
  filename,status,time,rows
  query10.sql,success,42,34
  query11.sql,success,75,100
  query12.sql,success,33,100
  query13.sql,success,45,1
  query14.sql,success,193,100
  query14.sql,success,193,100
  query15.sql,success,37,100
  query16.sql,success,63,1
  query17.sql,success,63,4
  query18.sql,success,56,100
  query19.sql,success,38,100
  query2.sql,success,59,2513
  query20.sql,success,35,100
  query21.sql,success,32,100
  query22.sql,success,88,100
  query23.sql,success,146,1
  query24.sql,success,118,2
  query26.sql,success,45,100
  query27.sql,success,47,1
  query28.sql,success,45,1
  query29.sql,success,69,3
  query3.sql,success,38,39
  query30.sql,success,41,100
  query31.sql,success,55,226
  query32.sql,success,32,1
  query33.sql,success,48,100
  query35.sql,success,59,100
  query36.sql,success,43,100
  query37.sql,success,40,1
  query38.sql,success,62,1
  ```

- 使用Spark运行TPC-DS SQL。

  TPC-DS工具集中包含Spark SQL用例，用例位于*spark-queries-tpcds*目录下，可以使用 `spark-sql` 或者 `spark-beeline` 等命令行工具执行这些SQL。本步骤以Spark Beeline工具连接Spark ThriftServer为例，介绍如何使用Spark运行TPC-DS SQL来查询步骤三：生成并加载数据生成的TPC-DS数据集。

  > **说明** EMR Spark支持HDFS和OSS等多种存储介质保存的数据表，也支持数据湖构建（DLF）元数据。

  i. 使用Spark Beeline ANALYZE命令获得Hive表统计信息，加快后续SQL查询速度。

  ```
  cd ~/hive-testbench
  spark-beeline -u jdbc:hive2://emr-header-1:10001/tpcds_bin_partitioned_orc_$SF \
    -f ./ddl-tpcds/bin_partitioned/analyze.sql
  ```

  ii. 切换到Spark SQL用例所在的文件目录。

```
cd spark-queries-tpcds/
```

iii. 通过以下命令执行单个SQL。

```
spark-beeline -u jdbc:hive2://emr-header-1:10001/tpcds_bin_partitioned_orc_$SF -f q1.sql
```

iv. 通过脚本顺序执行99个SQL。

TPC-DS工具集中没有包含批量执行Spark SQL的脚本，所以本步骤提供一个简单脚本供参考。

```
for q in `ls *.sql`; do
  spark-beeline -u jdbc:hive2://emr-header-1:10001/tpcds_bin_partitioned_orc_$SF -f $q > $q.out
done
```

> 注意
>
> - SQL列表中*q30.sql*文件存在列名c_last_review_date_sk错写为c_last_review_date的情况，所以该SQL运行失败属于正常现象。
> - 通过脚本顺序执行99个Spark SQL的时候，如果出现报错情况，解决方案请参见常见问题。

常见问题

Q：通过脚本顺序执行99个Spark SQL的时候报错，怎么解决？

A：Spark ThriftServer服务的默认内存不适合较大规模数据集测试，如果在测试过程中出现Spark SQL作业提交失败，原因可能是Spark ThriftServer出现OutOfMemory异常。针对这种情况的解决方法为调整Spark服务配置spark_thrift_daemon_memory的值后重启ThriftServer服务。具体操作步骤如下：

1. 进入Spark服务页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   v. 在**集群服务**页面，单击**Spark**服务区域的**配置**。
2. 调整服务配置spark_thrift_daemon_memory的值。
   i. 在**配置搜索**区域，输入spark_thrift_daemon_memory。
   ii. 根据使用的数据集规模调整对应的数值。
   您可以将默认的值调大。
   iii. 单击右上角的**保存**。
   iv. 在**确认修改**对话框中，输入执行原因，单击**确定**。
3. 重启Spark。
   i. 在Spark服务页面，选择右上角**更多操作 > 重启**。
   ii. 在弹出的对话框中，输入**执行原因**，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

5.1.1.7. 常见问题

本文汇总了Hive使用时的常见问题。

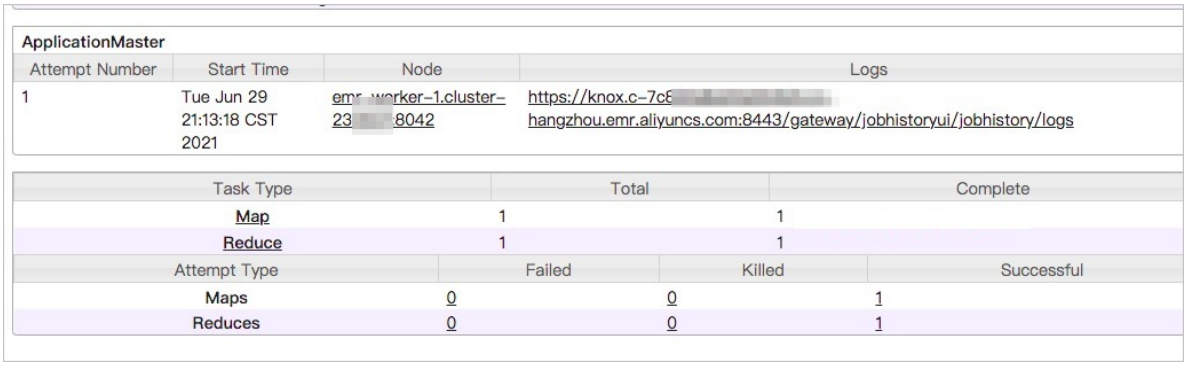
- 作业长时间处于等待状态，如何处理？
- Map端是否读取了小文件？
- Reduce Task任务耗时，是否出现了数据倾斜？

作业长时间处于等待状态，如何处理？

您可以通过以下步骤定位问题：

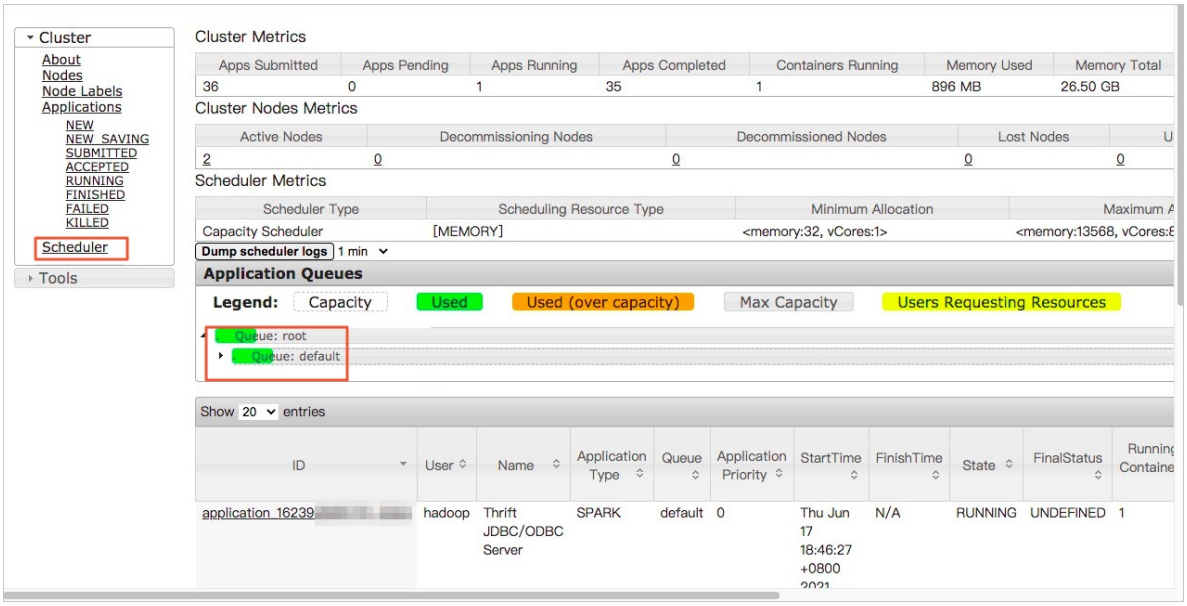
1. 在EMR控制台的**访问链接与端口**页面，单击YARN UI所在行的链接。
2. 单击Application ID。
3. 单击**Tracking URL**的链接。

可以看到有多个作业处于等待状态。



4. 在左侧导航中，单击**Scheduler**。

   即可进入队列，您可以看一下当前队列的繁忙程度，来分析是因为队列中没有空闲资源，还是当前任务确实比较耗时。如果是队列资源紧张，您可以考虑切换到空闲队列，否则需要优化代码。

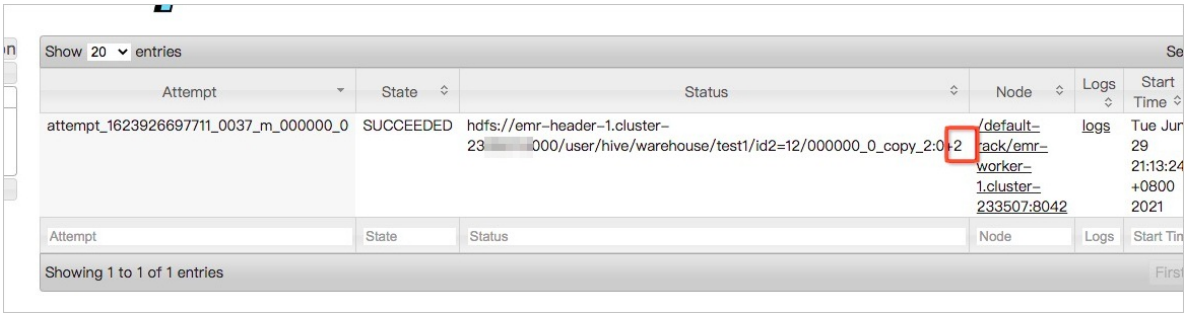


Map端是否读取了小文件？

您可以通过以下步骤定位问题：

1. 在EMR控制台的**访问链接与端口**页面，单击YARN UI所在行的链接。
2. 单击Application ID。

   进入Map Task的详情页面，可以看到每个Map Task读取的数据量，如下图所示，读取的数据量是2个字节记录。如果大部分的Map Task读取的文件量都比较小，就需要考虑小文件合并。

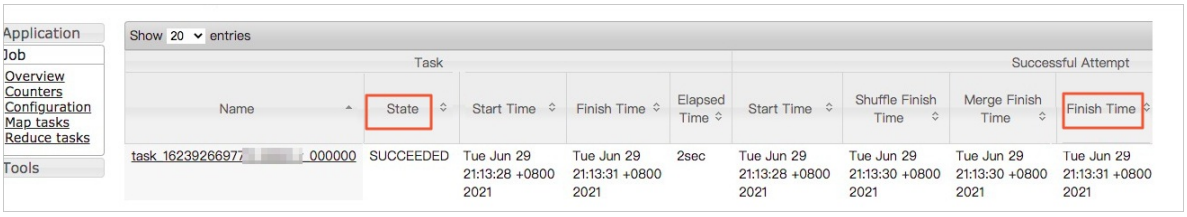


   您也可以通过查看Map Task的Log，获取更多的信息。

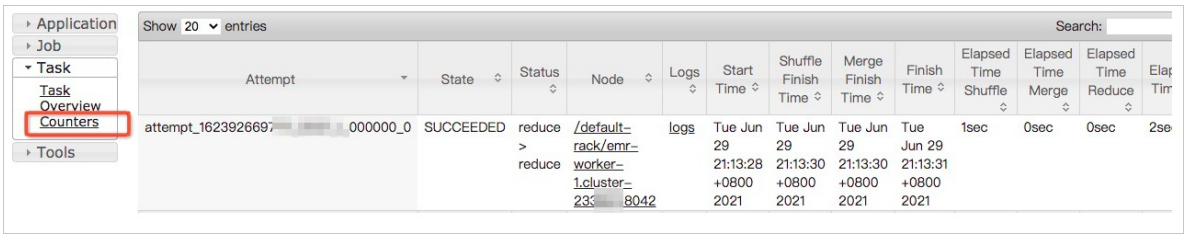
Reduce Task任务耗时，是否出现了数据倾斜？

您可以通过以下步骤定位问题：

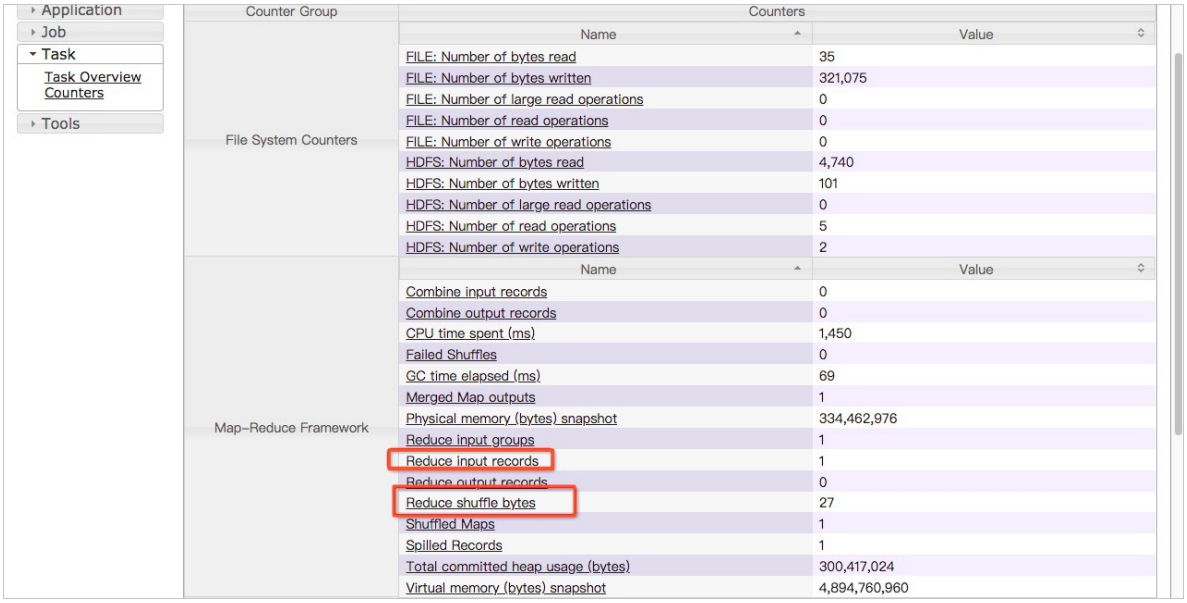
1. 在EMR控制台的**访问链接与端口**页面，单击YARN UI所在行的链接。
2. 单击Application ID。
3. 在Reduce Task列表页面，按照完成时间逆序排序，找出Top耗时的Reduce Task任务。



4. 单击Task的Name链接。
5. 在Task详情页面，单击左侧的Counters。



查看当前Reduce Task中Reduce Input bytes和Reduce shuffle bytes的信息，如果比其他的Task处理的数据量大很多，则说明出现了倾斜问题。



5.1.2. Flink

5.1.2.1. 概述

Flink（VVR）是基于Apache Flink（以下简称Flink）开发的商业版，VVR引擎接口完全兼容Flink开源版本，且提供GeminiStateBackend等高增值功能，以提升作业性能及稳定性。

背景信息

Flink核心是一个流式的数据流执行引擎，其针对数据流的分布式计算提供了数据分布、数据通信以及容错机制等功能。基于流执行引擎，Flink提供了更高抽象层的API以便您编写分布式任务。

Flink（VVR）完全兼容开源Flink，相关内容请参见如下文档：

- DataStream API
- Table API&SQL
- Python API

使用场景

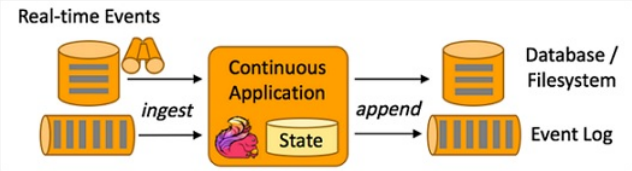
Flink广泛应用于大数据实时化的场景，本文从技术领域和企业应用场景进行介绍。

- 技术领域

  从技术领域的角度，Flink主要用于以下场景：

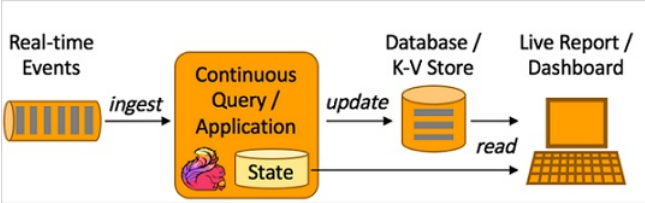
  - 实时ETL（Extract-transform-load）和数据流

    实时ETL和数据流的目的是实时地把数据从A点投递到B点。在投递的过程中可能添加数据清洗和集成的工作，例如实时构建搜索系统的索引和实时数仓中的ETL过程等。

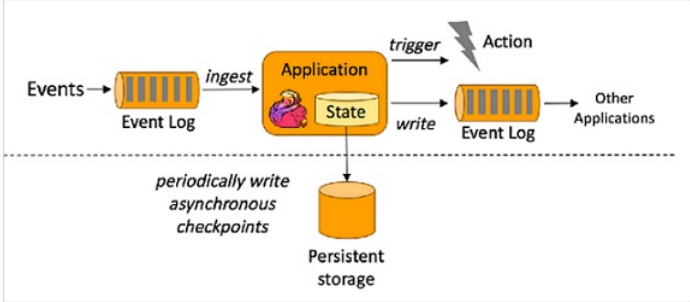
  - 实时数据分析

    实时数据分析指的是根据业务目标，从原始数据中抽取对应信息并整合的过程。例如，查看每天销量前10的商品、仓库平均周转时间、文档平均单击率和推送打开率等。实时数据分析则是上述过程的实时化，通常在终端体现为实时报表或实时大屏。

  - 事件驱动应用

    事件驱动应用是对一系列订阅事件进行处理或作出响应的系统。事件驱动应用通常需要依赖内部状态，例如欺诈检测、风控系统、运维异常检测系统等。当您的行为触发某些风险控制点时，系统会捕获这个事件，并根据您当前和之前的行为进行分析，决定是否对您进行风险控制。

  关于Apache Flink的更多介绍，请参见Apache Flink官网。

- 企业应用

  从企业应用的角度，Flink主要用于以下场景：

  - 业务部门：实时风控、实时推荐和搜索引擎的实时索引构建等。
  - 数据部门：实时数仓、实时报表和实时大屏等。
  - 运维部门：实时监控、实时异常检测和预警以及全链路Debug等。

入门指导

- Flink（VVR）的使用请参见Flink（VVR）作业配置。
- 在Zeppelin中使用Flink的详情请参见Zeppelin概述。

5.1.2.2. 基础使用

本文为您介绍如何在E-MapReduce上提交Flink作业以及查看作业。

背景信息

Dataflow集群中的Flink是以YARN模式部署的，在Dataflow集群中，您可以通过SSH方式登录Flink模式的Dataflow集群，在命令行中进行提交。

在基于YARN模式部署的Dataflow集群支持Session模式、Per-Job Cluster模式和Application模式。

模式	描述	特点
Session模式	Seesion模式会根据您设置的资源参数创建一个Flink集群，所有作业都将被提交到这个集群上运行。该集群在作业运行结束之后不会自动释放。 例如，某个作业发生异常，导致一个Task Manager关闭，则其他所有运行在该Task Manager上的作业都会失败。另外由于同一个集群中只有一个Job Manager，随着作业数量的增多，Job Manager的压力会相应增加。	• 优点：提交作业时，资源分配导致的时间开销相比其他模式较小。 • 缺点：由于所有作业都运行在该集群中，会存在对资源的竞争以及作业间的相互影响。 根据以上特点，该模式适合部署需要较短启动时间且运行时间相对较短的作业。
Per-Job Cluster模式	当使用Per-Job Cluster模式时，每次提交一个Flink作业，YARN都会为这个作业新启动一个Flink集群，然后运行该作业。当作业运行结束或者被取消时，该作业所属的Flink集群也会被释放。	• 优点：作业之间资源隔离，一个作业的异常行为不会影响到其他作业。 因为每个作业都和一个Job Manager一一对应，因此不会出现一个Job Manager因为运行多个Job而导致负载过高的问题。 • 缺点：每次运行一个作业都要启动一个专属Flink集群，启动作业的开销更大。 根据以上特点，该模式通常适合运行时间较长的作业。
Application模式	当使用Application模式时，每次提交一个Flink Application（一个Application包含一个或多个作业），YARN都会为这个Application新启动一个Flink集群。当Application运行结束或者被取消时，该Application所属的Flink集群也会被释放。 该模式与Per-Job模式不同的是，Application对应的JAR包中的main()方法会在集群中的Job Manager中被执行。 如果提交的JAR包中包含多个作业，则这些作业都会在该Application所属的集群中执行。	• 优点：可以减轻客户端提交作业时的负担。 • 缺点：每次运行一个Flink Application都要启动一个专属Flink集群，启动Application的时间开销会更大。

您可以根据需求，选择以下三种模式提交并查看作业：

- Session模式提交并查看作业
- Per-Job Cluster模式提交并查看作业
- Application模式提交并查看作业

前提条件

已创建Flink模式的Dataflow集群，详情请参见创建集群。

Session模式提交并查看作业

1. 通过SSH方式连接集群，详情请参见登录集群。
2. 执行以下命令，启动YARN Session。

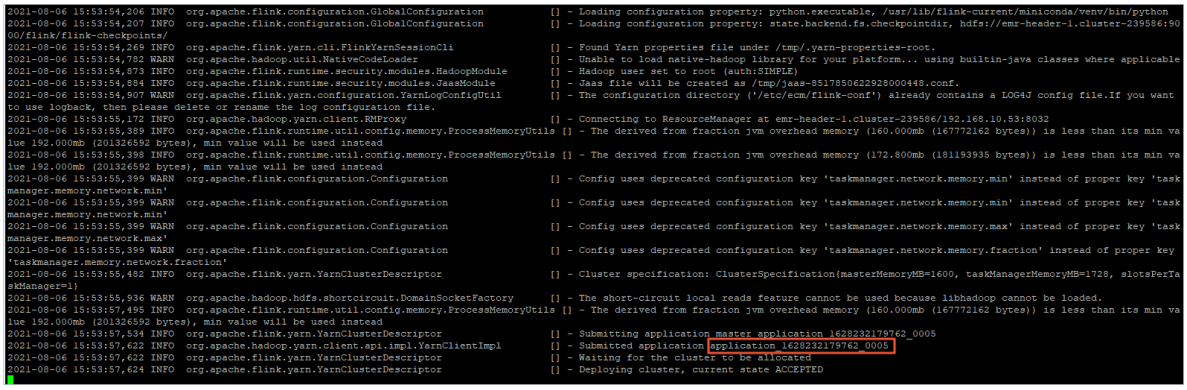
```
yarn-session.sh --detached
```

3. 执行以下命令，提交作业。

```
flink run /opt/apps/FLINK/flink-current/examples/streaming/TopSpeedWindowing.jar
```

> **说明** 本文使用Flink自身提供的TopSpeedWindowing示例进行介绍，该示例是一个会长时间运行的流作业。

提交成功后，会返回已提交的Flink作业的YARN Application ID。返回如下类似信息。



4. 执行以下命令，查看作业状态。

```
flink list -t yarn-session -Dyarn.application.id=<application_XXXX_YY>
```

> **说明**
>
> 您也可以通过Web UI的方式查看作业状态，详情请参见通过Web UI查看作业状态。

Per-Job Cluster模式提交并查看作业

1. 通过SSH方式连接集群，详情请参见登录集群。
2. 执行以下命令，提交作业。

```
flink run -t yarn-per-job --detached /opt/apps/FLINK/flink-current/examples/streaming/TopSpeedWindowing.jar
```

提交成功后，会返回已提交的Flink作业的YARN Application ID。返回如下类似信息。

3. 您可以执行以下命令，查看作业状态。

```
flink list -t yarn-per-job -Dyarn.application.id=<application_XXXX_YY>
```

> **说明** 本文示例中的 `<application_XXXX_YY>` 为作业运行后返回的Application ID。

您也可以通过Web UI的方式查看作业状态，详情请参见通过Web UI查看作业状态。

Application模式提交并查看作业

1. 通过SSH方式连接集群，详情请参见登录集群。
2. 执行以下命令，提交作业。

```
flink run-application -t yarn-application /opt/apps/FLINK/flink-current/examples/streaming/TopSpeedWindowing.jar
```

提交成功后，会返回已提交的Flink作业的YARN Application ID。返回如下类似信息。

```
Waiting for response...
------------------ Running/Restarting Jobs -------------------
06.08.2021 15:29:08 : c03f5afe3bacda7d9d7e82e4e03bc53a : CarTopSpeedWindowingExample (RUNNING)
--------------------------------------------------------------
No scheduled jobs.
[root@emr-header-1 ~]# flink run-application -t yarn-application /usr/lib/flink-current/examples/streaming/TopSpeedWindowing.jar
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/apps/ecm/service/flink/1.12-vvr-3.0.2/package/flink-1.12-vvr-3.0.2/lib/log4j-slf4j-impl-2.12.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/apps/ecm/service/hadoop/2.8.5-1.6.4/package/hadoop-2.8.5-1.6.4/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
2021-08-06 15:50:32,597 INFO  org.apache.flink.yarn.cli.FlinkYarnSessionCli                 [] - Found Yarn properties file under /tmp/.yarn-properties-root.
2021-08-06 15:50:32,997 INFO  org.apache.flink.yarn.cli.FlinkYarnSessionCli                 [] - Found Yarn properties file under /tmp/.yarn-properties-root.
2021-08-06 15:50:33,215 WARN  org.apache.flink.yarn.configuration.YarnLogConfigUtil         [] - The configuration directory ('/etc/ecm/flink-conf') already contains a LOG4J config file.If you want
to use logback, then please delete or rename the log configuration file.
2021-08-06 15:50:33,413 INFO  org.apache.hadoop.yarn.client.RMProxy                         [] - Connecting to ResourceManager at emr-header-1.cluster-239586/192.168.10.53:8032
2021-08-06 15:50:33,550 INFO  org.apache.flink.yarn.YarnClusterDescriptor                   [] - No path for the flink jar passed. Using the location of class org.apache.flink.yarn.YarnClusterDescri
ptor to locate the jar
2021-08-06 15:50:33,659 INFO  org.apache.flink.yarn.YarnClusterDescriptor                   [] - Cluster specification: ClusterSpecification(masterMemoryMB=1600, taskManagerMemoryMB=1728, slotsPerTa
skManager=1)
2021-08-06 15:50:34,026 WARN  org.apache.hadoop.hdfs.shortcircuit.DomainSocketFactory       [] - The short-circuit local reads feature cannot be used because libhadoop cannot be loaded.
2021-08-06 15:50:35,353 INFO  org.apache.flink.yarn.YarnClusterDescriptor                   [] - Submitting application master application_1628232179762_0003
2021-08-06 15:50:35,403 INFO  org.apache.hadoop.yarn.client.api.impl.YarnClientImpl         [] - Submitted application application_1628232179762_0003
2021-08-06 15:50:35,404 INFO  org.apache.flink.yarn.YarnClusterDescriptor                   [] - Waiting for the cluster to be allocated
2021-08-06 15:50:35,405 INFO  org.apache.flink.yarn.YarnClusterDescriptor                   [] - Deploying cluster, current state ACCEPTED
2021-08-06 15:50:40,936 INFO  org.apache.flink.yarn.YarnClusterDescriptor                   [] - YARN application has been deployed successfully.
2021-08-06 15:50:40,936 INFO  org.apache.flink.yarn.YarnClusterDescriptor                   [] - Found Web Interface emr-worker-1.cluster-239586:43135 of application 'application_1628232179762_0003'.
```

3. 执行以下命令，查看作业状态。

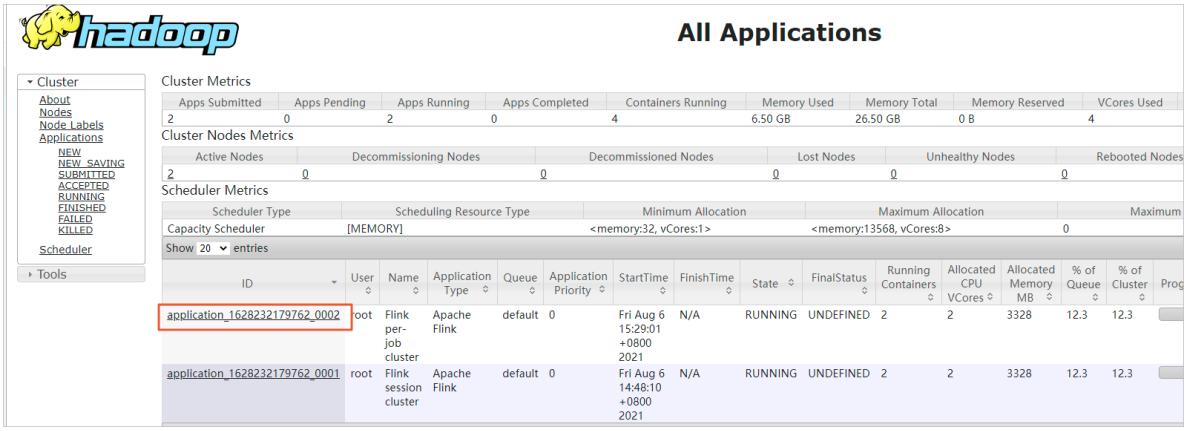
```
flink list -t yarn-application -Dyarn.application.id=<application_XXXX_YY>
```

> 说明

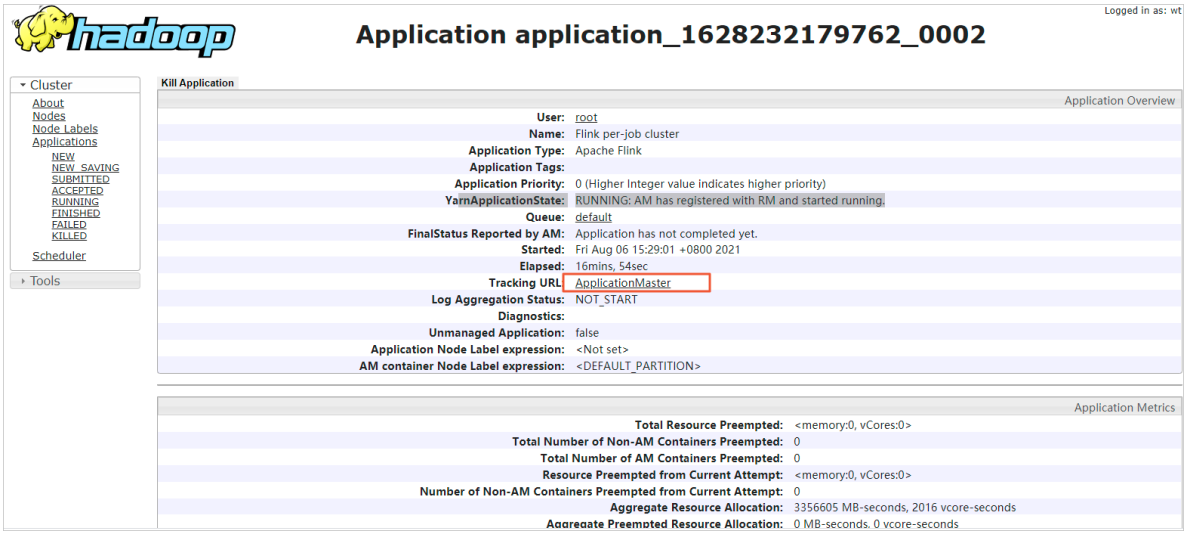
您也可以通过Web UI的方式查看作业状态，详情请参见通过Web UI查看作业状态。

通过Web UI查看作业状态

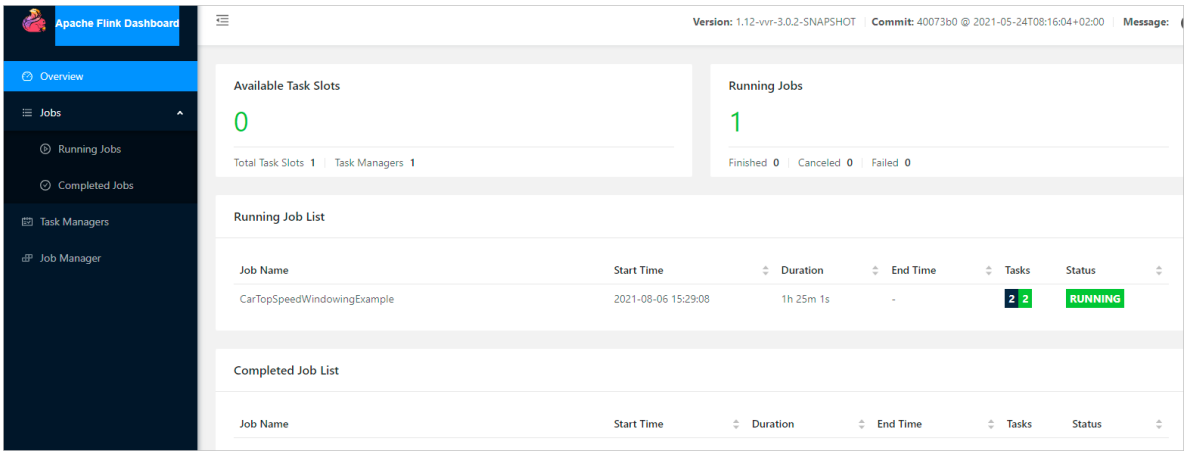
1. 访问Web UI。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群的**集群ID**。
   iv. 单击上方的**访问链接与端口**页签。
   v. 在**访问链接与端口**页面，单击YARN UI所在行的链接。
   访问Web UI的详细信息，请参见访问链接与端口。
2. 单击Application ID。



3. 单击**Tracking URL**的链接。



进入Apache Flink Dashboard页面，即可查看作业的状态。



相关文档

Flink on YARN的更多信息，请参见Apache Hadoop YARN。

5.1.2.3. 开发指南

5.1.2.3.1. Flink SQL参考

Flink SQL是为了简化计算模型、降低您使用Flink门槛而设计的一套符合标准SQL语义的开发语言。

本文通过以下方面，为您介绍Flink SQL的使用方法。

操作	文档
关键字	关键字
创建数据视图	创建数据视图

操作		文档
DDL数据定义语句	创建数据源表	• 日志服务SLS源表 • 消息队列Kafka源表 • 数据总线DataHub源表 • 全量MaxCompute源表 • 增量MaxCompute源表 • 消息队列RocketMQ版源表 • Hologres源表 • 全量Elasticsearch源表 • Postgres的CDC源表（公测中） • MySQL的CDC源表 • Upsert Kafka源表 • Datagen源表
	创建数据结果表	• 日志服务SLS结果表 • 消息队列Kafka结果表 • 数据总线DataHub结果表 • 表格存储Tablestore结果表 • MaxCompute结果表 • 消息队列RocketMQ版结果表 • 云数据库HBase版结果表 • 云数据库Redis版结果表 • Hologres结果表 • 云数据库MongoDB版结果表 • Elasticsearch结果表 • Phoenix5结果表 • 云数据库RDS MySQL结果表 • 云原生数据仓库AnalyticDB MySQL版3.0结果表 • ClickHouse结果表 • print结果表 • Upsert Kafka结果表 • Blackhole结果表 • InfluxDB结果表
	创建数据维表	• 云数据库HBase版维表 • 表格存储Tablestore维表 • MaxCompute维表 • 云数据库Redis版维表 • Elasticsearch维表 • Hologres维表 • 云数据库RDS MySQL维表 • 云原生数据仓库AnalyticDB MySQL版3.0维表 • FileSystem维表
DML数据操作语句（INSERT INTO）		INSERT INTO语句
DQL数据查询语句		DQL数据查询语句
窗口函数		• 概述 • 滚动窗口 • 滑动窗口 • 会话窗口 • OVER窗口

操作		文档
内置函数		• 概述 • 标量函数 • 表值函数 • 聚合函数 • 机器学习函数CLUSTER_SERVING
自定义函数	Java	• 概述 • 自定义标量函数（UDF） • 自定义聚合函数（UDAF） • 自定义表值函数（UDTF）
	Python	• 概述 • 自定义标量函数（UDF） • 自定义聚合函数（UDAF） • 自定义表值函数（UDTF）

5.1.2.3.2. Flink DataStream参考

本文通过以下方面，为您介绍Flink DataStream 的使用方法。

DataFlow集群的Flink DataStream API完全兼容开源的Flink版本，关于Flink DataStream API的详细信息，请参见Flink DataStream API Programming Guide。

上下游存储（Connector）

- 开源Flink的上下游存储，请参见DataStream Connectors。
- DataFlow集群中新增支持的上下游存储，请参见下表。

Connector版本	EMR版本	Connector类型	文档及Demo
1.13-vvr-4.0.10及以上	EMR-3.38.0及以上	ververica-connector-datahub	◦ 文档：DataHub DataStream Connector ◦ Demo：ververica-connector-datahub-demo
		ververica-connector-kafka	◦ 文档：Kafka DataStream Connector ◦ Demo：ververica-connector-kafka-demo
		◦ ververica-connector-odps ◦ ververica-connector-continuous-odps	◦ 文档：MaxCompute DataStream Connector ◦ Demo：ververica-connector-maxcompute-demo
		ververica-connector-mysql-cdc	文档：MySQL CDC DataStream Connector

5.1.2.3.3. Flink Python参考

本文通过以下方面，为您介绍Flink Python的使用方法。

背景信息

DataFlow集群的Flink Python API完全兼容开源的Flink版本，关于Flink Python API的详细信息，请参见Python API。

使用Python依赖

通过以下场景为您介绍如何使用Python依赖：

- 使用自定义的Python虚拟环境
- 使用第三方Python包
- 使用JAR包
- 使用数据文件

使用自定义的Python虚拟环境

- 方式一：在DataFlow集群中的某个节点创建Python虚拟环境

i. 在DataFlow集群的某个节点，准备*setup-pyflink-virtual-env.sh*脚本，其内容如下。

```
set -e
# 创建Python的虚拟环境。
python3.6 -m venv venv
# 激活Python虚拟环境。
source venv/bin/activate
# 准备Python虚拟环境。
pip install --upgrade pip
# 安装PyFlink依赖。
pip install "apache-flink==1.13.0"
# 退出Python虚拟环境。
deactivate
```

ii. 执行以下命令，运行该脚本。

```
./setup-pyflink-virtual-env.sh
```

该命令执行完成后，会生成一个名为*venv*的目录，即为Python 3.6的虚拟环境。您也可以修改上述脚本，安装其他版本的Python虚拟环境。

为了使用该Python虚拟环境，您可以选择将该Python虚拟环境分发到DataFlow集群的所有节点上，也可以在提交PyFlink作业的时候，指定使用该Python虚拟环境，详细信息请参见Command-Line Interface。

- 方式二：在本地开发机创建Python虚拟环境

i. 在本地准备*setup-pyflink-virtual-env.sh*脚本，其内容如下。

```
set -e
# 下载Python 3.7 miniconda.sh脚本。
wget "https://repo.continuum.io/miniconda/Miniconda3-py37_4.9.2-Linux-x86_64.sh" -O "miniconda.sh"
# 为Python 3.7 miniconda.sh脚本添加执行权限。
chmod +x miniconda.sh
# 创建Python的虚拟环境。
./miniconda.sh -b -p venv
# 激活Conda Python虚拟环境。
source venv/bin/activate ""
# 安装PyFlink依赖。
pip install "apache-flink==1.13.0"
# 退出Conda Python虚拟环境。
conda deactivate
# 删除缓存的包。
rm -rf venv/pkgs
# 将准备好的Conda Python虚拟环境打包。
zip -r venv.zip venv
```

ii. 在本地准备*build.sh*脚本，其内容如下。

```
#!/bin/bash
set -e -x
yum install -y zip wget
cd /root/
bash /build/setup-pyflink-virtual-env.sh
mv venv.zip /build/
```

iii. 在CMD命令行中，执行如下命令。

```
docker run -it --rm -v $PWD:/build  -w /build quay.io/pypa/manylinux2014_x86_64 ./build.sh
```

该命令执行完成后，会生成一个名为*venv.zip*的文件，即为Python 3.7的虚拟环境。您也可以修改上述脚本，安装其他版本的Python虚拟环境，或者在虚拟环境中安装所需的第三方Python包。

使用第三方Python包

如果您的第三方Python包是Zip Safe的，可以直接在Python作业中使用，详细信息请参见Python libraries。

如果您的第三方Python包是源码包，且源码包的根目录下存在*setup.py*文件，则这种类型的第三方Python包通常需要先编译才能被使用。您可以选择以下方式编译第三方Python包：

- 在DataFlow集群的某个节点编译第三方Python包。
- 使用*quay.io/pypa/manylinux2014_x86_64*镜像容器中的Python环境来编译第三方Python包，使用该容器编译生成的包兼容绝大多数Linux环境，关于该镜像容器的详细信息，请参见manylinux。

下面以第三方Python包*opencv-python-headless*为例，介绍如何编译和使用该第三方Python包。

1. 编译第三方Python包。

i. 在本地准备*requirements.txt*文件，其内容如下。

```
opencv-python-headless
```

ii. 在本地准备*build.sh*脚本，其内容如下。

```
#!/bin/bash
set -e -x
yum install -y zip
PYBIN=/opt/python/cp37-cp37m/bin
"${PYBIN}/pip" install --target __pypackages__ -r requirements.txt --no-deps
cd __pypackages__ && zip -r deps.zip . && mv deps.zip ../ && cd ..
rm -rf __pypackages__
```

iii. 在CMD命令行中，执行如下命令。

```
docker run -it --rm -v $PWD:/build  -w /build quay.io/pypa/manylinux2014_x86_64 /bin/bash build.sh
```

该命令执行完成后，会生成一个名为*deps.zip*的文件，该文件为编译之后的第三方Python包。您也可以修改*requirements.txt*，安装其他所需的第三方Python包。此外，*requirements.txt*文件中可以指定多个Python依赖。

2. 使用第三方Python包。

关于如何在PyFlink作业中，使用第三方Python包，详情请参见Python Libraries。

使用JAR包

如果您的Flink Python作业中使用了Java类，例如作业中使用了Connector或者Java自定义函数时，则需要指定Connector或者Java自定义函数所在的JAR包，详情请参见JAR Dependencies。

使用数据文件

如果您的Flink Python作业中需要访问数据文件，例如模型文件等，则可以通过Python Archives的方式来访问。

5.1.2.4. 作业迁移

5.1.2.4.1. 迁移方案

本文为您介绍从原有集群迁移Flink作业到DataFlow集群的整体流程，包括准备工作，如何迁移作业，以及常见问题等。

背景信息

EMR集群的基础信息，请参见集群规划。

准备工作

创建DataFlow集群时，集群Core实例的CPU和内存总量可以参考原有集群的规模，再根据作业在DataFlow集群的实际运行情况进行微调。在选择具体机型时，您可以根据是否有本地盘需求、是否希望使用规模较大的物理机等条件进行选择。

Master实例规格通常与整体集群规模大小有关，对应集群最大CU规格经验值如下表所示。

Master型号	集群最大CU规格
4核16 GB	80 CU
8核32 GB	160 CU
16核64 GB	800 CU
24核96 GB	800 CU以上

在EMR控制台创建完DataFlow集群之后，在迁移作业之前，您还需要确保提交Flink作业的客户端所在的机器与DataFlow集群之间的网络互通。

针对原集群的不同情况，您可以选择不同的解决方案：

- 线下IDC自建集群：可以通过阿里云高速通道建立线下IDC和线上E-MapReduce所在VPC网络的连通。
- ECS自建：由于VPC实现用户专有网络之间的逻辑隔离，所以建议使用VPC网络。
  - 经典网络与VPC网络互通：目前阿里云存在经典网络和VPC两种网络类型。由于E-MapReduce集群是在VPC网络中，而很多用户的业务系统还存在于经典网络中，为了解决此问题，阿里云推出了ClassicLink方案，您可以参见此方案进行网络互访，详情请参见建立ClassicLink连接。
  - VPC网络之间连通：选择新旧集群处在同一个区域的同一个可用区内。

由于DataFlow集群采用YARN部署模式，如果提交Flink作业的客户端不位于DataFlow集群内，您还需要在提交Flink作业的客户端的机器上配置好Hadoop相关的配置项，并设置好相关的环境变量。

您自建集群的环境可能与DataFlow集群的环境不完全一致，例如JDK版本不同等。为了保证作业的正常迁移，强烈建议您在迁移生产作业之前，先通过多种方式验证迁移方案是否有效，例如：

- 在迁移生产作业之前，先迁移若干测试作业，观察整个迁移流程是否完备。
- 分批迁移，在迁移重要作业之前，先迁移优先级比较低的作业。甚至可以在条件允许的情况下，您可以继续运行原来的作业，将迁移过来的作业试运行，如果没有问题，再下掉原来的作业。

集群规划

网络互通

Hadoop 环境配置

验证迁移环境

迁移和运行作业

类型	描述
迁移Checkpoint文件	将Checkpoint文件拷贝到DataFlow集群的HDFS上或者上传到OSS中，在提交Flink作业的时候，可以通过-s参数指定Checkpoint文件，即可在DataFlow集群中使用该Checkpoint文件恢复作业。 **注意** 对于DataStream作业来说，开源Flink和VVR的state是完全兼容的，但是对于SQL作业来说，VVR相比社区Flink，做了大量的优化工作，不能保证state完全兼容。对于state不能兼容的作业，无法从开源Flink生成的Checkpoint中恢复。对于这部分作业，客户可以选择使用VVR从零开始重跑，也可以选择使用社区Flink执行。
迁移Flink作业	与原集群中的提交方式类似，您只需要将Flink作业提交到DataFlow集群即可。
运行不同版本的Flink作业	您原集群中的作业使用的Flink版本可能不同，例如，既有基于Flink 1.9版本的，也有基于Flink 1.12版本的。如果您希望不同版本的Flink作业都可以在DataFlow集群中运行，详细方法如下： - 对于特定的Flink版本的作业，使用YARN per-job的方式提交。 在该方式下，作业执行过程中所使用的Flink版本为提交作业的客户端所使用的Flink版本，因此只需要在提交作业的客户端使用指定版本的Flink提交作业即可。 **说明** 如果希望使用本地Flink runtime，则无需指定yarn.provided.lib.dirs参数。 - 对于希望使用DataFlow集群自带的Flink版本（VVR）的作业，则需要通过yarn.provided.lib.dirs参数指定使用集群HDFS中的VVR Runtime（例如，`-D yarn.provided.lib.dirs=hdfs:///flink-current/`），并推荐使用YARN Application模式提交，充分利用集群资源。

对接自建平台

如果您自建了一套大数据平台，则DataFlow集群也可以轻松集成进您现有的平台中：

- 资源管理与运维

  DataFlow集群基于YARN进行资源调度与管理，因此只需要按照集成YARN集群到已有平台的通常操作进行即可。您可以根据需要配置YARN队列的资源等，之后可通过YARN的REST API来访问YARN作业状态，并进行运维。

- 日志查看
  - 对于运行中的作业，可以通过Flink Web UI进行查看，详细信息请参见基础使用。
  - 对于已经运行结束的作业，可以通过Flink History Server查看状态或者通过YARN提供的命令，例如`yarn logs -applicationId application_xxxx_yyyy`来访问作业的日志。

    说明 其中Flink History Server的日志默认存储在HDFS 群的*hdfs:///flink/flink-jobs/*目录下，YARN日志默认存储在HDFS集群的*hdfs:///tmp/logs/$USERNAME/logs/*目录下。

- 指标监控

  如果您的自建平台已有监控报警系统，则可以在Flink的作业配置中，配置相关的Metric Reporter即可。另外，DataFlow集群的Flink作业的指标也对接了EMR的监控报警。

- 报警

  除了可以使用您的报警体系之外，也可以使用阿里云提供的云监控（CloudMonitor）来配置报警规则，并对接邮箱、钉钉群等提醒机制。使用云监控配置详情，请参见创建阈值报警规则。

常见问题

- Q：JDK版本不一致，如何处理？

A：DataFlow集群中使用的是OpenJDK，如果您在原来集群的作业使用的是Oracle JDK，并且您的作业中使用了Oracle JDK中特有的功能时，JDK版本的不一致可能导致运行作业时部分类找不到（例如，javafx.util.Pair等）。因此，您的作业中需要避免显式使用Oracle JDK特有的依赖。

- Q：是否可以支持多个Flink版本？

  A：可以。DataFlow集群支持多种作业提交方式，例如YARN per-job方式、YARN Application方式等。基于Flink on YARN的部署模式，在未设置yarn.provided.lib.dirs参数的情况下，Flink作业在YARN集群中运行时所使用的Flink Runtime为提交作业的客户端所使用的Flink（例如，开源Flink 1.13等）。因此如果您想使用特定的Flink版本运行作业，有两种方式：

  - 直接使用Flink的特定版本进行提交，并且不设置yarn.provided.lib.dirs参数。
  - 通过指定yarn.provided.lib.dirs参数，来使用特定的Flink Runtime。另外，在该场景下，考虑到兼容性，推荐您使用YARN per-job模式进行提交。

5.1.2.5. 最佳实践

5.1.2.5.1. Dataflow集群连接数据湖元数据DLF

EMR-3.38.3及后续版本的Dataflow集群，可以通过数据湖元数据DLF（Data Lake Formation）作为元数据读取Hadoop集群中的数据。本文为您介绍Dataflow集群如何连接DLF，并读取Hive全量数据。

前提条件

- 已在E-MapReduce控制台上创建DataFlow集群和Hadoop集群，详情请参见创建集群。

  > **注意** 创建Hadoop集群时，元数据选择为**数据湖元数据**。

- 已开通数据湖构建DLF，详情请参见快速入门。

使用限制

- DataFlow集群和Hadoop集群需要在同一VPC下。
- 创建的Dataflow集群需要为EMR-3.38.3后续版本。

操作流程

1. 步骤一：数据准备
2. 步骤二：Dataflow集群连接DLF
3. 步骤三：验证读取Hive全量数据

步骤一：数据准备

1. 下载Hive作业需要的测试数据至OSS对应的目录。

   例如，上传目录为*oss://<yourBucketName>/hive/userdata/*，其中<yourBucketName>为您在OSS控制台上创建的Bucket名称。上传文件详细信息，请参见上传文件。

2. 在DLF控制台创建元数据库，详情请参见创建元数据库。

   例如，创建的元数据库名称为myhudi，选择路径为oss://<yourBucketName>/hive/db。

3. 在Hadoop集群中，查看已经创建的元数据库。

   i. 通过SSH方式登录Hadoop集群，详情请参见登录集群。

   ii. 执行以下命令，切换为hadoop用户。

   ```
   su hadoop
   ```

   iii. 执行以下命令，进入Hive命令行。

   ```
   hive
   ```

   iv. 执行以下命令，查看表信息。

   ```
   desc database myhudi;
   ```

   > **说明** 命令中的myhudi为上一步骤中创建的数据库的名称。

   返回信息如下。

   ```
   OK
   myhudi          oss://aliyu****/hive/db    acs:ram::125046002175****:user/299153685100086****       USER
   Time taken: 0.069 seconds, Fetched: 1 row(s)
   ```

4. 执行以下命令，创建Hive的外表。

```
USE myhudi;
set hive.input.format=org.apache.hadoop.hive.ql.io.HiveInputFormat;
set hive.stats.autogather=false;
DROP TABLE IF EXISTS emrusers;
CREATE EXTERNAL TABLE emrusers (
  userid INT,
  movieid INT,
  rating INT,
  unixtime STRING )
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY '\t'
 STORED AS TEXTFILE
 LOCATION 'oss://<yourBucketName>/hive/userdata/';
```

> **说明** 请替换命令中的<yourBucketName>为您实际在OSS控制台上创建的Bucket名称。

5. 在DLF控制台上，验证表信息。
   i. 登录数据湖构建控制台。
   ii. 在左侧导航栏，选择**元数据管理 > 元数据表**。
   iii. 在**元数据表**页面，通过库名过滤，可以查看已创建的表信息。

6. （可选）在Hive命令行中，查询数据。
   - 示例1

     ```
     SELECT userid,movieid,rating,unix_timestamp() from emrusers limit 10;
     ```

   - 示例2

     ```
     SELECT movieid,count(userid) as usercount from emrusers group by movieid order by usercount desc limit 50;
     ```

步骤二：Dataflow集群连接DLF

1. 通过SSH方式登录DataFlow集群，详情请参见登录集群。
2. 执行以下命令，新建test目录。

   ```
   mkdir /root/test
   ```

3. 上传Hive配置文件到DataFlow集群的新建路径下。

   您可以执行以下命令，复制Hadoop集群中的*hive-site.xml*文件到DataFlow集群。

   ```
   scp root@<master-1-1节点内网的IP地址>:/etc/taihao-apps/hive-conf/hive-site.xml /root/test/
   ```

   > **说明** 命令中的<master-1-1节点内网的IP地址>，您可以在EMR控制台的集群管理的**集群基础信息**页面或**主机列表**页面查看，*/root/test/*为DataFlow集群的路径，您可以根据实际情况修改。

4. 加载集群中内置的Hive Connector，启动SQL客户端。
   i. 在Maven仓库官网中下载依赖的JAR包，并上传至DataFlow集群中。

   本示例是上传到/root目录，下载的JAR包为Jackson Core、Jackson Databind和Jackson Annotations。

   具体版本请根据您实际情况下载，本示例下载的JAR包为*jackson-core-2.12.1.jar*、*jackson-databind-2.12.1.jar*和*jackson-annotations-2.12.1.jar*。

   ii. 执行以下命令，启动SQL客户端。

   ```
   sql-client.sh -j /opt/apps/FLINK/flink-current/opt/catalogs/hive-2.3.6/ververica-connector-hive-2.3.6-1.13-vvr-4.0.11-2-SNAPSHOT-jar-with-dependencies.jar -j /root/jackson-core-2.12.1.jar -j /root/jackson-databind-2.12.1.jar -j /root/jackson-annotations-2.12.1.jar
   ```

   > **说明** 本示例中vvr版本是4.0.11-2，您可以根据实际集群版本选择对应的Connector版本。

5. 在Flink SQL命令行中，创建Catalog。

```
CREATE CATALOG dlfcatalog WITH (
     'type' = 'hive',
     'default-database' = 'myhudi',
     'hive-version' = '2.3.6',
     'hive-conf-dir' = '/root/test',
     'hadoop-conf-dir' = '/etc/taihao-apps/yarn-conf/'
 );
```

涉及参数如下表。

参数	描述
type	固定值为hive。
default-database	步骤一：数据准备中创建的数据库的名称。 本示例为myhudi。
hive-version	固定值为2.3.6。
hive-conf-dir	前一步骤中复制的*hive-site.xml*所在的目录。 本示例为/root/test。
hadoop-conf-dir	固定值为/etc/taihao-apps/yarn-conf/。

返回信息如下，表示创建成功。

```
[INFO] Execute statement succeed.
```

6. 在Flink SQL命令行中，查看DLF的数据库。
   i. 执行以下命令，设置当前的Catalog。

   ```
   USE CATALOG dlfcatalog;
   ```

   ii. 执行以下命令，使用创建的Catalog。

   ```
   SHOW DATABASES;
   ```

   iii. 执行以下命令，设置当前的数据库。

   ```
   USE myhudi;
   ```

   iv. 执行以下命令，查看当前数据库中的表。

   ```
   SHOW TABLES;
   ```

   返回信息如下。

   ```
   +------------+
   | table name |
   +------------+
   |   emrusers |
   +------------+
   1 row in set
   ```

   v. 执行以下命令，查看表信息。

   ```
   desc emrusers;
   ```

   返回信息如下。

   ```
   +----------+--------+------+-----+--------+-----------+
   |     name |   type | null | key | extras | watermark |
   +----------+--------+------+-----+--------+-----------+
   |   userid |    INT | true |     |        |           |
   |  movieid |    INT | true |     |        |           |
   |   rating |    INT | true |     |        |           |
   | unixtime | STRING | true |     |        |           |
   +----------+--------+------+-----+--------+-----------+
   4 rows in set
   ```

步骤三：验证读取Hive全量数据

1. 通过SSH方式登录DataFlow集群，详情请参见登录集群。
2. 执行以下命令，设置为Per-Job Cluster模式。

```
echo "execution.target: yarn-per-job" >> /etc/taihao-apps/flink-conf/flink-conf.yaml
```

3. 执行以下命令，启动SQL客户端。

```
sql-client.sh -j /opt/apps/FLINK/flink-current/opt/catalogs/hive-2.3.6/ververica-connector-hive-2.3.6-1.13-vvr-4.0.11-2-SNAPSHOT-jar-with-dependencies.jar -j /root/jackson-core-2.12.1.jar -j /root/jackson-databind-2.12.1.jar -j /root/jackson-annotations-2.12.1.jar
```

4. 在Flink SQL命令行中进行如下操作。
   i. 执行以下命令，创建Catalog。

```
CREATE CATALOG dlfcatalog WITH (
     'type' = 'hive',
     'default-database' = 'myhudi',
     'hive-version' = '2.3.6',
     'hive-conf-dir' = '/root/test',
     'hadoop-conf-dir' = '/etc/taihao-apps/yarn-conf/'
 );
```

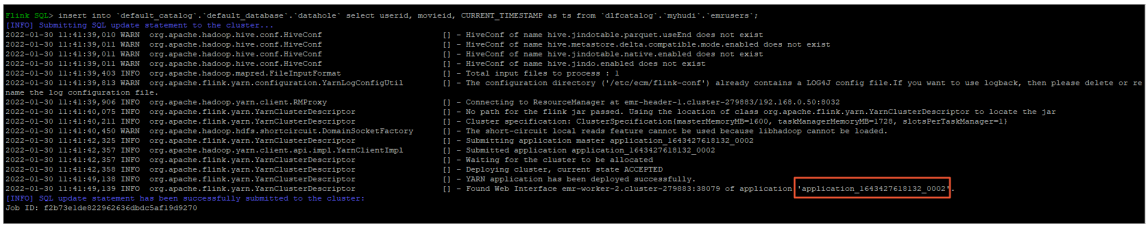
   ii. 执行以下命令，创建表。

```
create table default_catalog.default_database.datahole(userid int, movieid int, ts timestamp) with ('connector' = 'blackhole');
```

   iii. 执行以下命令，读取Hive全量数据到blackhole。

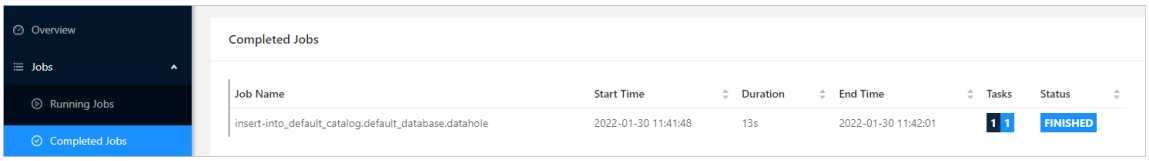
```
insert into `default_catalog`.`default_database`.`datahole` select userid, movieid, CURRENT_TIMESTAMP as ts from `dlfcatalog`.`myhudi`.`emrusers`;
```

   执行成功后，会返回已提交的Flink作业的Application ID。返回如下类似信息。



> **说明** 本示例返回的Application ID为application_1643427618132_0002。

5. 查看作业。
   i. 通过Web UI查看作业状态，详情请参见通过Web UI查看作业状态。
   ii. 在YARN控制台，单击目标作业的Application ID，可以查看作业运行的详情。
   iii. 单击**Tracking URL**所在行的链接。
   iv. 在左侧导航栏中，选择**Jobs > Completed Jobs**。
   可以查看已完成的作业。



5.1.2.5.2. 通过DataFlow集群将数据流式写入OSS

在EMR 3.37.1及之后的版本中，DataFlow集群内置了JindoFS相关的依赖。您可以在DataFlow集群中运行Flink作业，将数据流式写入阿里云OSS。本文通过示例为您介绍如何在Dataflow集群中运行Flink作业写入同账号下的阿里云OSS。

背景信息

关于JindoFS的部分高级配置（例如，熵注入），请参见支持Flink可恢复性写入JindoFS或OSS。

前提条件

- 已开通E-MapReduce服务和OSS服务。
- 已完成云账号的授权，详情请参见角色授权。

操作流程

1. 步骤一：准备环境
2. 步骤二：准备JAR包
3. 步骤三：运行Flink作业
4. 步骤四：在OSS上查看输出的结果

步骤一：准备环境

1. 创建Dataflow集群，详情请参见创建集群。

   > **说明** 本文以EMR-3.39.1版本为例。

2. 在OSS上创建与Dataflow集群相同区域的Bucket，详情请参见创建存储空间。

步骤二：准备JAR包

1. 下载DataFlowOSSDemo.tar.gz。

   基于JindoFS，您可以在Flink作业中，像写HDFS一样将数据以流式的方式写入OSS中（路径需要以oss://为前缀）。本示例中使用了Flink的StreamingFileSink方法来演示开启了检查点（Checkpoint）之后，Flink如何以Exactly-Once语义写入OSS。

```
import org.apache.flink.api.common.serialization.SimpleStringEncoder;
import org.apache.flink.api.common.typeinfo.TypeInformation;
import org.apache.flink.configuration.Configuration;
import org.apache.flink.core.fs.Path;
import org.apache.flink.streaming.api.CheckpointingMode;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.api.functions.sink.filesystem.StreamingFileSink;
import org.apache.flink.streaming.api.functions.sink.filesystem.rollingpolicies.OnCheckpointRollingPolicy;
import org.apache.flink.streaming.api.functions.source.RichParallelSourceFunction;
import java.util.Random;
public class OssDemoJob {
    public static void main(String[] args) throws Exception {
        // Set up the streaming execution environment
        final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        // Checkpoint is required
        env.enableCheckpointing(30000, CheckpointingMode.EXACTLY_ONCE);
        // Change to your oss path
        String outputPath = "oss://targetBucket/targetDir";
        DataStream<String> StringStream =
            env.addSource(new RandomStringSource()).returns(TypeInformation.of(String.class));
        StreamingFileSink<String> sink =
            StreamingFileSink.forRowFormat(
            new Path(outputPath), new SimpleStringEncoder<String>("UTF-8"))
            .withRollingPolicy(OnCheckpointRollingPolicy.build())
            .build();
        StringStream.addSink(sink);
        // Compile and submit the job
        env.execute();
    }
    public static class RandomStringSource extends RichParallelSourceFunction<String> {
        private volatile boolean cancelled = false;
        private Random random;
        @Override
        public void open(Configuration parameters) throws Exception {
            super.open(parameters);
            random = new Random();
        }
        @Override
        public void run(SourceContext<String> ctx) throws Exception {
            while (!cancelled) {
                long nextLong = random.nextLong();
                synchronized (ctx.getCheckpointLock()) {
                    ctx.collect("s" + nextLong);
                }
            }
        }
        @Override
        public void cancel() {
            cancelled = true;
        }
    }
}
```

> **说明** 本示例代码片段给出了主要的示例程序，您可以根据自身环境进行修改（例如，添加包名，修改代码中的OSS路径）后，进行编译。关于如何构建Flink作业的JAR包，可以参见Flink官方文档。

2. 在CMD命令行中，进入到下载文件中*pom.xml*所在的目录下，执行以下命令打包文件。

```
mvn clean package
```

根据您*pom.xml*文件中artifactId的信息，下载文件中的target目录下会出现*OssDemoJob.jar*的JAR包。

步骤三：运行Flink作业

1. 通过SSH方式连接Dataflow集群，详情请参见登录集群。
2. 上传打包好的*OssDemoJob.jar*至Dataflow集群的根目录下。

> **说明** 本文示例中*OssDemoJob.jar*是上传至*root*根目录下，您也可以自定义上传路径。

3. 执行以下命令，提交作业。

   本示例通过Per-Job Cluster模式提交作业，其他方式请参见基础使用。

   ```
   flink run -t yarn-per-job -d OssDemoJob.jar
   ```

   ```
   [root@emr-header-1 ~]# flink run -t yarn-per-job -d OssDemoJob.jar
   SLF4J: Class path contains multiple SLF4J bindings.
   SLF4J: Found binding in [jar:file:/opt/apps/ecm/service/flink/1.13-vvr-4.0.11-2/package/flink-1.13-vvr-4.0.11-2/lib/log4j-slf4j-impl-2.17.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
   SLF4J: Found binding in [jar:file:/opt/apps/ecm/service/hadoop/2.8.5-1.6.5/package/hadoop-2.8.5-1.6.5/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
   SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
   SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
   2022-03-23 14:13:20,075 WARN  org.apache.flink.yarn.configuration.YarnLogConfigUtil        [] - The configuration directory ('/etc/ecm/flink-conf') already contains a LOG4J config file.If you want
   name the log configuration file.
   2022-03-23 14:13:20,303 INFO  org.apache.hadoop.yarn.client.RMProxy                        [] - Connecting to ResourceManager at emr-header-1.cluster-290222/192.168.[redacted]8032
   2022-03-23 14:13:20,502 INFO  org.apache.flink.yarn.YarnClusterDescriptor                  [] - No path for the flink jar passed. Using the location of class org.apache.flink.yarn.YarnClusterDescri
   2022-03-23 14:13:20,807 INFO  org.apache.flink.yarn.YarnClusterDescriptor                  [] - Cluster specification: ClusterSpecification{masterMemoryMB=1600, taskManagerMemoryMB=1728, slotsPerTa
   2022-03-23 14:13:21,343 WARN  org.apache.hadoop.hdfs.shortcircuit.DomainSocketFactory      [] - The short-circuit local reads feature cannot be used because libhadoop cannot be loaded.
   2022-03-23 14:13:23,349 INFO  org.apache.flink.yarn.YarnClusterDescriptor                  [] - Submitting application master application_1647310741576_0001
   2022-03-23 14:13:23,794 INFO  org.apache.hadoop.yarn.client.api.impl.YarnClientImpl        [] - Submitted application application_1647310741576_0001
   2022-03-23 14:13:23,794 INFO  org.apache.flink.yarn.YarnClusterDescriptor                  [] - Waiting for the cluster to be allocated
   2022-03-23 14:13:23,797 INFO  org.apache.flink.yarn.YarnClusterDescriptor                  [] - Deploying cluster, current state ACCEPTED
   2022-03-23 14:13:33,884 INFO  org.apache.flink.yarn.YarnClusterDescriptor                  [] - YARN application has been deployed successfully.
   2022-03-23 14:13:33,886 INFO  org.apache.flink.yarn.YarnClusterDescriptor                  [] - The Flink YARN session cluster has been started in detached mode. In order to stop Flink gracefully,
   $ echo "stop" | ./bin/yarn-session.sh -id application_1647310741576_0001
   If this should not be possible, then you can also kill Flink via YARN's web interface or via:
   $ yarn application -kill application_1647310741576_0001
   Note that killing Flink might not clean up all job artifacts and temporary files.
   2022-03-23 14:13:33,891 INFO  org.apache.flink.yarn.YarnClusterDescriptor                  [] - Found Web Interface emr-worker-1.cluster-290222:34475 of application 'application_1647310741576_0001'
   Job has been submitted with JobID 17a841b5b1aa6c3ec5f45df3ec368b7e
   ```

   您可以执行以下命令，查看作业状态。

   ```
   flink list -t yarn-per-job -Dyarn.application.id=<appId>
   ```

   > **说明** `<appId>` 为作业运行后返回的Application ID。例如，本示例截图中的application_1647310741576_0001。

步骤四：在OSS上查看输出的结果

作业正常运行后，您可以在OSS控制台查看输出结果。

1. 登录 OSS管理控制台。
2. 单击创建的存储空间。
3. 在文件管理页面指定的输出目录下查看输出结果。

   输出结果类似下图所示。

   > **说明** 该作业为流式作业会持续运行，产生较多输出文件。在完成验证后，应及时在命令行中通过 `yarn application -kill <appId>` 命令终止该作业。

5.1.2.6. 常见问题

本文汇总了DataFlow集群使用时的常见问题。

- DataFlow集群外的机器，如何提交作业到DataFlow集群？
- 在DataFlow集群外机器上，如何解析DataFlow集群中的hostname？
- 如何查看Flink作业的运行状态？
- 如何访问Flink作业的日志？
- 如何访问DataFlow集群中的Flink HistoryServer？
- 如何使用DataFlow集群中所支持的商业化Connector？
- 如何使用GeminiStateBackend？
- 如何开启Flink作业JobManager的HA？
- 如何处理上下游存储（Connector）问题？
- 通过DataFlow集群运行Flink作业免密读写OSS时报错，该如何处理？

DataFlow集群外的机器，如何提交作业到DataFlow集群？

您可以根据以下步骤，通过DataFlow集群外的机器，提交作业到DataFlow集群：

1. 确保DataFlow集群和DataFlow集群外的机器网络互通。
2. 配置提交Flink作业的客户端的Hadoop YARN环境。

   DataFlow集群中的Hadoop YARN的软件安装目录是*/opt/apps/YARN/yarn-current*，配置文件的目录是*/etc/taihao-apps/yarn-conf*，您需要将*yarn-current*目录及*yarn-conf*目录下载到提交Flink作业的客户端上。

   然后，在提交Flink作业的客户端上，配置如下环境变量。

   ```
   export HADOOP_HOME=/path/to/yarn-current && \
   export PATH=${HADOOP_HOME}/bin/:$PATH && \
   export HADOOP_CLASSPATH=$(hadoop classpath) && \
   export HADOOP_CONF_DIR=/path/to/yarn-conf
   ```

   注意 Hadoop的配置文件中（例如*yarn-site.xml*等）配置的服务地址（例如ResourceManager等），使用的是全域名（FQDN，Fully Qualified Domain Name）。例如，master-1-1.c-xxxxxxxxxx.cn-hangzhou.emr.aliyuncs.com。因此，如果您通过集群外的机器提交作业，需要能够解析这些FQDN或者将配置文件中的FQDN修改成对应的IP地址。

3. 完成以上配置后，您在集群外的机器上启动Flink作业（例如，运行命令`flink run -d -t yarn-per-job -ynm flink-test $FLINK_HOME/examples/streaming/TopSpeedWindowing.jar`）后，应当能在DataFlow集群的YARN Web UI中看到相应的Flink作业。

在DataFlow集群外机器上，如何解析DataFlow集群中的hostname？

您可以通过以下方式，在DataFlow集群外的机器上，解析DataFlow集群中的hostname：

- 修改提交Flink作业的客户端上的*/etc/hosts*文件，添加相应的hostname到IP的映射。
- 通过PrivateZone提供的DNS解析服务。

  如果您有自己的域名解析服务，也可以通过如下方式，配置JVM的运行参数，使用自己的域名解析服务。

  ```
  env.java.opts.client: "-Dsun.net.spi.nameservice.nameservers=xxx -Dsun.net.spi.nameservice.provider.1=dns,sun -Dsun.net.spi.nameservice.domain=yyy"
  ```

如何查看Flink作业的运行状态？

- 通过EMR控制台查看。

  EMR支持Knox，可以通过公网方式访问YARN、Flink等的Web UI界面，Flink的Web UI可以通过YARN进行查看，详细信息请参见通过Web UI查看作业状态。

- 通过SSH隧道的方式查看，详情信息请参见通过SSH隧道方式访问开源组件Web UI。
- 直接访问YARN REST接口。

  ```
  curl --compressed -v  -H "Accept: application/json" -X GET "http://master-1-1:8088/ws/v1/cluster/apps?states=RUNNING&queue=default&user.name=***"
  ```

  说明 需确保安全组开放了8443和8088端口，可以访问到YARN的REST接口或者DataFlow集群和访问的节点处于同一内网中。

如何访问Flink作业的日志？

- 对于运行中的作业，可以通过Flink Web UI，访问Flink作业的日志。
- 对于已经运行结束的作业，可以通过Flink History Server查看作业的统计信息或者通过命令`yarn logs -applicationId application_xxxx_yyyy`访问作业的日志，已经运行结束的作业的日志默认保存在HDFS集群的*hdfs:///tmp/logs/$USERNAME/logs/*目录下。

如何访问DataFlow集群中的Flink HistoryServer？

DataFlow集群会默认在master-1-1节点（即header机器组的第一台机器）的8082端口启动Flink HistoryServer，用于收集已运行结束的作业的统计信息，具体访问方式如下：

1. 配置安全组规则，开放master-1-1节点的8082端口的访问权限。
2. 直接访问*http://$master-1-1-ip:8082*。

如何使用DataFlow集群中所支持的商业化Connector？

DataFlow集群提供了很多商业化Connector，例如Hologres、SLS、MaxCompute、DataHub、Elasticsearch和ClickHouse等，您在Flink作业中除了可以使用开源的Connector之外，还可以使用这些商业化Connector。下面以Hologres Connector为例，介绍如何在Flink作业中使用DataFlow集群所携带的商业化Connector。

- 作业开发
  i. 下载DataFlow集群所携带的商业化Connector的JAR包（位于DataFlow集群的*/opt/apps/FLINK/flink-current/opt/connectors*目录

下），并通过如下方式将商业化Connector安装在本地Maven环境中。

```
mvn install:install-file -Dfile=/path/to/ververica-connector-hologres-1.13-vvr-4.0.7.jar -DgroupId=com.alibaba.ververica -DartifactId=ververica-connector-hologres -Dversion=1.13-vvr-4.0.7 -Dpackaging=jar
```

ii. 在项目的*pom.xml*文件中添加以下依赖。

```
<dependency>
    <groupId>com.alibaba.ververica</groupId>
    <artifactId>ververica-connector-hologres</artifactId>
    <version>1.13-vvr-4.0.7</version>
    <scope>provided</scope>
</dependency>
```

- 运行作业
  - 方式一：
    a. 拷贝Hologres Connector到一个独立的目录。

    ```
    hdfs mkdir hdfs:///flink-current/opt/connectors/hologres/
    hdfs cp hdfs:///flink-current/opt/connectors/ververica-connector-hologres-1.13-vvr-4.0.7.jar  hdfs:///flink-current/opt/connectors/hologres/ververica-connector-hologres-1.13-vvr-4.0.7.jar
    ```

    b. 提交作业时，命令中添加以下参数。

    ```
    -D yarn.provided.lib.dirs=hdfs:///flink-current/opt/connectors/hologres/
    ```

  - 方式二：
    a. 拷贝Hologres Connector到提交Flink作业的客户端的*/opt/apps/FLINK/flink-current/opt/connectors/ververica-connector-hologres-1.13-vvr-4.0.7.jar*目录下，与DataFlow集群中的目录结构保持一致。
    b. 提交作业时，命令中添加以下参数。

    ```
    -C file:///opt/apps/FLINK/flink-current/opt/connectors/ververica-connector-hologres-1.13-vvr-4.0.7.jar
    ```

  - 方式三：将Hologres Connector打包到作业的JAR包中。

如何使用GeminiStateBackend？

DataFlow集群提供了企业版StateBackend（即GeminiStateBackend），性能是开源版本的3～5倍。关于如何使用GeminiStateBackend以及更详细的信息，请参见Flink（VVR）作业配置。

如何开启Flink作业JobManager的HA？

DataFlow集群基于YARN模式部署并运行Flink作业，您可以按照社区中的Configuration开启JobManager的HA，从而使Flink作业可以更稳定的运行。配置示例如下所示。

```
high-availability: zookeeper
high-availability.zookeeper.quorum: 192.168.**.**:2181,192.168.**.**:2181,192.168.**.**:2181
high-availability.zookeeper.path.root: /flink
high-availability.storageDir: hdfs:///flink/recovery
```

> 注意　开启HA后，默认情况下，JobManager在失败后最多重启一次。如果您想让JobManager重启多次，还需要设置YARN的yarn.resourcemanager.am.max-attempts参数和Flink的yarn.application-attempts参数，详情请参见Flink官方文档。除此之外，根据经验，通常还需要调整yarn.application-attempt-failures-validity-interval参数的值，将其从默认的10000毫秒（10秒）调整到一个比较大的值，例如调大为300000毫秒（5分钟），防止JobManager不停的重启。

如何处理上下游存储（Connector）问题？

关于上下游存储方面的常见问题，请参见上下游存储。

通过DataFlow集群运行Flink作业免密读写OSS时报错，该如何处理？

您需要根据具体的报错信息，进行相应的处理：

- 报错提示 `java.lang.UnsupportedOperationException: Recoverable writers on Hadoop are only supported for HDFS` 。
  - 问题原因：DataFlow集群目前是通过内置的JindoSDK来支持免密读写OSS，并支持StreamingFileSink等API的，不需要再按照社区文档进行额外的配置，否则会由于依赖冲突导致此报错。
  - 处理方法：检查您集群内提交作业的机器的*$FLINK_HOME/plugins*目录，查看是否放置了*oss-fs-hadoop*目录。如果放置了该目录，请删除该目录后重新提交作业。
- 报错提示 `Could not find a file system implementation for scheme 'oss'. The scheme is directly supported by Flink through the following plugin: flink-oss-fs-hadoop. ....` 。

- 问题原因：EMR-3.40及之前的版本的EMR集群中，master机器组内非master-1-1机器上可能缺少Jindo相关的JAR包。
- 处理方法：
  - EMR-3.40及之前的版本：检查您的集群内提交作业的机器的*$FLINK_HOME/lib*目录下是否有Jindo相关的JAR包，例如jindo-flink-4.0.0.jar。如果没有Jindo相关的JAR包，您可以在集群中运行以下命令将Jindo相关的JAR包，拷贝到*$FLINK_HOME/lib*目录后重新提交作业。

    ```
    cp /opt/apps/extra-jars/flink/jindo-flink-*-full.jar $FLINK_HOME/lib
    ```

  - EMR-3.40之后版本：优化了支持的方式，即使*$FLINK_HOME/lib*目录下没有Jindo相关的JAR包，读写OSS的作业也是可以正常运行的，如果您仍遇到此报错信息，请提交工单处理。

5.1.3. Kafka

5.1.3.1. 基础使用

5.1.3.1.1. 创建集群

本文为您介绍创建DataFlow Kafka集群（选择了Kafka服务）的详细操作步骤和相关配置。

注意事项

创建DataFlow Kafka集群前，您需要根据业务的预估负载，选择合适的ECS实例机型以及Broker实例个数。由于业务场景差异很大，所以无法给出通用的集群规划，您需要根据您的实际环境创建集群。通常，建议您选择机型时考虑以下配置：

- Broker机型的CPU和内存配比为1 Core：4 GB。
- 选择云盘作为数据存储盘。
- 充分考虑云盘的IO吞吐率以及网卡带宽之间的关系。

在部署参数上，考虑以下因素：

- 由于EMR Kafka版本仍依赖于Zookeeper，且Zookeeper的可用性直接关系到Kafka服务的高可用，因此，建议您创建集群时，选择高可用的部署方式。启用高可用后，将创建3个节点的Zookeeper服务。
- 如果Master机器组只部署Zookeeper，则Master机器组只需要配置1块数据盘即可。

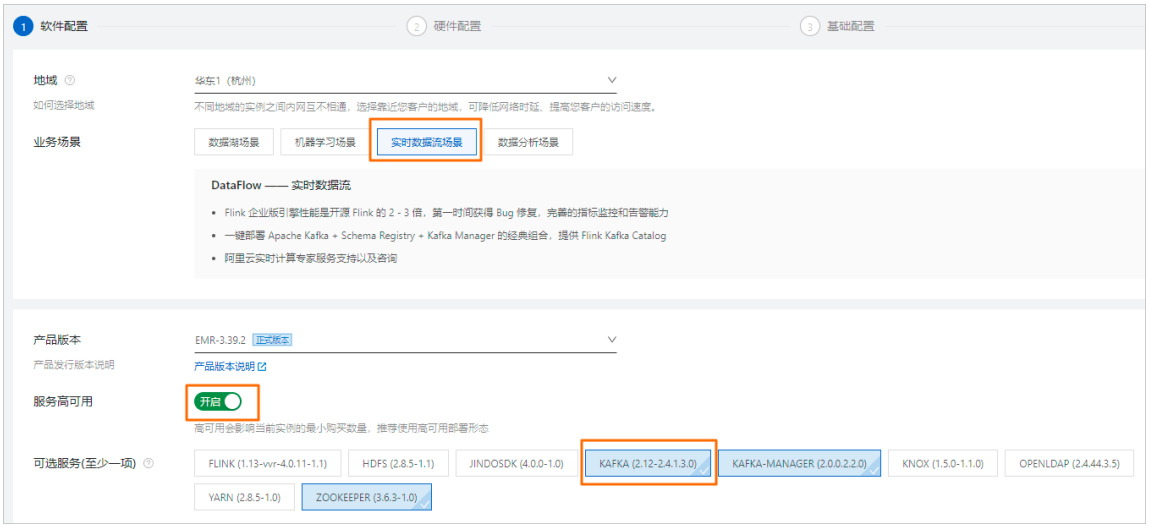
操作步骤

1. 进入创建集群页面。
   i. 登录EMR on ECS控制台。
   ii. （可选）在顶部菜单栏处，根据实际情况选择地域和资源组。
      - 地域：创建的集群将会在对应的地域内，一旦创建不能修改。
      - 资源组：默认显示账号全部资源。
   iii. 单击上方的**创建集群**，进行创建。
2. 配置集群信息。

   创建集群时，您需要对集群进行软件配置、硬件配置和基础配置。

   > 注意　集群创建完成后，除了集群名称以外，其他配置均无法修改，所以在创建时请仔细确认各项配置。

i. 软件配置。



配置项	示例	描述
地域	华东1（杭州）	创建的集群将会在对应的地域内，一旦创建不能修改。
业务场景	实时数据流场景	选择**实时数据流场景**。
产品版本	EMR-3.39.2	选择EMR版本后，您可以查看各服务的版本。 例如，EMR-3.39.2版本中的Kafka为2.12-2.4.1.3.0，其中2.12表示Scala的版本，2.4.1为开源Kafka的版本。
服务高可用	开启	默认开启。 **注意** 启用高可用后，将在Master机器组上部署3个节点的Zookeeper服务。由于EMR Kafka版本的服务可用性仍依赖于Zookeeper，所以建议您创建集群时，选择高可用的部署方式。
可选服务	Kafka	选择**Kafka**服务。 您也可以根据您的实际需求选择其他的一些组件，被选中的组件会默认启动相关的服务进程。
高级设置	不开启	**软件自定义配置**：可指定JSON文件对集群中的基础软件（例如Hadoop、Spark和Hive等）进行配置。默认不开启。

ii. 硬件配置。

配置项	示例	描述
付费类型	按量付费	默认包年包月。当前支持的付费类型如下： ■ **按量付费**：一种后付费模式，即先使用再付费。按量付费是根据实际使用的小时数来支付费用，每小时计费一次，适合短期的测试任务或是灵活的动态任务。 ■ **包年包月**：一种预付费模式，即先付费再使用。 **说明** 建议测试场景下使用**按量付费**，测试正常后再新建一个**包年包月**的生产集群正式使用。
可用区	华东1（杭州） 可用区 I	可用区为在同一地域下的不同物理区域，可用区之间内网互通。通常使用默认的可用区即可。
专有网络	emr_test/vpc-bp1f4epmkvncimpgs****	默认选择已有的专有网络。 如需创建新的专有网络，请在专有网络控制台新创建一个，详情请参见创建和管理专有网络。
交换机	vsw_test/vsw-bp1e2f5fhaplp0g6p****	选择在对应VPC下可用区的交换机，如果在这个可用区没有可用的交换机，则需要在专有网络控制台新创建一个，详情请参见创建和管理交换机。
安全组	sg-bp1ddw7sm2risw****/sg-bp1ddw7sm2risw****	默认选择已有的安全组。安全组详情请参见安全组概述。 您也可以单击**新建安全组**，在ECS控制台新建一个安全组，详情请参见创建安全组。 **注意** 禁止使用ECS上创建的企业安全组。
挂载公网	开启	集群是否挂载弹性公网IP地址，默认不开启。 **说明** 创建后如果您需要使用公网IP地址访问，请在ECS上申请开通公网IP地址，详情请参见弹性公网IP中的申请EIP的内容。
实例	根据实际情况配置	您可以根据需要选择实例规格，建议Broker机型的CPU和内存配比为1 Core：4 GB。 ■ **系统盘**：根据需要选择系统盘。 ■ **系统盘大小**：根据需要调整磁盘容量，推荐至少120 GB。取值范围为80 ~ 500 GB。 ■ **数据盘**：根据需要选择数据盘。 **说明** 建议选择云盘。 ■ **数据盘大小**：根据需要调整磁盘容量，推荐至少80 GB。取值范围为40 ~ 32768 GB。 ■ **实例数量**：默认3台Master，3台Core。

iii. 基础配置。

在**基础信息**区域，配置如下参数。

> 注意　暂不支持高级配置区域的参数，因此请勿设置。

配置项	示例	描述
集群名称	Emr-Kafka	集群的名字，长度限制为1~64个字符，仅可使用中文、字母、数字、短划线（-）和下划线（_）。
身份凭证	自定义密码	**密钥对**（默认）：使用SSH密钥对登录Linux实例。 关于密钥对的使用详情，请参见SSH密钥对。
		密码：设置Master节点的登录密码，使用密码对登录Linux实例。 密码规则：8~30个字符，且必须同时包含大写字母、小写字母、数字和特殊字符。 特殊字符包括：感叹号（!）、at（@）、井号（#）、美元符号（$）、百分号（%）、乘方（^）、and（&）和星号（*）。
高级设置	根据需求配置	■ **添加用户**：可选配置，添加访问开源大数据软件Web UI的账号。 ■ **ECS应用角色**：当用户的程序在EMR计算节点上运行时，可不填写阿里云AccessKey来访问相关的云服务（例如OSS），EMR会自动申请一个临时AccessKey来授权本次访问。**ECS应用角色**用于控制这个AccessKey的权限。 ■ **引导操作**：可选配置，您可以在集群启动Hadoop前执行您自定义的脚本。 ■ **资源组**：可选配置。

3. 在**确认订单**页面，选中**E-MapReduce服务条款**复选框。
4. 单击**创建**。

创建集群后可以通过刷新页面来查看进度，当集群**状态**显示为**运行中**时，表示集群创建成功。

后续步骤

集群创建成功后，您可以根据实际的业务场景，修改集群的默认参数，使集群正式交付生产时符合相关的要求。例如：

- 是否启用SSL来加密网络链接，详情请参见使用SSL加密Kafka链接。
- 是否启用SASL来进行登录认证，详情请参见使用SASL登录认证Kafka服务。

5.1.3.1.2. 通过公网访问Kafka

本文介绍如何在新建集群时或者为已有集群开通公网服务，使得客户端可以在公网环境访问E-MapReduce Kafka服务。

使用限制

本文适用于E-MapReduce 3.39.0之后版本和E-MapReduce 5.5.0之后版本。

新建集群时，开通公网服务

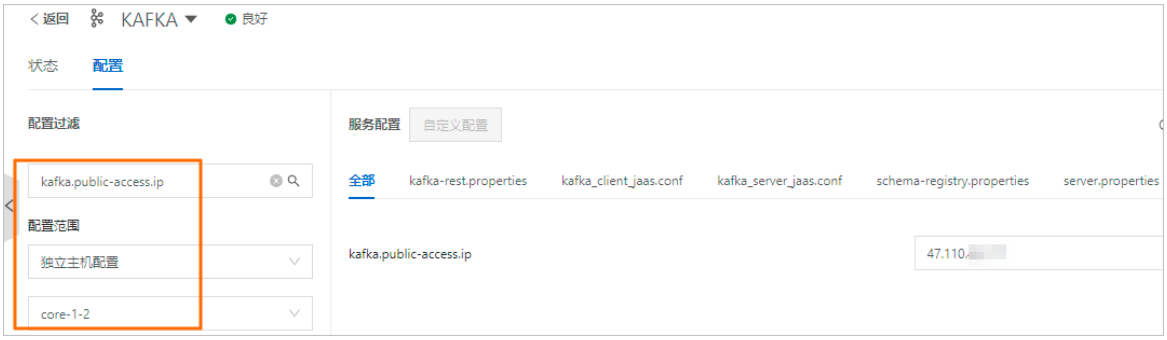
在EMR on ECS控制台创建Dataflow集群时，打开**挂载公网**的开关，则创建出来的Kafka集群将自动开通公网访问。您可以直接通过公网IP的9093端口访问kafka服务。开通9093端口的详细文档，请参见设置安全组访问。

开通已有Kafka集群的公网服务

对于已有集群，Kafka集群部署在VPC网络环境，有以下两种方式可以实现公网环境访问Kafka服务：

挂载弹性公网

1. 登录EMR on ECS控制台。
2. 在**节点管理**页面，单击安全组ID，为安全组开通9093端口，详情请参见设置安全组访问。
3. 为所有ECS实例挂载公网IP。
   i. 在**节点管理**页面，单击ECS ID。
   ii. 在ECS控制台，为当前ECS实例挂载公网，详情请参见绑定EIP。
   iii. 重复步骤i~步骤ii，为Kafka集群的所有ECS实例挂载公网IP。
4. 在E-MapReduce控制台的Kafka服务页面，修改以下配置。
   - **kafka.public-access.ip**：**配置范围**选择**独立节点配置**，选择各个主机，填写对应节点挂载的公网IP地址。



- **kafka.public-access.enable**：修改参数值为true。

5. 部署客户端配置。
   i. 在Kafka服务的**配置**页面，单击**部署客户端配置**。
   ii. 在**配置KAFKA服务**对话框中，输入执行原因，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。
6. 重启Kafka服务。
   i. 在Kafka服务的**配置**页面，选择**更多操作 > 重启**。
   ii. 在**重启KAFKA服务**对话框中，输入执行原因，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

部署高速通道

部署高速通道打通内网和公网网络，详情请参见什么是高速通道。

5.1.3.1.3. 使用SSL加密Kafka链接

本文为您介绍如何配置E-MapReduce Kafka集群的SSL（Secure Sockets Layer）功能，并通过SSL加密Kafka数据链接。

前提条件

已在E-MapReduce控制台创建DataFlow集群（即后面说的Kafka集群），并选择了Kafka服务，详情请参见创建集群。

配置SSL功能

E-MapReduce Kafka集群提供以下两种配置SSL的方式：

- 使用默认证书配置SSL：使用E-MapReduce默认创建的证书和默认配置方式快速启用SSL功能。
- 自定义配置SSL：使用自定义证书和配置值启用SSL功能。

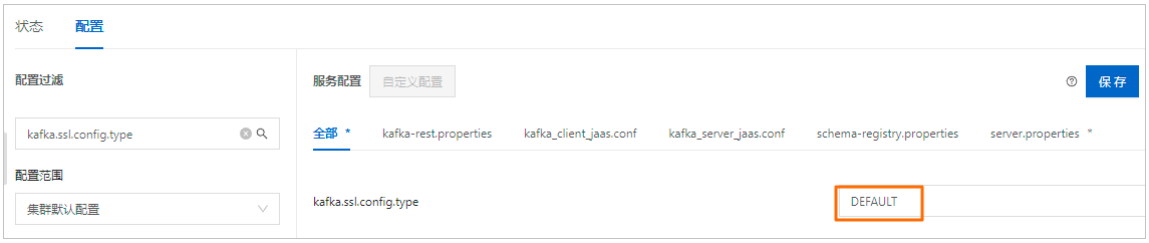
E-MapReduce通过*server.properties*配置文件的kafka.ssl.config.type配置项来管理配置SSL的策略。

使用默认证书配置SSL

> **说明** Kafka集群的SSL功能默认关闭，您可以执行以下步骤快速开启SSL功能。

1. 进入服务的配置页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 在**集群服务**页面，单击Kafka服务区域的**配置**。
2. 修改配置项。
   i. 在**配置过滤**中，输入配置项**kafka.ssl.config.type**，单击图标。

ii. 修改参数值为**DEFAULT**。



3. 保存配置。
   i. 在**服务配置**区域，单击**保存**。
   ii. 在**确认修改配置**对话框中，输入**执行原因**，打开**自动更新配置**开关。
   iii. 单击**确定**。
4. 部署客户端配置。
   i. 在Kafka服务的**配置**页面，单击**部署客户端配置**。
   ii. 在**配置KAFKA服务**对话框中，输入执行原因，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。
5. 重启Kafka服务。
   i. 在Kafka服务的**配置**页面，选择**更多操作 > 重启**。
   ii. 在**重启KAFKA服务**对话框中，输入执行原因，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

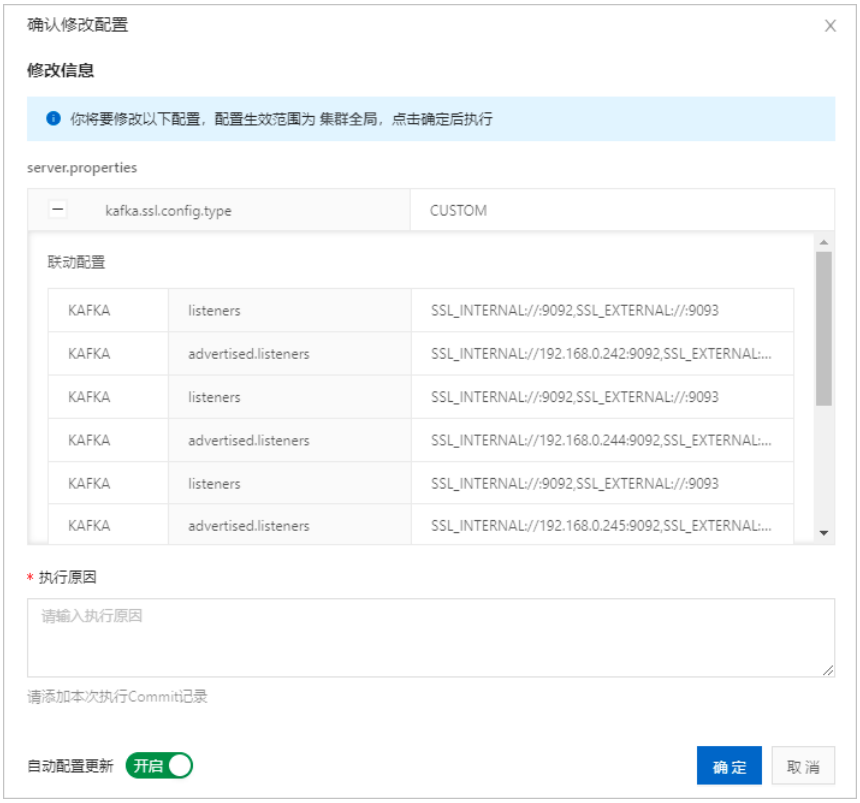
自定义配置SSL

> **说明** Kafka集群的SSL功能默认关闭，您可以通过自定义配置开启SSL功能。

1. 进入服务的配置页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 在**集群服务**页面，单击Kafka服务区域的**配置**。
2. 修改配置项。
   i. 在**配置过滤**中，输入配置项**kafka.ssl.config.type**，单击图标。
   ii. 修改参数值为**CUSTOM**。
3. 保存配置。
   i. 在**服务配置**区域，单击**保存**。

ii. 在**确认修改配置**对话框中，输入**执行原因**，打开**自动更新配置**开关。

确认修改配置对话框中，为您展示了本次配置联动修改的配置项及参数值。



iii. 单击**确定**。

4. 修改SSL其他配置。

   您需要根据业务需求，自行配置除listeners之外的SSL相关配置。例如，ssl.keystore.location，ssl.keystore.password，ssl.truststore.location，ssl.truststore.password，ssl.key.password，ssl.keystore.type和ssl.truststore.type等。

5. 部署客户端配置。

   i. 在Kafka服务的**配置**页面，单击**部署客户端配置**。

   ii. 在**配置KAFKA服务**对话框中，输入执行原因，单击**确定**。

   iii. 在**确认**对话框中，单击**确定**。

6. 重启Kafka服务。

   i. 在Kafka服务的**配置**页面，选择**更多操作 > 重启**。

   ii. 在**重启KAFKA服务**对话框中，输入执行原因，单击**确定**。

   iii. 在**确认**对话框中，单击**确定**。

使用SSL连接Kafka

使用SSL连接Kafka时，需要客户端配置参数security.protocol、ssl.truststore.password和ssl.truststore.password。

例如，在已开启SSL的Kafka集群中，使用Kafka自带的Producer和Consumer执行作业，操作步骤如下：

1. 使用SSH方式登录Kafka集群，详情请参见登录集群。
2. 创建配置文件。

   i. 执行以下命令，创建配置文件*ssl.properties*。

```
vim ssl.properties
```

ii. 添加以下内容至配置文件*ssl.properties*中。

```
security.protocol=SSL
ssl.truststore.location=/var/taihao-security/ssl/ssl/truststore
ssl.truststore.password=${password}
ssl.keystore.location=/var/taihao-security/ssl/ssl/keystore
ssl.keystore.password=${password}
ssl.endpoint.identification.algorithm=
```

上面参数的值，您可以在EMR控制台Kafka服务的配置页面查看。如果是在Kafka集群以外的环境执行作业，您可以将Kafka集群中任意节点相应目录下的*truststore*和*keystore*文件，拷贝至运行环境进行相应配置。

3. 执行以下命令，创建Topic。

```
kafka-topics.sh --partitions 10 --replication-factor 2 --zookeeper master-1-1:2181/emr-kafka --topic test --create
```

4. 执行以下命令，使用SSL配置文件产生数据。

```
export IP=<your_InnerIP>
kafka-producer-perf-test.sh --topic test --num-records 123456 --throughput 10000 --record-size 1024 --producer-props bo
otstrap.servers=${IP}:9092 --producer.config ssl.properties
```

说明 本文代码示例中的`your_InnerIP`为master-1-1节点的内网IP地址。

5. 执行以下命令，使用SSL配置文件消费数据。

```
export IP=<your_InnerIP>
kafka-consumer-perf-test.sh --broker-list ${IP}:9092 --messages 100000000 --topic test --consumer.config ssl.properties
```

5.1.3.1.4. 使用SASL登录认证Kafka服务

本文为您介绍如何配置E-MapReduce Kafka集群的SASL（Simple Authentication and Security Layer）功能，并通过SASL功能登录Kafka集群。

前提条件

已在E-MapReduce控制台创建DataFlow集群（即后面说的Kafka集群），并选择了Kafka服务，详情请参见创建集群。

配置SASL功能

E-MapReduce通过*server.properties*配置文件的kafka.ssl.config.type配置项来管理配置SASL的策略。

Kafka集群的SASL功能默认关闭，您可以执行以下步骤快速开启SASL功能。本文以配置SASL/SCRAM-SHA-512认证机制为例。

1. 进入服务的配置页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 在**集群服务**页面，单击Kafka服务区域的**配置**。
2. 修改listener配置。
   i. 在**配置过滤**中，输入配置项**kafka.ssl.config.type**，单击图标。
   ii. 修改参数值为**CUSTOM**。
   iii. 在**服务配置**区域，单击**保存**。
   iv. 在**确认修改配置**对话框中，输入**执行原因**，打开**自动更新配置**开关，单击**确定**。
   确认修改配置对话框中，为您展示了本次配置联动修改的配置项及参数值。
3. 修改SASL认证机制相关的配置。
   i. 使用SSH方式登录Kafka集群，详情请参见登录集群。
   ii. 执行以下命令，创建admin用户。

```
kafka-configs.sh --zookeeper master-1-1:2181/emr-kafka --alter --add-config 'SCRAM-SHA-256=[password=admin-secret],
SCRAM-SHA-512=[password=admin-secret]' --entity-type users --entity-name admin
```

iii. 设置SASL配置项。

在Kafka服务配置页面的**server.properties**页签，添加配置项。

a. 单击**自定义配置**。

b. 在**新增配置项**对话框中，添加以下配置。

参数	参数值
sasl.mechanism.inter.broker.protocol	SCRAM-SHA-512
sasl.enabled.mechanisms	SCRAM-SHA-512

c. 单击**确定**。

d. 在**确认修改配置**对话框中，输入**执行原因**，打开**自动更新配置**开关，单击**确定**。

iv. 配置服务端JAAS。

- 方式一：通过自定义配置项配置服务端JAAS。

a. 在Kafka服务配置页面，单击**server.properties**页签。

b. 单击**自定义配置**，新增以下配置项。

参数	参数值
listener.name.sasl_plaintext.sasl.enabled.mechanisms	SCRAM-SHA-512
listener.name.sasl_plaintext.scram-sha-512.sasl.jaas.config	org.apache.kafka.common.security.scram.ScramLoginModule required username="admin" password="admin-secret" ;

c. 单击**确定**。

d. 在**确认修改配置**对话框中，输入**执行原因**，打开**自动更新配置**开关，单击**确定**。

- 方式二：通过文本文件配置JAAS。

a. 在Kafka服务配置页面，修改以下配置项。

页签	参数	参数值

kafka_server_jaas.conf	kafka.server.jass.content	`KafkaServer {`
		`org.apache.kafka.common.security.scram.ScramLoginModule required`
		`username="admin"`
		`password="admin-secret";`
		`};`

server.properties	**kafka_opts**	-Djava.security.auth.login.config=/etc/taihao-apps/kafka-conf/kafka-conf/kafka_server_jaas.conf

b. 在**服务配置**区域，单击**保存**。

c. 在**确认修改配置**对话框中，输入**执行原因**，打开**自动更新配置**开关，单击**确定**。

v. 配置客户端JAAS。

通过kafka_client_jaas.conf配置文件的kafka.client.jass.content配置项，配置Kafka客户端JAAS，该配置将会用于启动Kafka Schema Registry以及Kafka Rest Proxy组件。

a. 在Kafka服务配置页面，修改以下配置项。

页签	参数	参数值

kafka_client_jaas.conf	kafka.client.jass.content	KafkaClient {
		org.apache.kafka.common.security.scram.ScramLoginModule required
		username="admin"
		password="admin-secret";
		};

schema_registry_opts	kafka_opts	-Djava.security.auth.login.config=/etc/taihao-apps/kafka-conf/kafka-conf/kafka_client_jaas.conf
kafka-rest.properties	kafkarest_opts	-Djava.security.auth.login.config=/etc/taihao-apps/kafka-conf/kafka-conf/kafka_client_jaas.conf

b. 在**服务配置**区域，单击**保存**。

c. 在**确认修改配置**对话框中，输入**执行原因**，打开**自动更新配置**开关，单击**确定**。

4. 部署客户端配置。

i. 在Kafka服务的**配置**页面，单击**部署客户端配置**。

ii. 在**配置KAFKA服务**对话框中，输入执行原因，单击**确定**。

iii. 在**确认**对话框中，单击**确定**。

5. 重启Kafka服务。

i. 在Kafka服务的**配置**页面，选择**更多操作 > 重启**。

ii. 在**重启KAFKA服务**对话框中，输入执行原因，单击**确定**。

iii. 在**确认**对话框中，单击**确定**。

使用SASL登录Kafka集群

本文以SASL/SCRAM-SHA-512机制为例，使用Kafka自带的Producer和Consumer执行作业，介绍客户端如何认证登录Kafka服务。

1. 使用SSH方式登录Kafka集群，详情请参见登录集群。
2. 执行以下命令，创建用于登录的用户名和密码。

```
kafka-configs.sh --zookeeper master-1-1:2181/emr-kafka --alter --add-config 'SCRAM-SHA-256=[password=client-secret],SCRAM-SHA-512=[password=client-secret]' --entity-type users --entity-name client-user
```

3. 创建配置文件。

i. 执行以下命令，创建配置文件*sasl.properties*。

```
vim sasl.properties
```

ii. 添加以下内容至配置文件*sasl.properties*中。

```
security.protocol=SASL_PLAINTEXT
sasl.mechanism=SCRAM-SHA-512
sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule required username="client-user" password="client-secret";
```

4. 执行以下命令，创建Topic。

```
kafka-topics.sh --partitions 10 --replication-factor 2 --zookeeper master-1-1:2181/emr-kafka --topic test --create
```

5. 执行以下命令，使用SASL配置文件产生数据。

```
kafka-producer-perf-test.sh --topic test --num-records 123456 --throughput 10000 --record-size 1024 --producer-props bootstrap.servers=core-1-1:9092 --producer.config sasl.properties
```

6. 执行以下命令，使用SASL配置文件消费数据。

```
kafka-consumer-perf-test.sh --broker-list core-1-1:9092 --messages 100000000 --topic test --consumer.config sasl.properties
```

5.1.3.2. 高阶使用

5.1.3.2.1. Kafka Rebalancer工具介绍

本文为您介绍使用Kafka Rebalancer工具的注意事项、常用参数以及使用示例。本文以EMR Kafka 2.4.1版本为例。

背景信息

在使用Kafka集群过程中，常常会碰到以下问题：

- leader分区不均衡：导致各个Broker负载不均衡，读写吞吐下降。
- Borker分区数据量的不均衡：导致部分机器的磁盘利用率明显高于集群平均值，增加Broker宕机的风险。
- 节点内磁盘间的利用率不均衡：部分磁盘的利用率明显高于节点磁盘的平均利用率，增加了副本offline甚至Broker宕机的风险。
- 热点Topic：导致相应地磁盘load不均衡。

当出现上述问题时，通常需要进行负载均衡操作。进行负载均衡通常需要进行leader的重新分配和分区reassign等操作。Kafka提供的kafka-preferred-replica-election.sh和kafka-reassign-partitions.sh等工具可以进行此类负载均衡操作，但是这些工具需要进行较多配置操作，增加了运维工作量和难度。

EMR Kafka提供的Rebalancer工具封装了kafka-preferred-replica-election.sh和kafka-reassign-partitions.sh等工具，简化了运维工作量和难度，不会影响相应被封装工具的使用，您仍然可以使用相关工具进行运维工作。

注意事项

- 在使用工具的时候，需要对运维流量进行限制。
- 由于Rebalancer工具根据相应的选项生成reassignment的JSON文件，需要对生成的文件进行确认以确保与预期的分配结果一致。
- 对于较多的分区副本复制和移动操作，您应该评估运维时长以决定是否使用该工具进行运维。如果运维时间过长，您可以直接使用kafka-reassign-partitions.sh工具对运维任务进行拆分以便分时段进行运维。
- 需要借助kafka-reassign-partitions.sh工具对运维的过程进行监控，因此需要注意手工保存reassignment的JSON文件，该文件将用作kafka-reassign-partitions.sh的verify输入参数。

Rebalancer工具功能

在EMR集群Broker的ECS实例上，可以直接输入kafka-rebalancer.sh来查看该脚本工具的功能。

- preferred-election：均衡leadership，需要输入Topic参数、对指定的Topic进行preferred election，详情请参见preferred-election。

  该功能对kafka-preferred-replica-election.sh工具进行了封装。
- balance-disks：基于磁盘空间使用率均衡节点内的磁盘分区副本分配，详情请参见balance-disks。
- rebalance：基于磁盘空间使用率均衡集群节点之间的磁盘分区副本分配，详情请参见rebalance。
- remove-broker-ids：移除指定Broker列表的所有分区副本。移除副本后，可以对Broker进行下线操作，详情请参见remove-broker-ids。

preferred-election

当分区副本leader不在preferred的Broker节点上时，有可能造成节点负载不均衡，此时需要均衡leadership。

重要参数

topics：待preferred election的topics。

示例

触发preferred-election操作：

1. 创建测试Topic。

   ```
   kafka-topics.sh --create --topic elelction-topic --zookeeper master-1-1:2181/emr-kafka --replication-factor 2 --partitions 50
   ```

2. 对测试Topic进行选举操作。

   ```
   kafka-rebalancer.sh --zookeeper master-1-1:2181/emr-kafka --preferred-election --bootstrap-server core-1-1:9092 --topics elelction-topic
   ```

balance-disks

该功能用于均衡节点内的磁盘分区副本分配，是对kafka-reassign-partitions.sh相应功能的封装。与kafka-reassign-partitions.sh不同的地方在于，Rebalancer工具会基于磁盘空间使用率自动生成同一Broker节点内的分区副本分配文件。

重要参数

参数	描述
replica-alter-log-dirs-throttle	限流参数，用来限制Broker内副本log目录迁移过程的迁移流量。 **说明** 您在使用该功能时，请设置合理的限流参数，避免因为资源争抢导致正常业务流量受到影响。
threshold	磁盘使用率偏移阈值，只有当Broker内部的各个磁盘使用率的差异大于该阈值时，才会触发实际的副本Broker内部磁盘迁移操作，默认值为0.1。

示例

1. 使用kafka-rebalancer.sh触发balance disks操作。
   i. 创建测试Topic。
   ```
   kafka-topics.sh --create --topic balance-disks-topic --zookeeper master-1-1:2181/emr-kafka --replication-factor 2 --partitions 50
   ```
   ii. 移动Broker 0 Topic分区副本在磁盘的位置，模拟磁盘不均衡的场景。
   ```
   mv /mnt/disk1/kafka/log/balance-disks-topic-* /mnt/disk2/kafka/log/
   mv /mnt/disk3/kafka/log/balance-disks-topic-* /mnt/disk4/kafka/log/
   ```
   iii. 写入测试数据。
   ```
   kafka-producer-perf-test.sh --producer-props bootstrap.servers=core-1-1:9092 --num-records 70000000 --throughput 200000 --record-size 1000 --topic balance-disks-topic
   ```
   iv. 均衡Broker 0上的磁盘。
   ```
   kafka-rebalancer.sh --bootstrap-server core-1-1:9092 --zookeeper master-1-1:2181/emr-kafka --balance-disks 0 --replica-alter-log-dirs-throttle 50000000 --threshold 0.1
   ```
   保存打印在“Current partition replica assignment”之后的JSON字符串到文件*move.json*中。

   注意
   - 需要保存打印在“Current partition replica movement”之后的JSON字符串到文件*move.json*中，用作之后校验过程的reassignment-json-file的参数值。
   - 由于使用了限流参数，迁移完成后，需要用kafka-reassign-partitions.sh工具verify选项去除设置在Topic以及Broker上的限流config参数。

2. 监控检查balance disk过程。
   您可以使用kafka-configs.sh工具进行限流参数确认，使用kafka-reassign-partitions.sh工具对迁移的进度进行确认。
   i. 查看限流参数是否生效。
   ```
   kafka-configs.sh --zookeeper master-1-1:2181/emr-kafka -entity-type brokers --entity-name 0 --describe
   ```
   ii. 查看balance disk任务。
   ```
   kafka-reassign-partitions.sh --verify --bootstrap-server core-1-1:9092 --zookeeper master-1-1:2181/emr-kafka --reassignment-json-file move.json
   ```
   说明 输入的参数move.json是步骤1中保存的*move.json*文件。

rebalance

该功能基于磁盘空间使用率，均衡集群节点之间的磁盘分区副本分配，是对kafka-reassign-partitions.sh相应功能的封装。与kafka-reassign-partitions.sh不同的地方在于，Rebalancer会基于磁盘的空间使用率自动生成分区副本的节点间分配文件。

重要参数

参数	描述
throttle	限流参数，用来限制reassign过程的迁移流量。 **说明** 您在使用该功能时，请设置合理的限流参数，避免因为资源争抢导致正常业务流量受到影响。

参数	描述
threshold	磁盘使用率偏移阈值，只有当Broker内部的各个磁盘使用率的差异大于该阈值时，才会触发实际的rebalance操作，默认值为10%。

示例

1. 使用kafka-rebalancer.sh触发rebalance操作。
   i. 创建测试Topic。

   ```
   kafka-topics.sh --create --topic rebalance-topic  --zookeeper master-1-1:2181/emr-kafka --replica-assignment 0:1,0:1,0:1,0:1,0:1,0:1,0:1,0:1,0:1,0:1
   ```

   ii. 写入测试数据。

   ```
   kafka-producer-perf-test.sh --topic rebalance-topic --num-records 7000000 --throughput 200000 --producer-props bootstrap.servers=core-1-1:9092 --record-size 1000
   ```

   iii. 查看rebalance disk任务。

   ```
   kafka-rebalancer.sh --bootstrap-server core-1-1:9092 --zookeeper master-1-1:2181/emr-kafka --rebalance --throttle 100000000 --threshold 0.1
   ```

   保存打印在“Current partition replica assignment”之后的JSON字符串到文件*move.json*中。

   > **注意**
   > - 需要保存打印在“Current partition replica movement”之后的JSON字符串到文件*move.json*中，用作之后校验过程的reassignment-json-file的参数值。
   > - 由于使用了限流参数，迁移完成后，需要用kafka-reassign-partitions.sh工具verify选项去除设置在Topic以及Broker上的限流config参数。

2. 监控检查rebalance过程。

   您可以使用kafka-configs.sh工具进行限流参数确认，使用kafka-reassign-partitions.sh工具对迁移的进度进行确认。
   i. 查看限流参数是否生效。

   ```
   kafka-configs.sh --zookeeper master-1-1:2181/emr-kafka -entity-type brokers --entity-name 0 --describe
   kafka-configs.sh --zookeeper master-1-1:2181/emr-kafka -entity-type topics --entity-name decommission-topic --describe
   ```

   ii. 查看rebalance任务。

   ```
   kafka-reassign-partitions.sh --verify --bootstrap-server core-1-1:9092 --zookeeper master-1-1:2181/emr-kafka --reassignment-json-file move.json
   ```

   > **说明** 输入的参数move.json是步骤1中保存的*move.json*文件。

remove-broker-ids

该功能用于移除指定Broker列表的所有分区副本。当您想要下线某个节点的时候，可以使用该功能，先将该节点上的所有分区副本移到其他Broker节点上。

通常Topic分区会被设置成3副本，因此一个集群通过需要保留3个Broker节点，不建议您在4个以下Broker节点规模的Kafka集群上执行下线Broker节点的操作。

重要参数

throttle：限流参数，用来限制reassign过程的迁移流量。

> **说明** 您在使用该功能时，请设置合理的限流参数，避免因为资源争抢导致正常业务流量受到影响。

示例

1. 使用kafka-rebalancer.sh触发remove Broker操作。
   i. 创建测试Topic。

   ```
   kafka-topics.sh --create --topic decommission-topic --partitions 50 --replication-factor 2 --zookeeper master-1-1:2181/emr-kafka
   ```

ii. 写入测试数据。

```
kafka-producer-perf-test.sh --topic decommission-topic --num-records 70000000 --throughput 200000 --producer-props bootstrap.servers=core-1-1:9092 --record-size 1000
```

iii. 将Broker 1上的分区副本都移除掉。

```
kafka-rebalancer.sh --bootstrap-server core-1-1:9092 --zookeeper master-1-1:2181/emr-kafka --remove-broker-ids 1 --throttle 50000000
```

保存打印在“Current partition replica assignment”之后的JSON字符串到文件*move.json*中。

> **注意**
> - 需要保存打印在“Current partition replica movement”之后的JSON字符串到文件*move.json*中，用作之后校验过程的reassignment-json-file的参数值。
> - 由于使用了限流参数，迁移完成后，需要用kafka-reassign-partitions.sh工具verify选项去除设置在Topic以及Broker上的限流config参数。

2. 监控检查remove过程。

您可以使用kafka-configs.sh工具进行限流参数确认，使用kafka-reassign-partitions.sh工具检查分区副本迁移过程，使用kafka-log-dirs.sh工具查看待remove的Broker节点上的分区副本是否已经都迁移到其他节点。

i. 查看限流参数是否生效。

```
kafka-configs.sh --zookeeper master-1-1:2181/emr-kafka -entity-type brokers --entity-name 0 --describe
kafka-configs.sh --zookeeper master-1-1:2181/emr-kafka -entity-type topics --entity-name decommission-topic --describe
```

ii. 查看rebalance任务。

```
kafka-reassign-partitions.sh --verify --bootstrap-server core-1-1:9092 --zookeeper master-1-1:2181/emr-kafka --reassignment-json-file move.json
```

> **说明** 输入的参数move.json是步骤1中保存的*move.json*文件。

iii. remove完成后，确保对应Broker上已经没有分区副本。

```
kafka-log-dirs.sh --bootstrap-server core-1-1:9092 --broker-list 1 --describe
```

5.1.3.3. Kafka运维

5.1.3.3.1. EMR Kafka磁盘故障运维

磁盘故障及其运维通常伴随着磁盘上的数据销毁。在进行磁盘运维时，您应考虑数据是否需要迁移备份。对于Kafka集群，您还需要考虑Topic分区副本数据是否可以从其他Broker节点分区副本同步恢复。本文以EMR Kafka 2.4.1版本为例，介绍Kafka磁盘故障运维的操作。

业务场景

Kafka将日志数据存储到磁盘中，当磁盘出现故障时，会导致磁盘IO能力下降、集群吞吐下降、消息读写延时或日志目录offline等问题。这些情况有可能影响到线上业务平稳运行、数据丢失、Kafka集群容错能力下降，单块盘故障甚至有可能因为IO处理能力下降导致集群出现雪崩效应、引起重大生产事故。因此需要对磁盘故障进行有效监控以便及时发现故障。当磁盘发生故障时，应及时完成相关故障的处理，及时恢复集群的容错能力。

磁盘运维概述

本文从磁盘监控和磁盘故障恢复角度来介绍磁盘运维策略。

磁盘监控

以下内容从Kafka服务层面以及ECS系统层面来简单了解一下磁盘的监控策略。

- Kafka服务层面：可以在云监控系统中设置EMR Kafka集群的OfflineLogDirectoryCount和UnderReplicatedPartitions等指标告警，及时发现相关指标的异常。
- ECS系统层面：可以在云监控中设置相应ECS实例的I/O wait和mbps等指标，来监控磁盘的健康状态。ECS后台也会自动的检测磁盘状态，当发现问题时，会自动为您推送相关的磁盘事件。

磁盘故障恢复

当出现log directory offline、Under Replicated Partition时，需要尽快定位是否是由于磁盘故障导致的。

当出现磁盘故障时，需要根据故障原因、故障影响程度、业务需求（是否接受数据丢失、是否允许服务较长时间不可用）、集群状态等综合考虑恢复采取的策略。

- 如果业务优先保证服务可用，但允许丢失部分数据，则应考虑在可能会丢失数据的情况下，先恢复服务可用性。
- 如果业务不允许数据丢失，但能容忍服务较长时间不可用，则需要考虑如何尽可能的避免丢失数据的情况出现。
- 如果业务需要高可用与数据不丢失，则您需要通过合理的集群配置、使用Kafka方式提高系统容错能力来避免出现一块盘故障就导致数据会丢失的情况出现。
- 如果发现因为故障盘IO性能下降导致集群整体性能下降，影响业务，则应快速隔离故障盘来进行业务止损。

当检测或定位到磁盘故障时，可以考虑如下运维策略：

- Broker迁移：将Broker迁移到新的ECS实例，适合单个Broker只有一块数据盘的场景，详情请参见Broker迁移方式。
- 节点间分区迁移：将故障磁盘中的分区副本迁移到其他Broker，详情请参见节点间分区迁移方式。
- 节点内分区迁移：将故障磁盘中的分区副本迁移到当前Broker的其他磁盘，详情请参见节点内分区迁移方式。
- 原Broker数据恢复：将异常磁盘从*log.dirs*中移除，重启Broker后会自动恢复丢失的Partition数据到本节点的其他目录，详情请参见原Broker数据恢复方式。

注意事项

磁盘故障运维应注意以下信息：

- 选择修复策略：综合各种因素考虑选择磁盘维修的策略。
- 注意限流：在磁盘运维过程当中，如果涉及到数据复制迁移，需要注意限制运维流量，避免对正常业务流量造成影响。
- 限流阈值：应根据磁盘IO能力和正常业务流量来评估迁移时的限流阈值。
- 分区副本迁移：物理盘维修完毕上线之后，应将Topic分区数据迁移回原磁盘，leader负载迁移回原Broker节点。
- 数据迁移时长：在磁盘运维过程当中，如果涉及到数据复制迁移，需要评估数据迁移时长对业务的影响。

Broker迁移方式

方案描述

当发生磁盘故障时，可以将挂载故障盘的Broker迁移到新的ECS实例。此方案的优点在于不需要等待故障磁盘下线、维修、上线周期，故障处理周期短。

适用场景

适合单个Broker只有一块数据盘的场景。

如果Broker上数据较少、恢复速度较快、集群负载较低，您也可以选择此方式来进行故障节点的运维。

> **说明** 如果坏盘为disk1，建议采用Broker迁移方式进行坏盘处理。

注意事项

- Broker迁移会丢失所有的磁盘数据，需要确认业务数据可以从其他Broker恢复（即确认故障磁盘所在Broker上的分区leader可以切换到其它Broker）或者业务数据允许丢失。
- Broker迁移时，需要根据实际情况评估是否设置限流参数，避免对正常业务造成影响。

操作步骤

请提交工单处理。

节点间分区迁移方式

方案描述

当发现磁盘故障隐患或者由于其他原因需要替换磁盘时，可以将故障磁盘上的分区副本数据迁移到其他磁盘。故障磁盘维修更换期间，分区副本个数不会变化、集群处于正常容错状态。故障盘维修更换完毕之后将分区数据迁回到原磁盘。此方案的优点在于磁盘运维周期，集群处于正常容错状态，各Broker负载较为均衡；缺点是当故障盘上的分区没有其他ISR副本时，有可能丢失数据。

适用场景

故障盘所在Broker磁盘容量不足场景，或者故障盘所在Broker负载过高的场景。

注意事项

- 注意限流：分区数据迁移将产生较大的数据迁移流量，需要对流量进行限制以避免对正常业务产生影响。
- 迁移时长：如果需要迁移的数据较多，相应的迁移时长也会加大。
- 限流阈值：应根据磁盘IO能力、正常业务流量来评估迁移时的限流阈值。
- 磁盘空间：需要确保集群有足够的磁盘容量来存放迁移的数据。
- 备份原始reassignment文件。分区迁移时，需要注意保存原来的reassignment文件，便于磁盘修复好后，将分区迁移回原来的磁盘目录。

操作步骤

本示例以Broker 0的坏盘*/mnt/disk7*为例，介绍如何采用节点间分区迁移的方式进行故障盘运维。

1. 分区迁移。
   i. 以SSH方式登录到源Kafka集群的Master节点，详情请参见登录集群。
   ii. （可选）待维修磁盘隔离。如果磁盘已经发生故障，您需要执行以下命令快速隔离该磁盘。
   ```
   chmod 000 /mnt/disk7
   ```
   iii. 执行以下命令，将待维修磁盘上的分区迁移到其他Broker节点。
   ```
   kafka-reassign-partitions.sh --zookeeper  master-1-1:2181/emr-kafka --reassignment-json-file reassign.json --throttle 30000000 --execute
   ```
   > **说明** reassign.json包含了待迁移磁盘上需要迁移的分区副本。

   执行完reassign命令后，会输出原始的reassignment，可以将该输出保存起来，便于后续恢复。
   iv. 执行以下命令，确认故障盘上数据已经迁移完毕。
   ```
   ls -lrt /mnt/disk7/kafka/log
   ```
   通过回显信息，查看日志目录下是否有日志文件，没有日志文件表示数据已经迁移完毕。
   v. （可选）如果前面步骤中没有隔离待维修磁盘，则需要执行以下命令隔离待维修磁盘。
   ```
   chmod 000 /mnt/disk7
   ```
2. 卸载故障磁盘目录。
   > **注意** 编辑fstab条目时，直接删除而不是注释掉故障盘条目。

   i. 执行以下命令，编辑*/etc/fstab*。
   ```
   vim /etc/fstab
   ```
   ii. 删除disk7的信息。
   iii. 执行以下命令，验证目录句柄是否释放。
   ```
   lsof +D /mnt/disk7
   ```
   如果发现句柄长时间（半个小时以上）无法释放，则可以通过控制台重启对应Broker服务。
   iv. 执行以下命令，卸载故障磁盘目录。
   ```
   umount /mnt/disk7
   ```
3. 修改*log.dirs*配置并重启服务。
   i. 在EMR控制台，修改恢复后的Broker节点级别的*log.dirs*配置项，移除故障磁盘对应的分区目录。
   本示例需要移除的目录为*/mnt/disk7/kafka/log*。
   ii. 在EMR控制台重启Broker服务，详情请参见重启服务。
   > **说明** 如果ECS的修复磁盘事件运维流程中需要重启ECS实例，则可以将重启Broker服务与重启ECS实例结合起来完成。
4. 通过修复磁盘事件，在ECS控制台进行后续磁盘修复工作。
   此过程可能需要的时间周期为几天。
5. 磁盘上线，详情请参见磁盘上线。
   您需要将修复后的磁盘重新mount到原来的目录。本示例为*/mnt/disk7*目录。
6. 在修复后的磁盘目录上创建kafka日志目录。
   ```
   //创建原始日志目录。本例子为/mnt/disk7/kafka/log。
   sudo mkdir -p /mnt/disk7/kafka/log
   sudo chown -R kafka:hadoop /mnt/disk7/kafka/log
   ```
7. 修改*log.dirs*配置并重启服务。
   i. 在EMR控制台，修改恢复后的Broker节点级别的*log.dirs*配置项，添加修复后磁盘对应的分区目录。
   本示例需要增加的目录为*/mnt/disk7/kafka/log*。
   ii. 在EMR控制台重启Broker服务，详情请参见重启服务。
8. 迁回分区副本。
   使用步骤1保存的原始assignment文件，将原来故障盘上的分区迁回到修复后的磁盘。

原Broker数据恢复方式

方案描述

当磁盘故障时，如果磁盘IO性能已经明显下降，则需要快速隔离故障磁盘避免因单点故障影响集群性能。

磁盘隔离之后，对应kafka日志目录处于offline状态。此时，如果分区存在ISR副本或者允许分区数据丢失，可以直接将故障磁盘从log.dirs中删除下线，然后重启Broker。重启Broker后，如果其他节点存在新的leader副本，原有故障盘分区副本将在其所在Broker进行恢复。磁盘修复上线后，您可以通过reassign工具将原有数据迁移回修复后的磁盘。

本方案的缺点在于：当故障盘上的分区没有ISR副本时，有可能丢失数据。

适用场景

- 故障磁盘IO性能已经明显下降，需要快速进行故障盘隔离的场景。
- 故障磁盘上的日志目录已经offline的场景。
- 故障盘上的分区在其他节点已经没有ISR副本时，允许丢失分区数据。

注意事项

- 流量限制：由于数据在同一个节点恢复，会产生副本恢复流量，应注意限流。
- 磁盘空间：需要注意本Broker的其他节点是否有足够的空间容纳异常磁盘的数据。
- 数据迁移：磁盘修复重新上线后，通常需要将数据从其他磁盘挪回到修复后的磁盘以保证磁盘负载均衡。

操作步骤

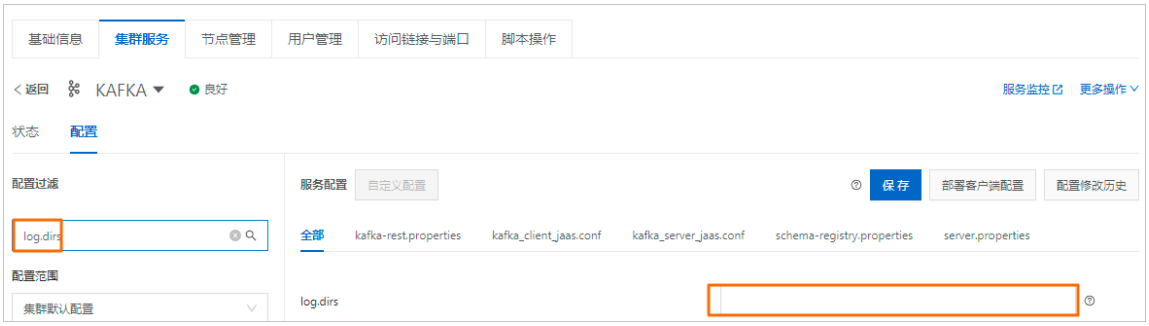
以下示例以Broker 0的*/mnt/disk7*坏盘为例，介绍如何采用原地数据恢复方式进行故障盘运维。

1. 故障盘隔离。
   i. 以SSH方式登录到源Kafka集群的Master节点，详情请参见登录集群。
   ii. 执行以下命令，隔离故障盘。
   ```
   chmod 000 /mnt/disk7
   ```
2. （可选）备份故障盘上Kafka日志分区目录名称。
   通过备份故障盘目录名称，当磁盘修复上线后，您可以将原有分区迁移回修复后的磁盘中。
3. 卸载故障磁盘目录。
   > **注意** 编辑fstab条目时，直接删除而不是注释掉故障盘条目。
   i. 执行以下命令，编辑*/etc/fstab*。
   ```
   vim /etc/fstab
   ```
   ii. 删除disk7的信息。
   iii. 执行以下命令，验证目录句柄是否释放。
   ```
   lsof +D /mnt/disk7
   ```
   如果发现句柄长时间（半个小时以上）无法释放，则可以通过控制台重启对应Broker服务。
   iv. 执行以下命令，卸载故障磁盘目录。
   ```
   umount /mnt/disk7
   ```
4. 修改故障Broker节点的*log.dirs*配置并限制恢复流量。
   i. 在EMR控制台，修改恢复后的Broker节点级别的*log.dirs*配置项，移除故障磁盘对应的分区目录。
   本示例需要移除的目录为*/mnt/disk7/kafka/log*。
   ii. （可选）通过设置reassign限流参数的方式来限制Broker上分区数据同步恢复时候的流量带宽，详情请参见限制Kafka服务端运维流量。
5. 在EMR控制台重启Broker服务，详情请参见重启服务。
   > **说明** 如果ECS的修复磁盘事件运维流程中需要重启ECS实例，则可以将重启Broker服务与重启ECS实例结合起来完成。
   在重启的过程中，Kafka将会恢复本Broker故障盘上缺失的副本分区到其他磁盘，待Kafka恢复好分区数据后，如果前一步骤设置了限流参数，则需要将限流参数去除掉。
6. 通过修复磁盘事件，在ECS控制台进行后续磁盘修复工作。
   此过程可能需要的时间周期为几天。
7. 磁盘上线，详情请参见磁盘上线。
   您需要将修复后的磁盘重新mount到原来的目录。本示例为*/mnt/disk7*目录。

8. 创建Kafka日志目录并重启服务。
   i. 在修复后的磁盘目录上创建Kafka日志目录。

```
//创建原始日志目录。本例子为/mnt/disk7/kafka/log。
sudo mkdir -p /mnt/disk7/kafka/log
sudo chown -R kafka:hadoop /mnt/disk7/kafka/log
```

   ii. 在EMR控制台，修改恢复后的Broker节点级别的*log.dirs*配置项，添加修复后磁盘对应的分区目录。



   本示例需要增加的目录为*/mnt/disk7/kafka/log*。
   iii. 在EMR控制台重启Broker服务，详情请参见重启服务。
9. 迁回分区副本。
   使用步骤2保存的分区副本目录，通过reassign工具迁移分区副本broker内部目录的方式迁回到修复后的磁盘。

节点内分区迁移方式

方案描述

当发现磁盘故障时，如果磁盘IO性能已经明显下降，需要快速隔离故障磁盘避免因单点故障影响集群性能。

磁盘隔离之后，对应的kafka日志目录处于offline状态。此时，如果故障盘上存在无法切换到其他Broker的副本且业务数据不能丢失，则可以将分区副本数据迁移到本Broker的其他磁盘后，再进行后续的磁盘运维流程。本方案用于处理如果使用其他运维方案会丢失数据，但业务不允许数据丢失的场景。

适用场景

故障盘上的分区在其他节点已经没有ISR副本，并且业务不允许数据丢失的场景。

注意事项

需要评估OS系统层面的mv操作产生的IO热点的影响。

操作步骤

本示例以Broker 0的坏盘*/mnt/disk7*为例，介绍如何采用节点内分区迁移的方式进行故障盘运维。

1. 故障盘隔离。
   i. 以SSH方式登录到源Kafka集群的Master节点，详情请参见登录集群。
   ii. 执行以下命令，隔离故障盘。

```
chmod 000 /mnt/disk7
```

2. （可选）备份故障盘上Kafka日志分区目录名称。
   通过备份故障盘目录名称，当磁盘修复上线后，您可以将原有分区迁移回修复后的磁盘中。
3. 迁移节点内分区。
   将*/mnt/disk7/kafka/log*下的所有数据以分区为单位，按照分区占用磁盘容量的大小，均匀的迁移到当前节点的其他目录，详情请参见节点内分区迁移方式恢复。
4. 卸载故障磁盘目录。

> 注意　编辑fstab条目时，直接删除而不是注释掉故障盘条目。

   i. 执行以下命令，编辑*/etc/fstab*。

```
vim /etc/fstab
```

   ii. 删除disk7的信息。

iii. 执行以下命令，验证目录句柄是否释放。

```
lsof +D /mnt/disk7
```

如果发现句柄长时间（半个小时以上）无法释放，则可以通过控制台重启对应Broker服务。

iv. 执行以下命令，卸载故障磁盘目录。

```
umount /mnt/disk7
```

5. 在EMR控制台重启Broker服务，详情请参见重启服务。

> **说明** 如果ECS的修复磁盘事件运维流程中需要重启ECS实例，则可以将重启Broker服务与重启ECS实例结合起来完成。

6. 通过修复磁盘事件，在ECS控制台进行后续磁盘修复工作。

此过程可能需要的时间周期为几天。

7. 磁盘上线，详情请参见磁盘上线。

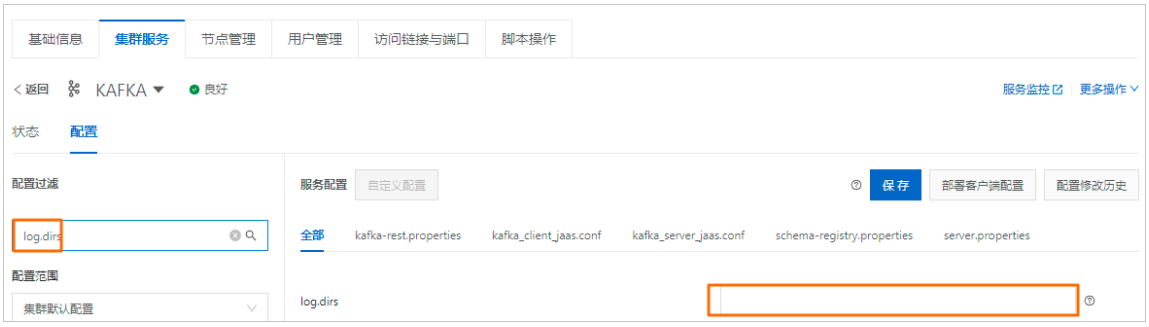
您需要将修复后的磁盘重新mount到原来的目录。本示例为*/mnt/disk7*目录。

8. 创建Kafka日志目录并重启服务。

i. 在修复后的磁盘目录上创建kafka日志目录。

```
//创建原始日志目录。本例子为/mnt/disk7/kafka/log。
sudo mkdir -p /mnt/disk7/kafka/log
sudo chown -R kafka:hadoop /mnt/disk7/kafka/log
```

ii. 在EMR控制台，修改恢复后的Broker节点级别的*log.dirs*配置项，添加修复后磁盘对应的分区目录。



iii. 在EMR控制台重启Broker服务，详情请参见重启服务。

9. （可选）迁回分区副本。

使用步骤2保存的分区副本目录，通过reassign工具迁移分区副本broker内部目录的方式迁回到修复后的磁盘。

5.1.3.3.2. EMR Kafka磁盘写满运维

本文以EMR Kafka 2.4.1版本为例，介绍Kafka磁盘写满时的运维操作。

业务场景

Kafka将日志数据存储到磁盘中，当磁盘写满时，相应磁盘上的Kafka日志目录会出现offline问题。此时，该磁盘上的分区副本不可读写，降低了分区的可用性与容错能力，同时由于leader迁移到其他Broker，增加了其他Broker的负载。因此，当磁盘出现写满情况时，应及时处理。

磁盘写满运维概述

本文从磁盘写满监控和磁盘写满恢复角度来介绍磁盘写满运维策略。

磁盘写满监控

Kafka服务层面：可以在云监控系统中设置EMR Kafka集群的OfflineLogDirectoryCount指标告警，及时发现日志目录offline的情况。

磁盘写满恢复

从排查问题角度来看，当Kafka出现log directory offline时，需要尽快定位是否是由于日志目录所在磁盘写满导致的。

当日志目录磁盘空间被写满时，您可以考虑如下运维策略：

- 磁盘扩容：通过扩容云盘的方式来增加磁盘容量，适用于Broker挂载云盘的场景，详情请参见磁盘扩容方式恢复。
- 节点内分区迁移：将写满磁盘中的分区迁移到本节点的其他磁盘,适用于本Broker节点内磁盘使用率不均衡的场景，详情请参见节点内分区迁移方式恢复。
- 数据清理：清理写满磁盘的日志数据，适用于旧数据可以删除的场景，详情请参见数据清理方式恢复。

磁盘扩容方式恢复

方案描述

磁盘扩容方式是指当Broker磁盘空间被写满时，通过扩容磁盘存储的方式来满足磁盘需求。其优点在于操作简单、风险小、能够快速解决磁盘空间不足的问题。

适用场景

适用于Broker挂载云盘的场景。

操作步骤

在EMR控制台扩容Broker节点的数据盘，详情请参见扩容磁盘。

节点内分区迁移方式恢复

方案描述

当Broker磁盘容量被写满时，对应的log directory被offline，无法使用*kafka-reassign-partitions.sh*工具迁移分区。此时，可以通过ECS实例层面的操作，将分区副本数据挪到当前Broker的其他磁盘并修改相应Kafka数据目录元数据的方式来解决故障盘空间不足的问题。

适用场景

故障磁盘所在Broker使用容量不均衡、存在空间使用率较低的磁盘。

注意事项

- 该方法只能进行节点内部磁盘迁移。
- 分区迁移有可能导致磁盘的IO热点，进而影响集群的性能。需要评估每次迁移数据的大小、迁移时长对业务的影响程度。
- 由于该方法是非标操作，请在相应的Kafka版本测试后再用于生产集群。

操作步骤

当磁盘写满时，由于log directory已经offline了，所以不能用*kafka-reassign-partitions.sh*工具进行分区迁移。本文通过非标方式直接移动文件、修改Kafka相关元数据的方式来迁移分区。

1. 创建测试Topic。
   i. 以SSH方式登录到源Kafka集群的Master节点，详情请参见登录集群。
   ii. 执行以下命令，创建测试Topic，分区副本分布在Broker 0，1节点。

   ```
   kafka-topics.sh --zookeeper master-1-1:2181/emr-kafka --topic test-topic --replica-assignment 0:1 --create
   ```

   您可以通过以下命令查看Topic详情。

   ```
   kafka-topics.sh --zookeeper  master-1-1:2181/emr-kafka --topic test-topic --describe
   ```

   返回如下信息，此时Broker 0是在Isr列表中的。

   ```
   Topic: test-topic       PartitionCount: 1       ReplicationFactor: 2    Configs:
           Topic: test-topic       Partition: 0    Leader: 0       Replicas: 0,1   Isr: 0,1
   ```

2. 执行以下命令，模拟数据写入。

   ```
   kafka-producer-perf-test.sh --topic test-topic --record-size 1000 --num-records 600000000 --print-metrics --throughput
   10240 --producer-props linger.ms=0 bootstrap.servers=core-1-1:9092
   ```

3. 修改Broker 0分区对应的日志目录权限。
   i. 在Master节点上切换到emr-user账号。

   ```
   su emr-user
   ```

   ii. 免密码登录到对应的Core节点。

   ```
   ssh core-1-1
   ```

   iii. 通过sudo获得root权限。

   ```
   sudo su - root
   ```

   iv. 执行以下命令，查找分区所在磁盘。

   ```
   sudo find / -name test-topic-0
   ```

   返回信息如下，则表示分区在*/mnt/disk4/kafka/log*目录下。

   ```
   /mnt/disk4/kafka/log/test-topic-0
   ```

v. 执行以下命令，将Broker 0分区对应的日志目录权限设置成000。

```
sudo chmod 000 /mnt/disk4/kafka/log
```

vi. 执行以下命令，查看test-topic的状态。

```
kafka-topics.sh --zookeeper  master-1-1:2181/emr-kafka --topic test-topic --describe
```

返回信息如下，可以看到Broker 0已经不在Isr列表中了。

```
Topic: test-topic       PartitionCount: 1       ReplicationFactor: 2    Configs:
        Topic: test-topic       Partition: 0    Leader: 1       Replicas: 0,1   Isr: 1
```

4. 停止Broker 0节点。

   在EMR控制台停止Broker 0的Kafka服务。

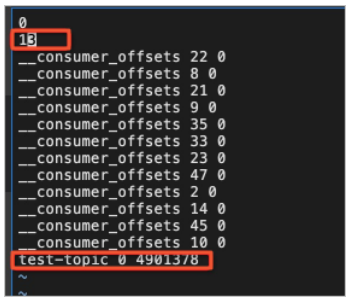
5. 执行以下命令，将Broker 0的test-topic的分区移动到本节点的其他磁盘。

```
mv /mnt/disk4/kafka/log/test-topic-0 /mnt/disk1/kafka/log/
```

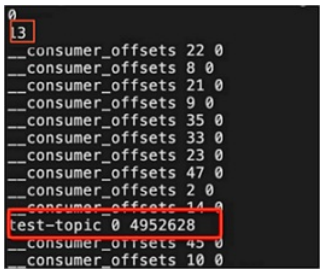
6. 修改文件。

   根据源目录/mnt/*disk4/kafka/log*和目标目录*/mnt/disk1/kafka/log*的元数据文件。需要修改的文件包括*replication-offset-checkpoint*和*recovery-point-offset-checkpoint*。

   - 修改*replication-offset-checkpoint*文件，将*test-topic*相关的条目从原日志目录下的*replication-offset-checkpoint*文件挪到目标日志目录的*replication-offset-checkpoint*中，并修改该文件条目的数量。

   - 修改*recovery-point-offset-checkpoint*文件，将*test-topic*相关的条目从原日志目录下的*recovery-point-offset-checkpoint*文件挪到目标日志目录的*replication-offset-checkpoint*中，并修改该文件条目的数量。

7. 执行以下命令，将源broker 0的日志目录权限设置成正确的权限。

```
sudo chmod 755 /mnt/disk4/kafka/log
```

8. 启动Broker 0节点。

   在EMR控制台启动Broker 0的Kafka服务。

9. 执行以下命令，观察集群的状态是正常的。

```
kafka-topics.sh --zookeeper  master-1-1:2181/emr-kafka --topic test-topic --describe
```

数据清理方式恢复

方案描述

数据清理是指当磁盘被写满时，将业务日志数据（非Kafka内部Topic数据）按照从旧到新的方式删除，直到释放出足够的空间。

适用场景

写满磁盘存在允许删除的业务数据。

如果不改变数据的存储时长，数据有可能又被快速的写满。因此，通常适用于因特殊情况而导致的数据突增的场景。

注意事项

不能删除Kafka内部的Topic数据，即不能删除以下划线（_）开头命名的Topic。

操作步骤

1. 登录到相应的机器上。
2. 找到写满的磁盘，删除掉不需要的业务数据。

   数据清理原则：

   - 不可直接删除Kafka的数据目录，避免造成不必要的数据丢失。
   - 找到占用空间较多或者明确不需要的Topic，选择其中某些Partition，从最早的日志数据开始删除。删除segment及相应地index和timeindex文件。不要清理内置的Topic，例如__consumer_offsets和_schema等。
3. 重启磁盘被写满的相应的Broker节点，使日志目录online。

5.1.3.3.3. EMR Kafka ECS磁盘事件处理

本文为您介绍如何修复EMR Kafka集群Broker节点的ECS实例存在的磁盘事件。

背景信息

当您收到提示本地磁盘硬件异常风险的邮件时，需要进行Kafka服务磁盘修复操作。

此时，您可以在ECS的控制台，查看修复磁盘事件流程。

事件处理概述

EMR Kafka Broker节点的ECS磁盘事件处理与响应和普通的ECS实例不完全相同，在ECS磁盘事件流程的不同阶段，需要穿插Kafka Broker相应的运维操作。整个流程大致可以分为磁盘下线、等待ECS磁盘修复和磁盘上线3个阶段。

本文以故障盘的设备名*/dev/vdh*，目标挂载目录*/mnt/disk7*为例，介绍如何运维EMR Kafka Broker节点的磁盘事件。

> **说明** ECS修复磁盘详情，请参见隔离损坏的本地盘（控制台）。

故障磁盘下线

1. 选择合适的磁盘恢复策略并执行相关操作。

   当ECS磁盘事件处于**修改配置**阶段时，您需要选择合适的EMR Kafka磁盘恢复策略进行Kafka磁盘运维操作，详细的运维策略请参见EMR Kafka磁盘故障运维。

   按照Kafka磁盘恢复策略进行相关操作之后，单击ECS磁盘事件流程中**修改配置**中的**下一步**至**重启**阶段。如果没有**重启**阶段，则处理至**新盘插入中**阶段。

   进行此步骤时，需注意以下信息：

   - 相关的修复策略通常会包含隔离坏盘操作，隔离坏盘之后，重启Broker之前，您需要将坏盘目录路径移出Kafka的*log.dirs*配置中。
   - 如果修复策略要求重启Broker，且ECS磁盘事件流程中也要求重启ECS实例，则可以将两者合并操作。
2. （可选）重启ECS实例。

   如果ECS修复磁盘事件流程中，要求重启ECS，则可以按照如下步骤重启：

   i. 在EMR控制台停止故障Broker节点服务。

   ii. 重启故障ECS实例。

   在ECS修复磁盘事件流程中，单击**重启**。

   iii. 在EMR控制台启动故障Broker节点服务。

   观察Broker节点状态是否处于正常状态。

   当执行完ECS实例重启后，**修复磁盘**事件处于**新盘插入中**阶段。

3. 等待ECS修复磁盘。

   在**新盘插入中**阶段，单击**确定**。确认后请等待阿里云在所宿物理机上更换损坏的本地盘，维护操作通常在五个工作日内完成，之后您会收到恢复磁盘的事件通知。

磁盘上线

收到恢复磁盘的事件通知后，您可以进行磁盘上线操作。

1. 确认磁盘当前状态。
   i. 使用SSH方式登录Kafka集群，详情请参见登录集群。
   ii. 执行以下命令，确认磁盘已经插入，且磁盘大小为1 MB。

   ```
   lsblk
   ```

2. 在ECS控制台，单击修复磁盘事件流程中的**恢复磁盘**，等待流程进入下一阶段。

3. （可选）重启ECS实例。

   如果您的引导页面出现重启阶段，则需要进行重启。请按照如下步骤重启：
   i. 在EMR控制台停止故障Broker节点服务。
   ii. 重启故障ECS实例。

   在修复磁盘事件流程中，单击**重启**。
   iii. 在EMR控制台启动故障Broker节点服务。

   观察Broker节点状态是否处于正常状态。

4. 在故障节点执行以下命令，查看磁盘大小是否恢复。

   ```
   lsblk
   ```

5. 重启完成后，在修复磁盘事件流程中，单击**完成**。

   > **说明** ECS磁盘恢复流程完成后，您仍需要进行后续操作，完成Kafka Broker节点日志目录的恢复。

6. 格式化磁盘和挂载磁盘。
   i. 执行以下命令，格式化磁盘。

   ```
   which  mkfs.ext4
   mkfs.ext4 -m 0 /dev/***
   ```

   > **说明** 请替换 `/dev/***` 为您实际的设备名。

   ii. 执行以下命令，配置*/etc/fstab*文件。

   ```
   echo "/dev/*** /mnt/*** ext4 defaults,noatime,nofail 0 0 " >> /etc/fstab
   ```

   > **说明** 请替换 `/dev/***` 和 `/mnt/***` 为您实际的设备名和挂载目录。

iii. 执行以下命令，确认*/etc/fstab*文件是否配置正确。

```
more /etc/fstab
```

根据以下回显信息，可以确认*/etc/fstab*文件已正确配置。

```
/dev/vdi /mnt/disk8 ext4 defaults,noatime,nofail 0 0
/dev/vdc /mnt/disk2 ext4 defaults,noatime,nofail 0 0
/dev/vdd /mnt/disk3 ext4 defaults,noatime,nofail 0 0
/dev/vde /mnt/disk4 ext4 defaults,noatime,nofail 0 0
/dev/vdf /mnt/disk5 ext4 defaults,noatime,nofail 0 0
/dev/vdg /mnt/disk6 ext4 defaults,noatime,nofail 0 0
/dev/vdb /mnt/disk1 ext4 defaults,noatime,nofail 0 0
/dev/vdh /mnt/disk7 ext4 defaults,noatime,nofail 0 0
~
```

iv. 执行以下命令，挂载磁盘。

```
mount /dev/** /mnt/***
```

请替换设备名和挂载目录。本文示例是将*/dev/vdh*挂载至*/mnt/disk7*，所以命令为 `mount /dev/vdh /mnt/disk7` 。

v. 执行以下命令，确认磁盘是否已经挂载。

```
df -h
```

根据以下回显信息，可以确认磁盘已经挂载。

```
[root@emr-worker-5 ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs         32G     0   32G   0% /dev
tmpfs            32G  104K   32G   1% /dev/shm
tmpfs            32G  788K   32G   1% /run
tmpfs            32G     0   32G   0% /sys/fs/cgroup
/dev/vda1       118G   32G   81G  29% /
/dev/vde        5.4T  1.6T  3.8T  30% /mnt/disk4
/dev/vdg        5.4T  684G  4.7T  13% /mnt/disk6
/dev/vdd        5.4T  1.3T  4.2T  23% /mnt/disk3
/dev/vdb        5.4T  433G  5.0T   8% /mnt/disk1
/dev/vdi        5.4T  920G  4.5T  17% /mnt/disk8
/dev/vdc        5.4T  1.1T  4.3T  21% /mnt/disk2
/dev/vdf        5.4T  1.2T  4.2T  22% /mnt/disk5
tmpfs           6.3G     0  6.3G   0% /run/user/0
tmpfs           6.3G     0  6.3G   0% /run/user/1013
tmpfs           6.3G     0  6.3G   0% /run/user/1001
/dev/vdh        5.4T   89M  5.4T   1% /mnt/disk7
[root@emr-worker-5 ~]#
```

7. 按照您选择的Kafka磁盘修复策略，恢复Kafka Broker节点日志目录，详情请参见EMR Kafka磁盘故障运维。
8. （可选）按照您选择的Kafka磁盘修复策略，迁移Kafka分区副本至修复后的磁盘，使负载更加均衡，详情请参见EMR Kafka磁盘故障运维。

5.1.3.3.4. 限制Kafka服务端运维流量

本文为您介绍如何在Kafka集群运维中对Kafka运维流量进行限制，以避免由于运维流量影响到正常的业务流量。本文以EMR Kafka 2.4.1版本为例。

背景信息

由于运维操作而出现的IO流量称为运维流量。在以下运维场景中需要对运维流量进行限制：

- Partition Reassign场景。
- 节点内副本移动到不同目录的场景。
- 集群Broker恢复时，副本数据同步的场景。

注意事项

- 由于Kafka流量构成不同、业务场景不同和运维场景不同，您需要综合判断是否需要对运维流量进行限制。
- 限流的阈值需要根据具体的业务场景来确定。通常运维流量阈值过小将导致运维操作无法完成，运维流量阈值过大将造成IO争抢或带宽满载等

问题，从而影响到正常业务流量，您应该合理的评估限流阈值。

- 限流阈值设定需要考虑Topic业务流量的大小、业务可以承受的延迟、业务场景是否允许Kafka服务中断、Kafka集群自身的磁盘IO与网络IO的带宽能力等因素。
- 通常情况下，建议您在业务低峰期间进行此类运维操作。

Kafka运维流量限制

Kafka限流相关参数

参数	描述
leader.replication.throttled.replicas	Topic级别，表示需要限流的分区leader副本列表。 格式为[`PartitionId]:[BrokerId],[PartitionId]:[BrokerId]:...` 或者用星号（*）表示该Topic的所有leader副本限流。
follower.replication.throttled.replicas	Topic级别，表示需要限流的分区follower副本列表。 格式为 `[PartitionId]:[BrokerId],[PartitionId]:[BrokerId]:...` 或者用星号（*）表示该Topic的所有follwer副本限流。
leader.replication.throttled.rate	Broker级别，leader节点复制读流量。
follower.replication.throttled.rate	Broker级别，follower节点复制写流量。

限流参数查看方式

您可以通过 `kafka-configs.sh` 命令来查看限流参数的值。

- 查看指定节点的Broker参数。

```
kafka-configs.sh --zookeeper master-1-1:2181/emr-kafka -entity-type brokers --entity-name <your broker id> --describe
```

- 查看指定Topic的参数。

```
kafka-configs.sh --zookeeper master-1-1:2181 -entity-type topics --entity-name <your topic name> --describe
```

Partition Reassign场景限流

> 注意
> - 限流速度不能过小，如果限流速度过小，将不能触发实际的reassign复制过程。
> - 限流参数不会对正常的副本fetch流量进行限速。
> - 任务完成后，您需要通过verify参数移除Topic和Broker上的限速参数配置。
> - 如果刚开始已经设置了throttle参数，则可以通过 `execute` 命令再次修改throttle参数。
> - 如果刚开始没有设置throttle参数，则需要使用 `kafka-configs.sh` 命令修改Topic上的leader.replication.throttled.replicas和follower.replication.throttled.replicas参数、修改Broker上的leader.replication.throttled.rate和follower.replication.throttled.rate参数。

通常使用*kafka-reassign-partitions.sh*工具来进行Partition Reassign操作，通过使用throttle参数来设置限流的大小。示例如下所示：

1. 创建测试Topic。
   i. 以SSH方式登录到Kafka集群的Master节点，详情请参见登录集群。
   ii. 执行以下命令，创建测试Topic。

   ```
   kafka-topics.sh --zookeeper master-1-1:2181/emr-kafka --topic test-throttled --partitions 1 --replication-factor 3 --create
   ```

   您可以通过以下命令查看Topic详情。

   ```
   kafka-topics.sh --zookeeper  master-1-1:2181/emr-kafka --topic test-throttled --describe
   ```

2. 执行以下命令，模拟数据写入。

   ```
   kafka-producer-perf-test.sh --topic test-throttled --record-size 1000 --num-records 600000000 --print-metrics --throughput 10240 --producer-props acks=-1  linger.ms=0 bootstrap.servers=core-1-1:9092
   ```

3. 设置throttle参数并执行reassign操作。

i. 创建reassignment-json-file文件*reassign.json*，写入如下内容。

```
{"version":1,"partitions":[{"topic":"test-throttled","partition":0,"replicas":[2,0,3],"log_dirs":["any","any","any"]}]}
```

ii. 执行reassign操作。

由于模拟的写入速度为10 Mbit/s，所以将reassign限流速度设置为30 Mbit/s。

```
kafka-reassign-partitions.sh --zookeeper  master-1-1:2181/emr-kafka --reassignment-json-file reassign.json --throttle 30000000 --execute
```

4. 查看限流参数。
   - 查看指定节点的Broker参数。

     ```
     kafka-configs.sh --zookeeper master-1-1:2181/emr-kafka -entity-type brokers --entity-name 2 --describe
     ```

   - 查看指定Topic的参数。

     ```
     kafka-configs.sh --zookeeper master-1-1:2181 -entity-type topics --entity-name test-throttled --describe
     ```

5. 查看reassign任务执行情况。

```
kafka-reassign-partitions.sh --zookeeper master-1-1:2181/emr-kafka --reassignment-json-file reassign.json --verify
```

> **说明** 任务完成后，您需要重复执行上述命令以移除限流参数。

节点内副本移动到不同目录的场景限流

通过*kafka-reassign-partitions.sh*工具可以进行Broker节点内的副本迁移，参数replica-alter-log-dirs-throttle可以对节点内的迁移IO进行限制。示例如下所示：

1. 创建测试Topic。

   i. 以SSH方式登录到Kafka集群的Master节点，详情请参见登录集群。

   ii. 执行以下命令，创建测试Topic。

   ```
   kafka-topics.sh --zookeeper master-1-1:2181/emr-kafka --topic test-throttled --partitions 1 --replication-factor 3 --create
   ```

   您可以通过以下命令查看Topic详情。

   ```
   kafka-topics.sh --zookeeper  master-1-1:2181/emr-kafka --topic test-throttled --describe
   ```

2. 执行以下命令，模拟数据写入。

```
kafka-producer-perf-test.sh --topic test-throttled --record-size 1000 --num-records 600000000 --print-metrics --throughput 10240 --producer-props acks=-1  linger.ms=0 bootstrap.servers=core-1-1:9092
```

3. 设置参数replica-alter-log-dirs-throttle并执行reassign操作。

   i. 创建文件*reassign.json*，将目标目录写入reassignment文件中，内容如下。

   ```
   {"version":1,"partitions":[{"topic":"test-throttled","partition":0,"replicas":[2,0,3],"log_dirs":["any","/mnt/disk1/kafka/log","any"]}]}
   ```

   ii. 执行replicas movement操作。

   ```
   kafka-reassign-partitions.sh --zookeeper master-1-1:2181/emr-kafka --reassignment-json-file reassign.json --replica-alter-log-dirs-throttle 30000000 --execute
   ```

4. 查看限流参数。

   Broker节点内目录间移动副本会在Broker上配置限流参数，参数名为Brokerreplica.alter.log.dirs.io.max.bytes.per.second。

   执行以下命令，查看指定节点的Broker参数。

```
kafka-configs.sh --zookeeper master-1-1:2181 -entity-type brokers --describe --entity-name 0
```

5. 查看reassign任务执行情况。

```
kafka-reassign-partitions.sh --zookeeper master-1-1:2181/emr-kafka --reassignment-json-file reassign.json --verify
```

> **说明** 任务完成后，您需要重复执行上述命令以移除限流参数。

集群Broker恢复时，副本数据同步场景限流

> 注意
> - 限流速度不能过小，如果限流速度过小，将不能触发实际的reassign复制过程。
> - 限流参数不会对正常的副本fetch流量进行限速。
> - 数据恢复完成后，需要使用 `kafka-configs.sh` 命令删除相应的参数。

当Broker重启时，需要从leader副本进行副本数据的同步。在Broker节点迁移、坏盘修复重新上线等场景时，由于之前的副本数据完全丢失、副本数据恢复会产生大量的同步流量，有必要对恢复过程进行限流避免恢复流量过大影响正常流量。示例如下所示：

1. 创建测试Topic。
   i. 以SSH方式登录到Kafka集群的Master节点，详情请参见登录集群。
   ii. 执行以下命令，创建测试Topic。

   ```
   kafka-topics.sh --zookeeper master-1-1:2181/emr-kafka --topic test-throttled --partitions 1 --replication-factor 3
   --create
   ```

   您可以通过以下命令查看Topic详情。

   ```
   kafka-topics.sh --zookeeper  master-1-1:2181/emr-kafka --topic test-throttled --describe
   ```

2. 执行以下命令，写入测试数据。

   ```
   kafka-producer-perf-test.sh --topic test-throttled --record-size 1000 --num-records 600000000 --print-metrics --through
   put 10240 --producer-props acks=-1  linger.ms=0 bootstrap.servers=core-1-1:9092
   ```

3. 通过 `kafka-configs.sh` 命令设置限流参数。

   ```
   //设置Topic上的限流参数。
   kafka-configs.sh --zookeeper master-1-1:2181/emr-kafka --entity-type topics --entity-name test-throttled --alter --add-
   config "leader.replication.throttled.replicas=*,follower.replication.throttled.replicas=*"
   //设置Broker上的限流参数。
   kafka-configs.sh --bootstrap-server master-1-1:2181/emr-kafka --entity-type brokers --alter --add-config "leader.replic
   ation.throttled.rate=1024,follower.replication.throttled.rate=1024" --entity-name 0
   kafka-configs.sh --bootstrap-server master-1-1:2181/emr-kafka --entity-type brokers --alter --add-config "leader.replic
   ation.throttled.rate=1024,follower.replication.throttled.rate=1024" --entity-name 1
   kafka-configs.sh --bootstrap-server master-1-1:2181/emr-kafka --entity-type brokers --alter --add-config "leader.replic
   ation.throttled.rate=1024,follower.replication.throttled.rate=1024" --entity-name 2
   ......
   ```

4. 在EMR控制台停止Broker 1节点。
5. 删除Broker 1上的副本数据，模拟数据丢失的场景。

   ```
   rm -rf /mnt/disk2/kafka/log/test-throttled-0/
   ```

6. 在EMR控制台启动Broker 1节点，观察限流参数是否起作用。
7. 待Broker 1相应的副本恢复到ISR列表后，使用 `kafka-configs.sh` 命令删除限流参数的配置。

   ```
   //删除Topic上的限流参数。
   kafka-configs.sh --zookeeper master-1-1:2181/emr-kafka -entity-type topics --alter --delete-config 'leader.replication.
   throttled.replicas,follower.replication.throttled.replicas' --entity-name test-throttled
   //删除Broker上的限流参数
   kafka-configs.sh --zookeeper master-1-1:2181/emr-kafka -entity-type brokers --alter --delete-config 'leader.replication
   .throttled.replicas,follower.replication.throttled.replicas,leader.replication.throttled.rate,follower.replication.thro
   ttled.rate' --entity-name 0
   ......
   ```

5.1.3.4. 最佳实践

5.1.3.4.1. 使用MirrorMaker 2.0同步数据

本文通过示例为您介绍如何通过EMR的集群脚本功能，快速部署使用MirrorMaker 2.0（MM2）服务同步数据。

背景信息

本文的业务场景以EMR DataFlow集群作为目的集群，并且在目的集群中以Dedicated MirrorMaker集群的方式部署MM2，即EMR DataFlow集群既作为目的集群又作为Dedicated MirrorMaker集群。在实际业务场景中，您可以将MirrorMaker集群部署到单独的服务器上。

前提条件

- 已创建两个Kafka集群，一个为源集群，一个为目的集群（EMR DataFlow集群），并选择了Kafka服务，创建DataFlow集群详情请参见创建集

群。

> **说明** 本文示例的源和目的集群都以EMR-3.41.0版本的DataFlow集群为例。

- 已在OSS上创建存储空间，详情请参见创建存储空间。

使用限制

EMR DataFlow集群的Kafka软件的版本为2.12_2.4.1及以上。

操作步骤

1. 准备MM2配置文件*mm2.properties*并上传到您的OSS存储。

   以下配置内容仅作为参考，您需要替换文本中的源集群和目标集群的src.bootstrap.servers和dest.bootstrap.servers，并根据实际业务需求进行相应的配置。MM2配置的详细信息请参见Configuring Geo-Replication。

   ```
   # see org.apache.kafka.clients.consumer.ConsumerConfig for more details
   # Sample MirrorMaker 2.0 top-level configuration file
   # Run with ./bin/connect-mirror-maker.sh connect-mirror-maker.properties
   # specify any number of cluster aliases
   clusters = src, dest
   # connection information for each cluster
   src.bootstrap.servers = <your source kafka cluster servers>
   dest.bootstrap.servers = <your destination kafka cluster servers>
   # enable and configure individual replication flows
   src->dest.enabled = true
   src->dest.topics = foo-.*
   groups=.*
   topics.blacklist="__.*"
   # customize as needed
   replication.factor=3
   ```

2. 准备部署脚本*kafka_mm2_deploy.sh*并上传到OSS存储。

   ```
   #!/bin/bash
   SIGNAL=${SIGNAL:-TERM}
   PIDS=$(ps ax | grep -i 'org.apache.kafka.connect.mirror.MirrorMaker' | grep java | grep -v grep | awk '{print $1}')
   if [ -n "$PIDS" ]; then
     echo "stop the exist mirror maker server."
     kill -s $SIGNAL $PIDS
   fi
   KAFKA_CONF=/etc/taihao-apps/kafka-conf/kafka-conf
   TAIHAO_EXECUTOR=/usr/local/taihao-executor-all/executor/1.0.1
   cd $KAFKA_CONF
   if [ -e "./mm2.properties" ]; then
     mv mm2.properties mm2.properties.bak
   fi
   ${TAIHAO_EXECUTOR}/ossutil64 cp oss://<yourBuket>/mm2.properties ./ -e <yourEndpoint> -i <yourAccessKeyId> -k <yourAcce
   ssKeySecret>
   su - kafka <<EOF
   exec connect-mirror-maker.sh -daemon $KAFKA_CONF/mm2.properties
   exit;
   EOF
   ```

   涉及替换参数如下。

参数	描述
KAFKA_CONF	检查变量路径是否正确，如果不正确，则需要修改为实际的地址。
TAIHAO_EXECUTOR	
oss://<yourBucket>/mm2.properties	替换为*mm2.properties*的实际存储路径。
<yourEndpoint>	OSS服务的地址。
<yourAccessKeyId>	阿里云账号的AccessKey ID。
<yourAccessKeySecret>	阿里云账号的AccessKey Secret。

3. 在EMR控制台执行脚本， 具体操作请参见手动执行脚本。

> **说明** 在创建执行脚本的过程中，您应正确选择脚本的执行节点，通常选择所有的Broker节点。

执行完成后，即实现了Kafka集群间的数据迁移。

5.1.4. Kafka Manager

E-MapReduce支持通过Kafka Manager服务对DataFlow Kafka集群进行管理。

前提条件

已创建DataFlow类型的集群，并选择Kafka服务，创建详情请参见创建集群。

> **说明** 创建DataFlow Kafka集群（选择了Kafka服务）时，默认安装Kafka Manager软件服务，并开启了Kafka Manager的认证功能。

注意事项

使用Kafka Manager进行partition reassign功能时，当前版本的Kafka Manager是没有提供限流功能的。如果需要限流，您可以通过*kafka-configs.sh*手工配置限流参数，具体限流方式请参见限制Kafka服务端运维流量。

操作步骤

1. 使用SSH隧道方式访问Web页面，详情请参见通过SSH隧道方式访问开源组件Web UI。

   > **说明**
   > - 建议您首次使用Kafka Manager时修改默认密码。
   > - 为了防止8085端口暴露，建议使用SSH隧道方式来访问Web界面。如果使用*http://localhost:8085*方式访问Web界面，请做好IP白名单保护，避免数据泄漏。

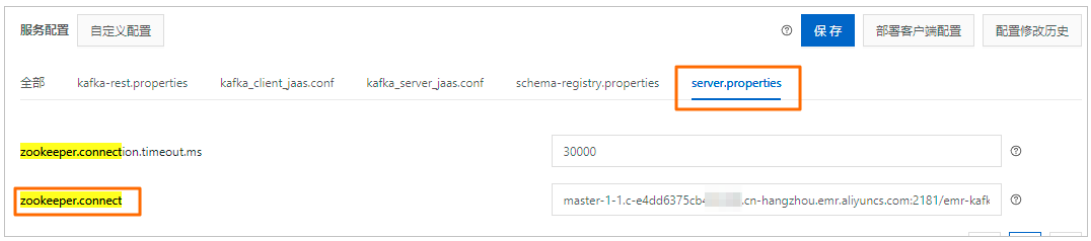
2. 在登录页面，输入用户名和密码。

   用户名、密码和Zookeeper地址，您可以通过以下步骤获取：

   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 获取以下配置信息。
   - 获取用户名和密码：
     a. 在**集群服务**页面，单击Kafka Manager服务区域的**配置**。
     b. 在**服务配置**区域，查看以下参数的值：
     - **kafka.manager.authentication.username**：登录Kafka Manager页面的用户名。
     - **kafka.manager.authentication.password**：登录Kafka Manager页面的密码。

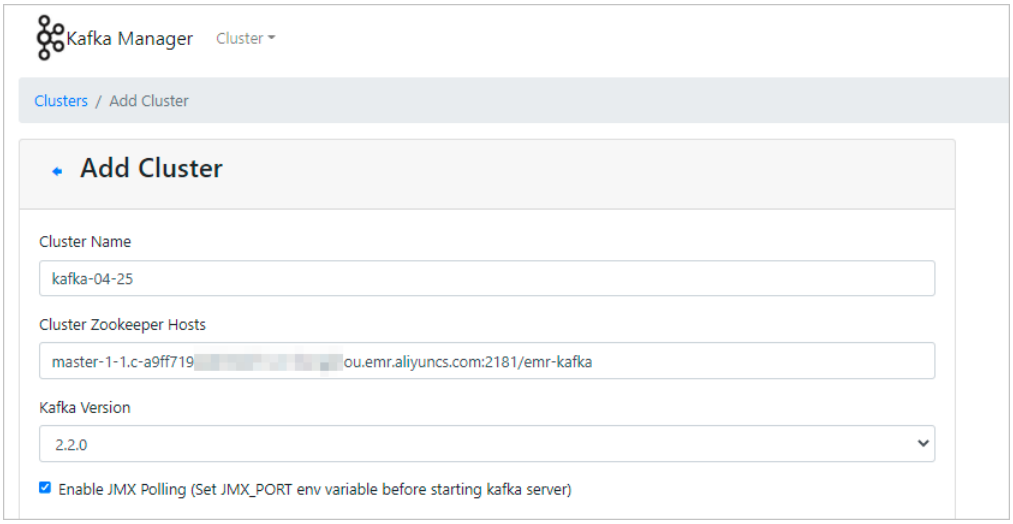
   - 获取集群的Zookeeper地址：
     a. 在**集群服务**页面，单击Kafka服务区域的**配置**。
     b. 在**服务配置**区域，查看**server.properties**页签下的**zookeeper.connect**的参数值，即集群的Zookeeper地址。

3. 在Kafka Manager页面，选择**Cluster > Add Cluster**。

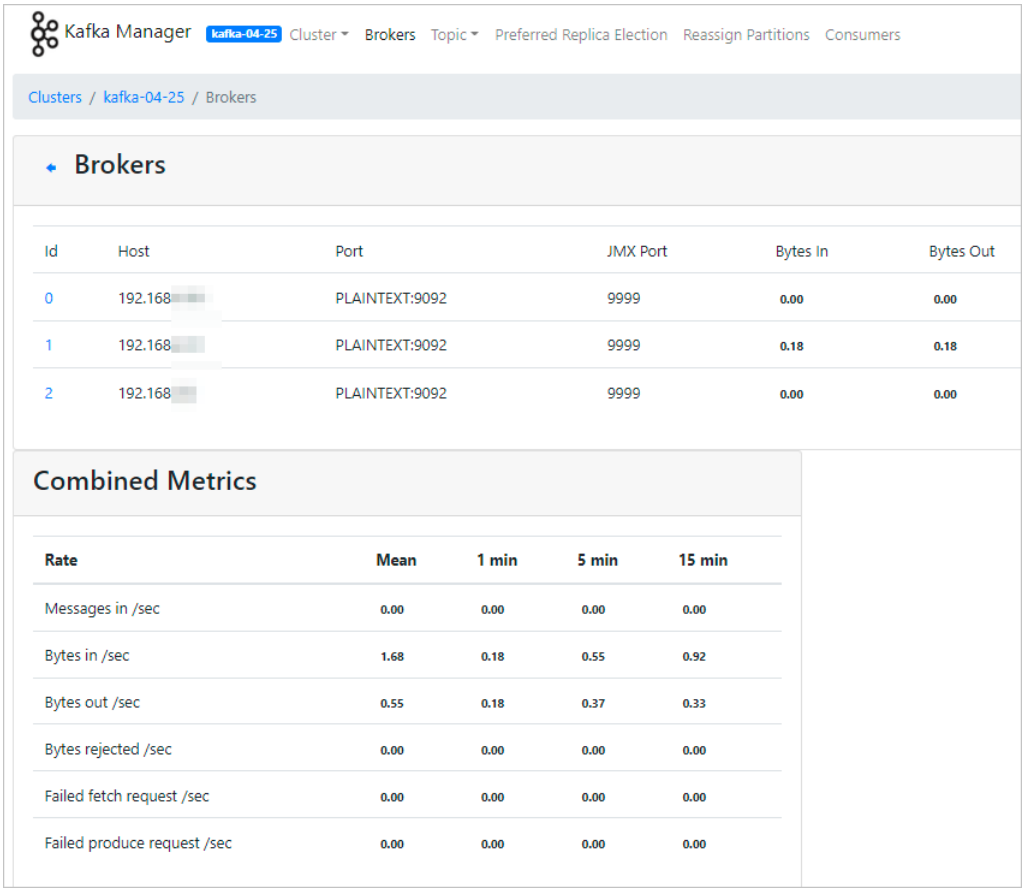
   添加已创建的DataFlow Kafka集群。

4. 在**Add Cluster**页面，配置以下参数，单击**Save**。



参数	描述
Cluster Name	集群名称。
Cluster Zookeeper Hosts	集群的Zookeeper地址。 填写在步骤2中获取到的zookeeper.connect的值。
Kafka Version	选择对应的Kafka版本。 **说明** 由于Kafka Manager可能不支持高版本的Kafka集群，所以此参数值选择最接近的版本即可。
Enable JMX Polling (Set JMX_PORT env variable before starting kafka server)	是否开启JMX功能。 本示例开启了JMX功能。
brokerViewThreadPoolSize	获取数据时的线程池大小。 **注意** 修改参数值大于2。
offsetCacheThreadPoolSize	缓存offset的线程池大小。 **注意** 修改参数值大于2。
kafkaAdminClientThreadPoolSize	AdminClient的线程池大小。 **注意** 修改参数值大于2。

创建好之后即可使用常见的Kafka功能。



5.2. ClickHouse

5.2.1. ClickHouse概述

开源大数据平台E-MapReduce（简称EMR）的ClickHouse提供了开源OLAP分析引擎ClickHouse的云上托管服务。EMR ClickHouse完全兼容开源版本的产品特性，同时提供集群快速部署、集群管理、扩容、缩容和监控告警等云上产品功能，并且在开源的基础上优化了ClickHouse的读写性能，提升了ClickHouse与EMR其他组件快速集成的能力。

特性

特性	描述
列式存储	相较于行式存储，列式存储在查询性能上更优。同时列式存储的数据压缩比更高，更加节省存储空间。
MPP架构	每个节点只访问本地内存和存储，节点信息交互和节点本身是并行处理的。查询性能好，易于扩展。 向量化引擎：为了高效的使用CPU，数据不仅仅按列存储，同时还按向量（列的一部分）进行处理，这样可以更加高效地使用CPU。
支持SQL	ClickHouse支持一种基于SQL的声明式查询语言，它在许多情况下与ANSI SQL标准相同。支持GROUP BY、ORDER BY、FROM、JOIN和IN查询以及非相关子查询。
实时的数据更新	ClickHouse支持在表中定义主键。为了使查询能够快速在主键中进行范围查找，数据总是以增量的方式有序的存储在MergeTree中。 近实时数据更新， Clickhouse支持近实时的数据插入、指标聚合以及索引创建。
支持索引	按照主键对数据进行排序，ClickHouse可以在几十毫秒以内完成对数据特定值或范围的查找。

典型应用场景

场景	描述
用户行为分析	行为分析系统的表可以制作成一张大的宽表，每个表包含大量的列，可以超过一千列。JOIN的形式相对少一点，可以实现路径分析、漏斗分析和路径转化等功能。
流量和监控	可以将系统和应用监控指标通过流式计算引擎Flink或Spark streaming将监控数据清洗处理以后，实时写入ClickHouse，然后结合Grafana进行可视化展示。
用户画像	可以将各种用户特征进行数据加工，制作成包含全部用户的一张或多张用户特征表，提供灵活的用户画像分析、支撑广告和圈人等业务需求。
实时BI报表	根据业务需求，可以实时制作一些及时产出的查询灵活的BI报表，实现秒级查询，绝大多数查询能够实时反馈。BI报表包括订单分析、营销效果分析和大促活动分析。

> **说明** 不合适的场景：
> - 没有完整的事务支持。
> - 缺少高频率、低延迟的修改或删除已存在数据的能力。
> - 仅能用于批量删除或修改数据。

5.2.2. 快速入门

5.2.2.1. 创建集群

本文为您介绍创建ClickHouse集群的详细操作步骤和相关配置。

背景信息

机型、内存和磁盘的设置，请参见Usage Recommendations。

前提条件

已在目标地域创建一个专有网络和交换机，详情请参见创建和管理专有网络和创建和管理交换机。

操作步骤

1. 进入创建集群页面。
   i. 登录EMR on ECS控制台。
   ii. （可选）在顶部菜单栏处，根据实际情况选择地域和资源组。
      - 地域：创建的集群将会在对应的地域内，一旦创建不能修改。
      - 资源组：默认显示账号全部资源。
   iii. 单击上方的**创建集群**，进行创建。
2. 配置集群信息。

   创建集群时，您需要对集群进行软件配置、硬件配置、基础配置和确认订单。

   > **注意** 集群创建完成后，除了集群名称以外，其他配置均无法修改，所以在创建时请仔细确认各项配置。

   i. 软件配置。

配置项	说明
地域	创建的集群将会在对应的地域内，一旦创建不能修改。
业务场景	选择**数据分析场景**。
产品版本	默认最新的软件版本。
服务高可用	默认开启，如果不需要高可用集群，可以关闭**服务高可用**开关。
可选服务	需要选择**CLICKHOUSE**。 默认的服务组件，后期可以在管理页面中启停服务。
高级设置	**软件自定义配置**：可指定JSON文件对集群中的基础软件进行配置，详细使用方法请参见配置自定义软件。默认不开启。

ii. 硬件配置。

配置项	说明
付费类型	默认包年包月。当前支持的付费类型如下： ■ **按量付费**：一种后付费模式，即先使用再付费。按量付费是根据实际使用的小时数来支付费用，每小时计费一次，适合短期的测试任务或是灵活的动态任务。 ■ **包年包月**：一种预付费模式，即先付费再使用。 说明 建议测试场景下使用**按量付费**，测试正常后再新建一个**包年包月**的生产集群正式使用。
可用区	可用区为在同一地域下的不同物理区域，可用区之间内网互通。通常使用默认的可用区即可。
网络类型	默认专有网络。
VPC	选择在该地域的VPC。如果没有可用的VPC，单击**创建VPC/子网（交换机）**前往新建。
交换机	选择在对应VPC下可用区的交换机，如果在这个可用区没有可用的交换机，则需要新创建一个。
安全组	默认选择已有的安全组。安全组详情请参见安全组概述。 您也可以单击**新建安全组**，然后直接输入安全组名称来新建一个安全组。 注意　禁止使用ECS上创建的企业安全组。
挂载公网	集群是否挂载弹性公网IP地址，默认不开启。 说明　不开启**挂载公网**，将无法使用EMR控制台访问链接与端口功能查看开源组件的Web UI。
实例	您可以根据需要选择实例规格，详情请参见实例规格族。 ■ **系统盘**：根据需要选择ESSD云盘、SSD云盘或者高效云盘。 ■ **系统盘大小**：根据需要调整磁盘容量，默认为80 GB。取值范围为80 ~ 5000 GB。 ■ **数据盘**：根据需要选择ESSD云盘、SSD云盘或者高效云盘。 ■ **数据盘大小**：根据需要调整磁盘容量，默认为80 GB。取值范围为40 ~ 32768 GB。 ■ **实例数量**： 　■ 关闭**服务高可用**开关：默认1台Master，1台Core。 　■ 开启**服务高可用**开关：默认3台Master，3台Core。

iii. 基础配置。

配置项	说明
集群名称	集群的名字，长度限制为1~64个字符，仅可使用中文、字母、数字、短划线（-）和下划线（_）。
身份凭证	**密钥对**（默认）：使用SSH密钥对登录Linux实例。 关于密钥对的使用详情，请参见SSH密钥对。
	密码：设置Master节点的登录密码，使用密码对登录Linux实例。 密码规则：8~30个字符，且必须同时包含大写字母、小写字母、数字和特殊字符。 特殊字符包括：感叹号（!）、at（@）、井号（#）、美元符号（$）、百分号（%）、乘方（^）、and（&）和星号（*）。
应用配置	配置ClickHouse的副本（Replica）与分片（Shard）。
高级设置	■ **添加用户**：添加访问开源大数据软件Web UI的账号。 ■ **ECS应用角色**：当您的程序在EMR计算节点上运行时，可不填写阿里云AccessKey来访问相关的云服务（例如OSS），EMR会自动申请一个临时AccessKey来授权本次访问。**ECS应用角色**用于控制这个AccessKey的权限。 ■ **引导操作**：可选配置，您可以在集群启动Hadoop前执行您自定义的脚本，详情请参见管理引导操作。 ■ **资源组**：可选配置。详情请参见使用资源组。

3. 当所有的信息确认正确后，选中服务协议，单击**确认订单**。

> 注意
> - 按量付费集群：立刻开始创建。
>
>   集群创建完成后，集群的状态变为**运行中**。
> - 包年包月集群：先生成订单，支付完成订单以后集群才会开始创建。

5.2.2.2. 快速使用ClickHouse

本文通过示例为您介绍，如何快速将数据随机写入ClickHouse集群各个节点的本地表。

前提条件

已创建ClickHouse集群，详情请参见创建集群。

操作步骤

1. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。
2. 执行以下命令，下载官方样例数据集。

```
curl https://datasets.clickhouse.com/hits/tsv/hits_v1.tsv.xz  | unxz --threads=`nproc` > hits_v1.tsv
```

3. 执行如下命令，启动ClickHouse客户端。

```
clickhouse-client -h core-1-1 -m
```

> **说明** 本示例登录core-1-1节点，如果您有多个Core节点，可以登录任意一个节点。

4. 执行如下命令，创建数据库。

   可以使用on CLUSTER参数在集群的所有节点创建数据库，默认集群标识为cluster_emr。

```
CREATE DATABASE IF NOT EXISTS demo on CLUSTER cluster_emr;
```

返回信息如下所示。

```
core-1-1.c-2ecd13e05 cn-hangzhou.emr.aliyuncs.com :) CREATE DATABASE IF NOT EXISTS demo on CLUSTER cluster_emr;

CREATE DATABASE IF NOT EXISTS demo ON CLUSTER cluster_emr

Query id: ecf856dc-3490-498a-8368-386f38dc

┌─host──────────────────────────────────────────┬─port─┬─status─┬─error─┬─num_hosts_remaining─┬─num_hosts_active─┐
│ core-1-1.c-2ecd13e05 .cn-hangzhou.emr.aliyuncs.com │ 9000 │      0 │       │                   0 │                0 │
└───────────────────────────────────────────────┴──────┴────────┴───────┴─────────────────────┴──────────────────┘

1 rows in set. Elapsed: 0.123 sec.
```

5. 在集群上的所有节点创建一张复制表（Replicated表）。

   复制表（Replicated表）会根据副本的个数，实现数据的多副本，并实现数据的最终一致性。

```
CREATE TABLE demo.hits_local ON CLUSTER cluster_emr
(
    `WatchID` UInt64,
    `JavaEnable` UInt8,
    `Title` String,
    `GoodEvent` Int16,
    `EventTime` DateTime,
    `EventDate` Date,
    `CounterID` UInt32,
    `ClientIP` UInt32,
    `ClientIP6` FixedString(16),
    `RegionID` UInt32,
    `UserID` UInt64,
    `CounterClass` Int8,
    `OS` UInt8,
    `UserAgent` UInt8,
    `URL` String,
    `Referer` String,
    `URLDomain` String,
    `RefererDomain` String,
    `Refresh` UInt8,
    `IsRobot` UInt8,
    `RefererCategories` Array(UInt16),
    `URLCategories` Array(UInt16),
    `URLRegions` Array(UInt32),
    `RefererRegions` Array(UInt32),
    `ResolutionWidth` UInt16,
    `ResolutionHeight` UInt16,
    `ResolutionDepth` UInt8,
    `FlashMajor` UInt8,
    `FlashMinor` UInt8,
    `FlashMinor2` String,
    `NetMajor` UInt8,
    `NetMinor` UInt8,
    `UserAgentMajor` UInt16,
    `UserAgentMinor` FixedString(2),
    `CookieEnable` UInt8,
    `JavascriptEnable` UInt8,
    `IsMobile` UInt8,
    `MobilePhone` UInt8,
    `MobilePhoneModel` String,
    `Params` String,
    `IPNetworkID` UInt32,
    `TraficSourceID` Int8,
    `SearchEngineID` UInt16,
    `SearchPhrase` String,
    `AdvEngineID` UInt8,
    `IsArtifical` UInt8,
    `WindowClientWidth` UInt16,
    `WindowClientHeight` UInt16,
    `ClientTimeZone` Int16,
    `ClientEventTime` DateTime,
    `SilverlightVersion1` UInt8,
    `SilverlightVersion2` UInt8,
    `SilverlightVersion3` UInt32,
    `SilverlightVersion4` UInt16,
    `PageCharset` String,
    `CodeVersion` UInt32,
```

```
    `IsLink` UInt8,
    `IsDownload` UInt8,
    `IsNotBounce` UInt8,
    `FUniqID` UInt64,
    `HID` UInt32,
    `IsOldCounter` UInt8,
    `IsEvent` UInt8,
    `IsParameter` UInt8,
    `DontCountHits` UInt8,
    `WithHash` UInt8,
    `HitColor` FixedString(1),
    `UTCEventTime` DateTime,
    `Age` UInt8,
    `Sex` UInt8,
    `Income` UInt8,
    `Interests` UInt16,
    `Robotness` UInt8,
    `GeneralInterests` Array(UInt16),
    `RemoteIP` UInt32,
    `RemoteIP6` FixedString(16),
    `WindowName` Int32,
    `OpenerName` Int32,
    `HistoryLength` Int16,
    `BrowserLanguage` FixedString(2),
    `BrowserCountry` FixedString(2),
    `SocialNetwork` String,
    `SocialAction` String,
    `HTTPError` UInt16,
    `SendTiming` Int32,
    `DNSTiming` Int32,
    `ConnectTiming` Int32,
    `ResponseStartTiming` Int32,
    `ResponseEndTiming` Int32,
    `FetchTiming` Int32,
    `RedirectTiming` Int32,
    `DOMInteractiveTiming` Int32,
    `DOMContentLoadedTiming` Int32,
    `DOMCompleteTiming` Int32,
    `LoadEventStartTiming` Int32,
    `LoadEventEndTiming` Int32,
    `NSToDOMContentLoadedTiming` Int32,
    `FirstPaintTiming` Int32,
    `RedirectCount` Int8,
    `SocialSourceNetworkID` UInt8,
    `SocialSourcePage` String,
    `ParamPrice` Int64,
    `ParamOrderID` String,
    `ParamCurrency` FixedString(3),
    `ParamCurrencyID` UInt16,
    `GoalsReached` Array(UInt32),
    `OpenstatServiceName` String,
    `OpenstatCampaignID` String,
    `OpenstatAdID` String,
    `OpenstatSourceID` String,
    `UTMSource` String,
    `UTMMedium` String,
    `UTMCampaign` String,
    `UTMContent` String,
    `UTMTerm` String,
    `FromTag` String,
    `HasGCLID` UInt8,
    `RefererHash` UInt64,
    `URLHash` UInt64,
    `CLID` UInt32,
    `YCLID` UInt64,
    `ShareService` String,
    `ShareURL` String,
    `ShareTitle` String,
    `ParsedParams` Nested(Key1 String,Key2 String,Key3 String,Key4 String,Key5 String,ValueDouble Float64),
    `IslandID` FixedString(16),
    `RequestNum` UInt32,
    `RequestTry` UInt8
)
```

```
)
ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}/{database}/hits_local', '{replica}')
PARTITION BY toYYYYMM(EventDate)
ORDER BY (CounterID, EventDate, intHash32(UserID))
SAMPLE BY intHash32(UserID);
```

> **说明** {shard}和{replica}是阿里云EMR为ClickHouse集群自动生成的宏定义，可以直接使用。

6. 执行以下命令，创建分布式（Distributed）表。

   分布式表不存储数据，仅仅是底层表的一个View，但可以在多个服务器上进行分布式查询。本例中使用随机函数rand()，表示数据会随机写入各个节点的本地表。

```
CREATE TABLE demo.hits_all on CLUSTER cluster_emr AS demo.hits_local
ENGINE = Distributed(cluster_emr, demo, hits_local, rand());
```

7. 退出ClickHouse客户端，在样例数据的目录下执行以下命令，导入数据。

```
clickhouse-client -h core-1-1 --query "INSERT INTO demo.hits_all FORMAT TSV" --max_insert_block_size=100000 < hits_v1.t
sv;
```

8. 重新启动ClickHouse客户端，查看数据。

   因为数据是随机写入的，各节点数据量可能不同。

   - 查看core-1-1节点*demo.hits_all*的数据量。

```
select count(*) from demo.hits_all;
```

     返回信息如下。

```
┌─count()─┐
│ 8873898 │
└─────────┘
```

   - 查看core-1-1节点*demo.hits_local*的数据量。

```
select count(*) from demo.hits_local;
```

     返回信息如下。

```
┌─count()─┐
│ 8873898 │
└─────────┘
```

   - 查看core-1-2节点*demo.hits_local*的数据量。

     > **说明** 其余节点，您也可以按照以下步骤来查看*demo.hits_local*的数据量。节点名称您可以在EMR控制台的**节点管理**页面查看。

     a. 执行以下命令，登录ClickHouse客户端。

```
clickhouse-client -h core-1-2 -m
```

     b. 在ClickHouse客户端，执行以下命令，查看*demo.hits_local*的数据量。

```
select count(*) from demo.hits_local;
```

     返回信息如下。

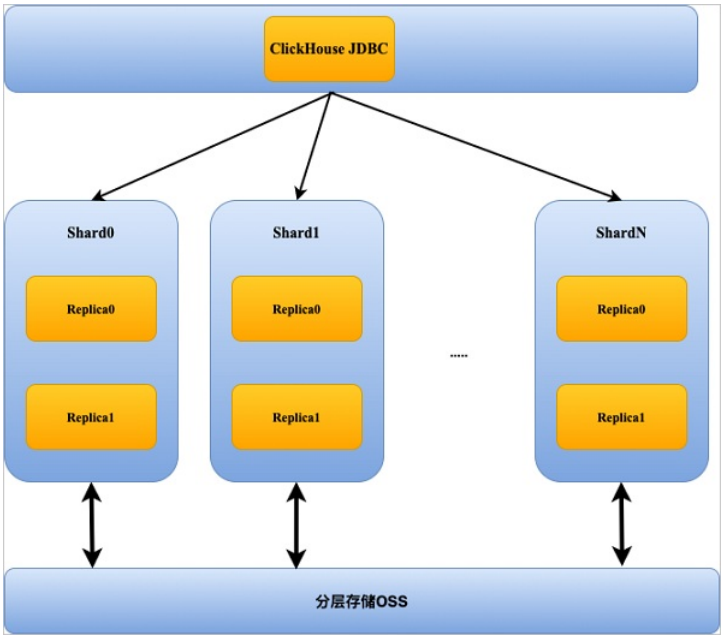
```
┌─count()─┐
│ 8873898 │
└─────────┘
```

5.2.2.3. 访问模式

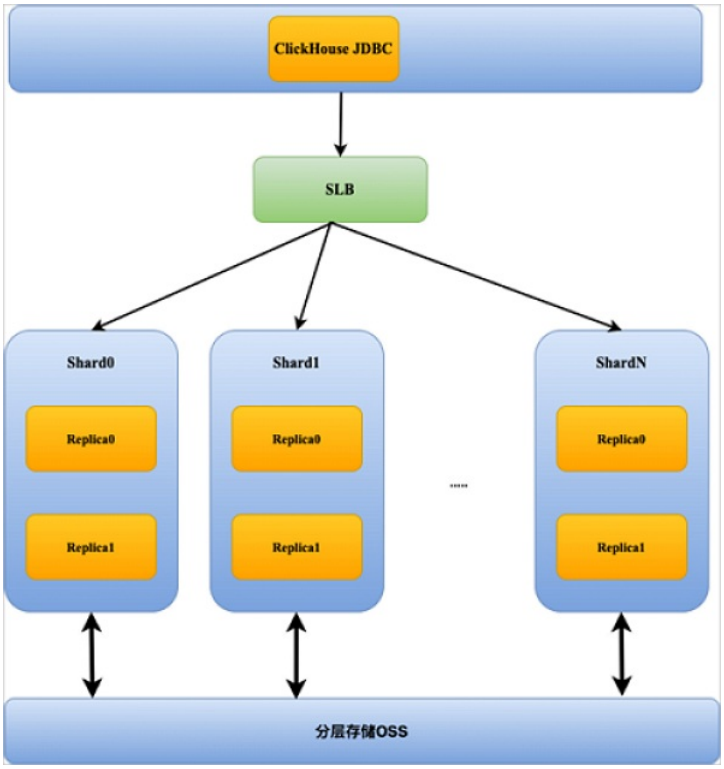
访问E-MapReduce（简称EMR）上的ClickHouse集群支持通过原生JDBC访问和通过负载均衡SLB访问两种方式。本文为您介绍如何通过这两种方式访问ClickHouse集群。

背景信息

- 通过原生JDBC访问ClickHouse集群的架构图如下。



- 通过负载均衡器SLB访问ClickHouse集群的架构图如下。



前提条件

- 已创建E-MapReduce的ClickHouse集群，详情请参见创建集群。
- 已创建SLB服务，详情请参见创建和管理CLB实例。

> **注意** 如果是想通过负载均衡器SLB访问ClickHouse集群，则需要创建SLB服务。并且在创建SLB服务时，如果**实例类型**选择的是**私网**，则在选择专有网络时，必须选择与EMR ClickHouse集群相同的VPC。

通过原生JDBC访问ClickHouse集群

1. 获取主机的IP地址。
   i. 登录阿里云E-MapReduce控制台。

ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
iii. 单击上方的**集群管理**页签。
iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
v. 在左侧导航栏，单击**主机列表**。
在此页面您可以查看ClickHouse集群的IP地址。

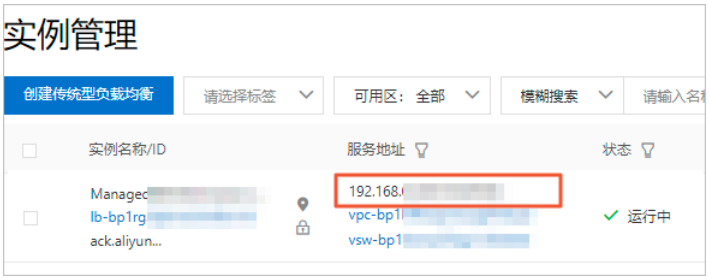
2. 配置JDBC以访问ClickHouse集群，详情请参见ClickHouse JDBC driver。

通过负载均衡器SLB访问ClickHouse集群

1. 配置SLB服务，详情请参见配置实例。
通常情况下，ClickHouse使用SLB服务仅需要配置HTTP及TCP两种协议的监听，如果您有需要，也可以配置HTTPS的监听。配置监听详情，请参见添加TCP监听、添加HTTP监听和添加HTTPS监听。

> 注意
> - TCP监听所使用的虚拟服务器组，其端口应为ClickHouse通过TCP连接到服务器的端口，默认为9000。您可以在EMR控制台ClickHouse服务的**配置**页面，在搜索区域搜索**tcp_port**参数，参数值即为TCP端口。
> - HTTP监听所使用的虚拟服务器组，其端口应为ClickHouse通过HTTP连接到服务器的端口，默认为8123。您可以在EMR控制台ClickHouse服务的**配置**页面，在搜索区域搜索**http_port**参数，参数值即为HTTP端口。

2. 在**实例管理**页面，获取SLB的服务地址。



3. 配置JDBC以访问ClickHouse集群，详情请参见ClickHouse JDBC driver。

5.2.3. ClickHouse运维

5.2.3.1. 日志配置说明

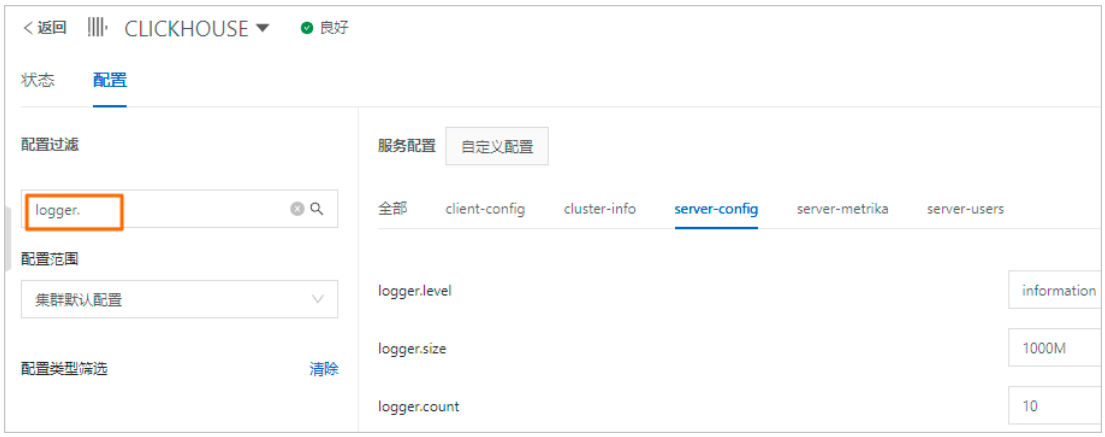
E-MapReduce（简称EMR）支持在控制台查看或配置日志参数，也支持在命令行中设置参数。本文为您介绍ClickHouse服务的日志配置。

前提条件

已创建OLAP集群，且选择了ClickHouse服务，详情请参见创建集群。

Clickhouse控制台日志配置

您可以在ClickHouse服务**配置**页面的**服务配置**区域，在*server-config*页签中查看或修改配置，或者在ClickHouse服务的**配置**页面，在搜索区域搜索**logger.**，即可查看或修改所有的日志配置项。



参数	描述
logger.level	日志的等级，默认等级为information。可以配置的等级从严格到宽松依次为 • none：关闭日志。 • fatal：致命信息。 • critical：危险信息。 • error：错误信息。 • warning：警告信息。 • notice：普通但需要注意的信息。 • information（默认值）：重要或者您感兴趣的信息。 • debug：调试信息。 • trace：程序执行路径跟踪信息。
logger.size	日志文件的大小。当文件达到该参数设置的值时，ClickHouse会将其存档并重命名，并创建一个新的日志文件。默认值为**1000M**。
logger.count	存档的ClickHouse日志文件个数。当存档的日志文件个数达到该参数设置的值时，ClickHouse会将最早的存档删除。默认值为**10**。

ClickHouse客户端日志配置

您可以通过配置客户端日志，来接收来自服务端的日志，默认接收fatal级别的日志。

1. 通过SSH方式登录集群，详情请参见登录集群。
2. 基本操作示例。
   - 查看每次执行的日志。
     a. 执行以下命令，进入ClickHouse客户端。

     ```
     clickhouse-client -h core-1-1 -m
     ```

     > **说明** 本示例登录core-1-1节点，如果您有多个Core节点，可以登录任意一个节点。

     b. 您可以执行以下命令，设置参数send_logs_level查看每次执行的日志。

     ```
     set send_logs_level='debug';
     ```

     返回信息如下所示。

     ```
     SET send_logs_level = 'debug'
     Ok.
     0 rows in set. Elapsed: 0.002 sec.
     ```

   - 在启动ClickHouse客户端时，您可以执行以下命令，将日志保存到指定的文件中。

     ```
     clickhouse-client -h core-1-1 -m --send_logs_level=trace --log-level=trace --server_logs_file='/tmp/query.log'
     ```

5.2.3.2. 系统表说明

系统表存储于System数据库中，仅提供数据读取功能，不能被删除或更改，但可以对其进行分离（detach）操作。大多数系统表将其数据存储在RAM中，一个ClickHouse服务在刚启动时便会创建此类系统表。本文为您介绍E-MapReduce（简称EMR）中常用的系统表。

背景信息

常用系统表如下：

- system.clusters
- system.query_log
- system.zookeeper
- system.replicas
- system.storage_policies
- system.disks

system.clusters

该表包含了配置文件中可用的集群及其服务器的信息。

参数	数据类型	描述
cluster	String	集群名。
shard_num	UInt32	集群中的分片数，从1开始。
shard_weight	UInt32	写数据时该分片的相对权重。
replica_num	UInt32	分片的副本数量，从1开始。
host_name	String	配置中指定的主机名。
host_address	String	从DNS获取的主机IP地址。
port	UInt16	连接到服务器的端口。
user	String	连接到服务器的用户名。
errors_count	UInt32	此主机无法访问副本的次数。
slowdowns_count	UInt32	在与对端请求建立连接时导致副本更改的slowdown的次数。
estimated_recovery_time	UInt32	在复制副本错误计数归零并被视为恢复正常之前剩余的秒数。

示例：您可以执行以下命令，查看表信息。

```
SELECT * FROM system.clusters LIMIT 2 FORMAT Vertical;
```

返回信息如下。

```
Row 1:
──────
cluster:                 cluster_emr
shard_num:               1
shard_weight:            1
replica_num:             1
host_name:               emr-header-1.cluster-24****
host_address:            192.168.**.**
port:                    9000
is_local:                1
user:                    default
default_database:
errors_count:            0
estimated_recovery_time: 0
Row 2:
──────
cluster:                 cluster_emr
shard_num:               1
shard_weight:            1
replica_num:             2
host_name:               emr-worker-1.cluster-24****
host_address:            192.168.**.**
port:                    9000
is_local:                0
user:                    default
default_database:
errors_count:            0
estimated_recovery_time: 0
```

system.query_log

该表包含了已执行查询的相关信息。例如，开始时间、处理持续时间和错误消息。

system.query_log表中记录了两种查询：

- 客户端直接运行的初始查询。
- 由其它查询启动的子查询（用于分布式查询执行）。 对于这些类型的查询，有关父查询的信息显示在 `initial_*` 列。

根据查询的状态（请参见 `type` 列），每个查询在查询日志表中创建一行或两行记录：

- 如果查询执行成功，则会创建 `type` 为 `QueryStart` 和 `QueryFinish` 的两行记录信息。
- 如果在查询处理期间发生错误，则会创建 `type` 为 `QueryStart` 和 `ExceptionWhileProcessing` 的两行记录信息。
- 如果在启动查询之前发生错误，则会创建 `type` 为 `ExceptionBeforeStart` 的一行记录信息。

参数	数据类型	描述
type	Enum8	执行查询时的事件类型。取值如下： • `'QueryStart' = 1` ：查询成功启动。 • `'QueryFinish' = 2` ：查询成功完成。 • `'ExceptionBeforeStart' = 3` ：查询执行前有异常。 • `'ExceptionWhileProcessing' = 4` ：查询执行期间有异常。
event_date	Date	查询开始日期。
event_time	DateTime	查询开始时间。
event_time_microseconds	DateTime64	以微秒精度查询开始时间。
query_start_time	DateTime	查询执行的开始时间。
query_start_time_microseconds	DateTime64	以微秒精度查询执行的开始时间。
query_duration_ms	UInt64	查询消耗的时间。单位为毫秒。
read_rows	UInt64	从参与了查询的所有表和表函数读取的总行数。包括常用的子查询，IN和JOIN的子查询。对于分布式查询 `read_rows` 包括在所有副本上读取的行总数。每个副本发送它的 `read_rows` 值，并且查询的发起方将所有接收到的和本地的值汇总。 缓存卷不会影响此值。

参数	数据类型	描述
read_bytes	UInt64	从参与了查询的所有表和表函数读取的总字节数。包括常用的子查询，IN和JOIN的子查询。对于分布式查询 `read_bytes` 包括在所有副本上读取的字节总数。 每个副本发送它的 `read_bytes` 值，并且查询的发起方将所有接收到的值和本地的值汇总。 缓存卷不会影响此值。
written_rows	UInt64	对于INSERT查询，为写入的行数。 对于其它查询，值为0。
written_bytes	UInt64	对于INSERT查询时，为写入的字节数。 对于其它查询，值为0。
result_rows	UInt64	SELECT查询结果的行数，或INSERT的行数。
result_bytes	UInt64	存储查询结果的RAM量。
memory_usage	UInt64	查询使用的内存。
query	String	查询语句。
exception	String	异常信息。
exception_code	Int32	异常码。
stack_trace	String	如果查询成功完成，则为空字符串。
is_initial_query	UInt8	查询类型。取值如下： • 0：由另一个查询发起的，作为分布式查询的一部分。 • 1：客户端发起的查询。
user	String	发起查询的用户。
query_id	String	查询ID。
address	Ipv6	发起查询的客户端IP地址。
port	UInt16	发起查询的客户端端口。
initial_user	String	初始查询的用户名（用于分布式查询执行）。
initial_query_id	String	初始查询的ID（用于分布式查询执行）。
initial_address	Ipv6	运行父查询的IP地址。
initial_port	UInt16	进行父查询的客户端端口。
interface	UInt8	发起查询的接口。取值如下： • 1：TCP • 2：HTTP
os_user	String	运行 `clickhouse-client` 的操作系统的用户名。
client_hostname	String	运行 `clickhouse-client` 或其他TCP客户端的机器的主机名。
client_name	String	`clickhouse-client` 或其他TCP客户端的名称。
client_revision	UInt32	`clickhouse-client` 或其他TCP客户端的Revision。
client_version_major	UInt32	`clickhouse-client` 或其他TCP客户端的Major Version。
client_version_minor	UInt32	`clickhouse-client` 或其他TCP客户端的Minor Version。
client_version_patch	UInt32	`clickhouse-client` 或其他TCP客户端的Patch component。
http_method	UInt8	发起查询的HTTP方法。取值如下： • 0：TCP接口的查询 • 1：GET • 2：POST
http_user_agent	String	HTTP查询中传递的HTTP请求头UserAgent。

参数	数据类型	描述
quota_key	String	在quotas配置里设置的quota key. 详细信息可以参见配额。
revision	UInt32	ClickHouse revision。
ProfileEvents	Map（String，UInt64）	其它事件的指标，可以在表system.events中找到相关的描述。
Settings	Map（String，String）	客户端运行查询时更改的设置。要启用对设置的日志记录更改，请将 `log_query_settings` 参数设置为1。
thread_ids	Array（UInt64）	参与查询的线程数。
Settings.Names	Array（String）	客户端运行查询时更改的设置的名称。 要启用对设置的日志记录更改，请将 `log_query_settings` 参数设置为1。
Settings.Values	Array（String）	Settings.Names列中列出的设置的值。

示例：您可以执行以下命令，查看表信息。

```
SELECT * FROM system.query_log LIMIT 1 FORMAT Vertical;
```

返回信息如下。

```
Row 1:
──────
type:                 QueryStart
event_date:           2021-08-12
event_time:           2021-08-12 14:11:58
query_start_time:     2021-08-12 14:11:58
query_duration_ms:    0
read_rows:            0
read_bytes:           0
written_rows:         0
written_bytes:        0
result_rows:          0
result_bytes:         0
memory_usage:         0
current_database:     default
query:                SELECT * FROM system.query_log LIMIT 1 FORMAT Vertical;
exception_code:       0
exception:
stack_trace:
is_initial_query:     1
user:                 default
query_id:             08e1336c-696f-4fad-b01b-255b77e56b1f
address:              ::ffff:127.0.0.1
port:                 60500
initial_user:         default
initial_query_id:     08e1336c-696f-4fad-b01b-255b77e56b1f
initial_address:      ::ffff:127.0.0.1
initial_port:         60500
interface:            1
os_user:              root
client_hostname:      emr-header-1.cluster-235053
client_name:          ClickHouse
client_revision:      54438
client_version_major: 20
client_version_minor: 8
client_version_patch: 12
http_method:          0
http_user_agent:
quota_key:
revision:             54438
thread_ids:           []
ProfileEvents.Names:  []
ProfileEvents.Values: []
Settings.Names:       ['use_uncompressed_cache','load_balancing','max_memory_usage']
Settings.Values:      ['0','random','10000000000']
```

system.zookeeper

该表可以查到ZooKeeper中的节点信息。

如果未配置ZooKeeper，则该表不存在。 允许从配置中定义的ZooKeeper集群读取数据。 查询必须具有 `path=` 条件，或使用WHERE子句设置了path IN条件，这对应于ZooKeeper中要获取数据的子对象的路径。

查询语句 `SELECT * FROM system.zookeeper WHERE path = '/clickhouse'` ，输出 `/clickhouse` 节点的对所有子路径的数据。如果需要输出所有根节点的数据，请写入路径为 `'/'` 。如果 `path` 中指定的路径不存在，则将提示异常。

查询语句 `SELECT * FROM system.zookeeper WHERE path IN ('/', '/clickhouse')` ，输出 `/` 和 `/clickhouse` 节点上所有子节点的数据。如果 `path` 中指定的路径不存在，则将提示异常。它可以用于一批ZooKeeper路径的查询。

参数	数据类型	描述
name	String	节点的名字。
path	String	节点的路径。
value	String	节点的值。
dataLength	Int32	节点的值长度。
numChildren	Int32	子节点的个数。
czxid	Int64	创建该节点的事务ID。
mzxid	Int64	最后修改该节点的事务ID。
pzxid	Int64	最后删除或者增加子节点的事务ID。
ctime	DateTime	节点的创建时间。
mtime	DateTime	节点的最后修改时间。
version	Int32	节点版本和节点被修改的次数。
cversion	Int32	最后删除或者增加子节点的事务ID。
aversion	Int32	ACL的修改次数。
ephemeralOwner	Int64	针对临时节点，拥有该节点的事务ID。

示例：您可以执行以下命令，查看表信息。

```
SELECT *
FROM system.zookeeper
WHERE path = '/clickhouse/tables/01-08/visits/replicas'
FORMAT Vertical
```

返回信息如下。

```
Row 1:
──────
name:           example01-08-1.yandex.ru
value:
czxid:          932998691229
mzxid:          932998691229
ctime:          2015-03-27 16:49:51
mtime:          2015-03-27 16:49:51
version:        0
cversion:       47
aversion:       0
ephemeralOwner: 0
dataLength:     0
numChildren:    7
pzxid:          987021031383
path:           /clickhouse/tables/01-08/visits/replicas
Row 2:
──────
name:           example01-08-2.yandex.ru
value:
czxid:          933002738135
mzxid:          933002738135
ctime:          2015-03-27 16:57:01
mtime:          2015-03-27 16:57:01
version:        0
cversion:       37
aversion:       0
ephemeralOwner: 0
dataLength:     0
numChildren:    7
pzxid:          987021252247
path:           /clickhouse/tables/01-08/visits/replicas
```

system.replicas

该表包含本地服务所有复制表的信息和状态，可以用于监控。

参数	数据类型	描述
database	String	数据库名称。
table	String	表名。
engine	String	表引擎名称。
is_leader	UInt8	副本是否是领导者。 一次只有一个副本可以成为领导者。 领导者负责选择要执行的后台合并。 注意 可以对任何可用且在Zookeeper中具有会话的副本执行写操作，不管该副本是否为leader。
can_become_leader	UInt8	副本是否可以当选为领导者。
is_readonly	UInt8	副本是否处于只读模式。 存在以下情形时，开启此配置： • 配置中缺省了zookeeper的部分。 • 在zookeeper重新加载会话时发生未知错误。 • 在会话期间重新初始化了zookeeper。
is_session_expired	UInt8	与ZooKeeper的会话已经过期。用法基本上与 `is_readonly` 相同。
future_parts	UInt32	由于尚未完成的插入或合并而显示的数据部分的数量。
parts_to_check	UInt32	队列中用于验证的part的数量。 如果怀疑part可能损坏了，则将其放入验证队列。
zookeeper_path	String	在ZooKeeper中的表数据路径。

参数	数据类型	描述
replica_name	String	ZooKeeper中的副本名称。同一表的不同副本具有不同的名称。
replica_path	String	ZooKeeper中副本数据的路径。 与*zookeeper_path/replicas/replica_path*下的内容相同。
columns_version	Int32	表结构的版本号。 表示执行ALTER的次数。 如果副本有不同的版本，则意味着部分副本还没有进行所有的更改。
queue_size	UInt32	等待执行的操作的队列大小。 操作包括插入数据块、合并和某些其它操作。它通常与 `future_parts` 一致。
inserts_in_queue	UInt32	需要插入的数据块的数量。 数据的插入通常很快。 如果该数值很大，则说明有问题。
merges_in_queue	UInt32	等待进行合并的数量。 有时合并时间很长，因此此值可能长时间大于零。
part_mutations_in_queue	UInt32	等待进行的突变的数量。
queue_oldest_time	DateTime	如果 `queue_size` 大于0，则显示何时将最早的操作添加到队列。
inserts_oldest_time	DateTime	
merges_oldest_time	DateTime	
part_mutations_oldest_time	DateTime	
log_max_index	UInt64	一般活动日志中的最大条目数。 注意 存在与ZooKeeper的活动会话时才具有非零值。
log_pointer	UInt64	副本复制到其执行队列的常规活动日志中的最大条目数，再加一。 如果log_pointer比log_max_index小，则说明有问题。 注意 存在与ZooKeeper的活动会话时才具有非零值。
last_queue_update	DateTime	上次更新队列的时间。 注意 存在与ZooKeeper的活动会话时才具有非零值。
absolute_delay	UInt64	当前副本最大延迟时间。单位为秒。 注意 存在与ZooKeeper的活动会话时才具有非零值。
total_replicas	UInt8	此表的已知副本总数。
active_replicas	UInt8	在ZooKeeper中具有会话的此表的副本的数量（即正常运行的副本的数量）。

示例：您可以执行以下命令，查看表信息。

```
SELECT * FROM system.replicas WHERE table = 'visits' FORMAT Vertical
```

返回信息如下。

```
Row 1:
──────
database:                   cltest
table:                      flat_xl_customer_local
engine:                     ReplicatedMergeTree
is_leader:                  1
can_become_leader:          1
is_readonly:                0
is_session_expired:         0
future_parts:               0
parts_to_check:             0
zookeeper_path:             /cltest/cluster_emr-1/flat_xl_customer_local
replica_name:               emr-header-1.cluster-235053
replica_path:               /cltest/cluster_emr-1/flat_xl_customer_local/replicas/emr-header-1.cluster-235053
columns_version:            -1
queue_size:                 0
inserts_in_queue:           0
merges_in_queue:            0
part_mutations_in_queue:    0
queue_oldest_time:          1970-01-01 08:00:00
inserts_oldest_time:        1970-01-01 08:00:00
merges_oldest_time:         1970-01-01 08:00:00
part_mutations_oldest_time: 1970-01-01 08:00:00
oldest_part_to_get:
oldest_part_to_merge_to:
oldest_part_to_mutate_to:
log_max_index:              100157
log_pointer:                100158
last_queue_update:          1970-01-01 08:00:00
absolute_delay:             0
total_replicas:             2
active_replicas:            2
zookeeper_exception:
```

system.storage_policies

该表包含了有关存储策略和卷的优先级相关的信息。

参数	数据类型	描述
policy_name	String	存储策略的名称。
volume_name	String	存储策略中定义的卷的名称。
volume_priority	UInt64	配置中定义的卷的优先级。
disks	String	存储策略中定义的磁盘名称。
max_data_part_size	UInt64	可以存储在磁盘卷上的数据part的最大值。
move_factor	Float64	可用磁盘空间的比率。当比率超过配置参数的值时，数据将会被移动到下一个卷。

示例：您可以执行以下命令，查看表信息。

```
SELECT * FROM system.storage_policies
```

返回信息如下。

```
┌policy_name─┬volume_name─┬volume_priority─┬disks─────────────┬volume_type─┬max_data_part_size┬move_factor┐
│ default      │ default     │              1 │ ['default']                      │ JBOD        │                 0 │
0 │
│ hdd_in_order │ single      │              1 │ ['disk1','disk2','disk3','disk4'] │ JBOD        │                 0 │
0.1 │
└──────────┴─────┴────────┴──────────────┴─────┴────────┴────────┘
```

system.disks

该表包含了配置中定义的磁盘信息。

参数	数据类型	描述
name	String	配置的磁盘名称。
path	String	文件系统中挂载的磁盘路径。
free_space	UInt64	磁盘上的可用空间（Bytes）。
total_space	UInt64	磁盘的总空间（Bytes）。
keep_free_space	UInt64	磁盘上需要保持空闲的空间。定义在磁盘配置的keep_free_space_bytes 参数中。

示例：您可以执行以下命令，查看表信息。

```
SELECT * FROM system.disks;
```

返回信息如下。

```
┌─name─┬─path────────────┬─free_space─┬─total_space─┬─keep_free_space─┬─type──┐
│ default │ /var/lib/clickhouse/   │ 17236594688 │ 84014424064 │               0 │ local │
│ disk1   │ /mnt/disk1/clickhouse/ │ 17226108928 │ 84003938304 │        10485760 │ local │
│ disk2   │ /mnt/disk2/clickhouse/ │ 28623364096 │ 84003938304 │        10485760 │ local │
│ disk3   │ /mnt/disk3/clickhouse/ │ 34770505728 │ 84003938304 │        10485760 │ local │
│ disk4   │ /mnt/disk4/clickhouse/ │ 59107045376 │ 84003938304 │        10485760 │ local │
└─────┴─────────────┴─────────┴─────────┴─────────┴─────┘
```

5.2.3.3. 监控

E-MapReduce（简称EMR）上的ClickHouse集群提供了完善的监控体系，分为服务监控和节点监控两个维度。本文为您介绍如何查看服务监控和节点监控。

背景信息

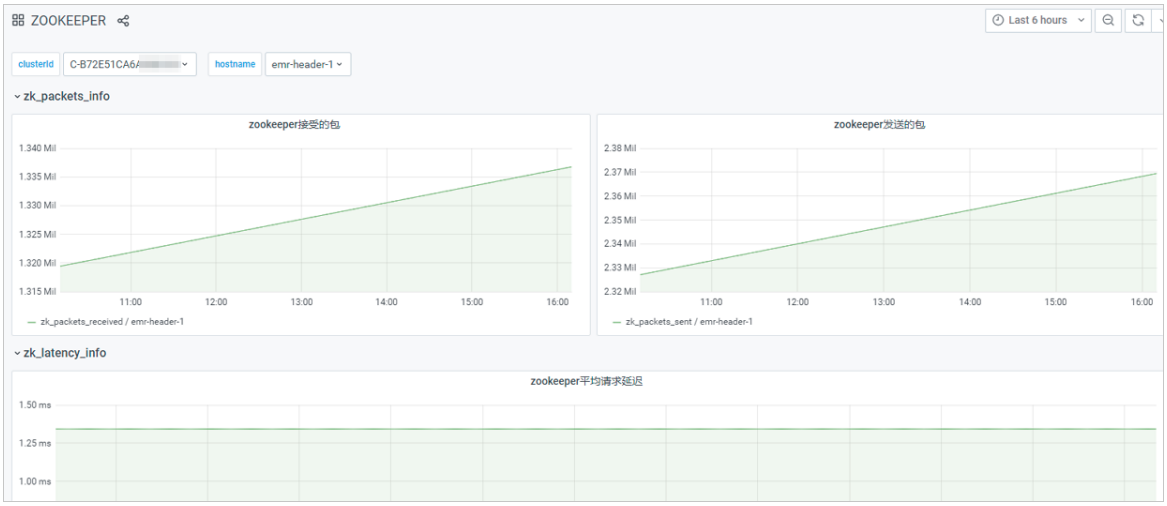
ClickHouse集群服务监控只有ClickHouse和Zookeeper服务。在集群监控大盘的集群指标页面，可以查看不同组件的监控数据，可以根据需求选择时间粒度。

前提条件

已创建ClickHouse集群，详情请参见创建集群。

查看服务监控

1. 进入监控大盘页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**服务管理**。
   iv. 在**服务管理**页面，单击ClickHouse服务区域的**配置**。
   v. 单击右侧的**服务监控**。
2. 在**集群指标**页面，查看Zookeeper服务和ClickHouse服务的监控数据。
   - 单击**HOST**，选择**ZOOKEEPER**服务。



- 单击**HOST**，选择**CLICKHOUSE**服务。

ClickHouse的监控指标分为3组，分别来自ClickHouse的三个系统表metrics、events和asynchronous_metrics。

查看节点监控

查看节点监控又分为节点部署状态和查看节点详细监控指标。

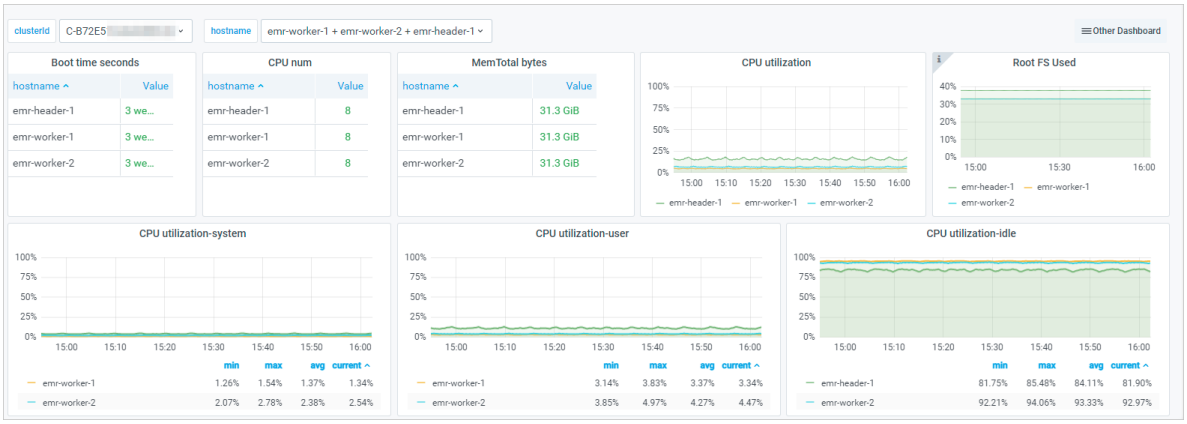
1. 进入集群服务页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
2. 查看Zookeeper服务或ClickHouse服务的监控数据。
   i. 在**集群服务**页面，单击Zookeeper服务区域的**状态**。
   ClickHouse服务，需要单击ClickHouse服务区域的**状态**。

ii. 单击**组件列表**区域的图标。

该页面展示了部署服务的进程的实时状态，方便您监控组件的情况。

1. 进入监控大盘页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**服务管理**。
   iv. 在**服务管理**页面，单击ClickHouse服务区域的**配置**。
   v. 单击右侧的**服务监控**。
2. 从**clusterid**下拉列表中，选择您创建的集群ID，还可以从**hostname**下拉列表中，选择节点名称。

   该页面展示了节点具体的监控指标的情况，包括Disk、内存、网络和TCP等信息。



查看部署状态

查看节点详细监控指标

5.2.3.4. 配置项说明

阿里云E-MapReduce（简称EMR）的ClickHouse集群中，主要提供了四种服务配置项以配置ClickHouse集群，包括客户端配置、服务端配置、用户权限配置和拓展配置。本文为您介绍ClickHouse服务的客户端配置、服务端配置和拓展配置。

背景信息

ClickHouse集群提供的四种服务配置项信息如下表。

配置项	详情
客户端配置	client-config
服务端配置	server-config
拓展配置	server-metrika
用户权限配置	访问权限控制

配置项	详情

前提条件

已创建E-MapReduce的ClickHouse集群，详情请参见创建集群。

注意事项

由于ClickHouse配置采用XML文件形式，所以相对比较灵活，可以进行多层嵌套。自定义配置时规则如下：

- 如果配置可以直接填写在*yandex*标签下，则可以直接新增。
- 如果配置包含多层嵌套，每层需要直接使用半角句号（.）进行连接。

  例如，在**server-users**页签中，添加新的用户aliyun，可以设置参数为**users.aliyun.password**，参数值为密码，您可以自定义。
- 自定义配置中，请勿使用XML类型作为参数或者参数值。

client-config

该服务配置项用于生成clickhouse-client所使用的*config.xml*文件。您可以在EMR控制台ClickHouse服务的**配置**页面，单击**client-config**页签，查看以下参数。

配置项	描述
user	设置*clickhouse-client*使用的用户。默认值为default。
password	设置*clickhouse-client*进程使用的用户密码。默认值为空。
prompt_by_server_display_name.production	在使用*clickhouse-client*的时候，允许自定义提示符。设置这个选项以配置在不同的display_name情况下的提示符，例如在**server-config**选项中设置display_name为default时，所展示的提示符为**prompt_by_server_display_name.default**中所设置的值。设置提示符允许使用的颜色，请参见Color prompts with readline和tip_colors_and_formatting。
prompt_by_server_display_name.default	
prompt_by_server_display_name.test	

server-config

该服务配置项用来生成*clickhouse-server*进程所使用的*config.xml*文件。您可以在EMR控制台ClickHouse服务的**配置**页面，单击**server-config**页签，查看以下参数。

配置项	描述
tcp_port	通过TCP协议与客户端通信的端口。默认值为**9000**。
logger.count	存档的ClickHouse日志文件个数。当存档的日志文件个数达到该参数设置的值时，ClickHouse会将最早的存档删除。默认值为**10**。
logger.level	日志的等级，默认等级为information。可以配置的等级从严格到宽松依次为none（关闭日志）、fatal、critical、error、warning、notice、information、debug和trace。
logger.size	日志文件的大小。当文件达到该参数设置的值时，ClickHouse会将其存档并重命名，并创建一个新的日志文件。默认值为**1000M**。
distributed_ddl.path	该参数指定了ZooKeeper中用于存储DDL查询队列的路径。默认值为*/clickhouse/task_queue/ddl*。默认情况下，ClickHouse的操作CREATE、DROP、ALTER和RENAME等都只会影响正在处理查询的这一台机器。设置distributed_ddl相关参数，允许ClickHouse的查询运行在集群中（当且仅当ZooKeeper被使用时）。
default_database	默认数据库。默认值为default。
uncompressed_cache_size	表引擎使用MergeTree时，为解压后的block所建立的Cache大小。默认值为0。 如果设置为0，则表示不启用Cache。
timezone	设置服务器的时区。默认值为*Asia/Shanghai*。
max_session_timeout	Session最大超时时间，单位为秒。默认值为3600。
default_session_timeout	Session默认超时时间，单位为秒。默认值为60。
max_concurrent_queries	可以同时处理查询的最大数量。默认值为0。
keep_alive_timeout	ClickHouse在关闭连接之前等待传入请求的秒数，单位为秒。默认值为10。

配置项	描述
http_port	通过HTTP连接到服务器的端口。默认值为8123。 ClickHouse官方JDBC也通过此端口访问ClickHouse，请参见clickhouse-jdbc。
listen_host	ClickHouse服务器所监听的IP地址。可以指定IPv4和IPv6的地址，如果指定::则代表允许所有地址。可以设置多个IP地址，多个IP地址以逗号（,）分割。例如，*127.0.0.1,localhost*。默认值为*0.0.0.0*。
default_profile	默认会使用的profile。默认值为default。
mark_cache_size	表引擎使用MergeTree时，mark索引所使用的Cache大小的近似值。默认值为5368709120，单位为Byte。
merge_tree.allow_remote_fs_zero_copy_replication	设置为true，以在Replicated*MergeTree使用DiskHDFS等远程存储时利用其自身的多副本进行备份，ClickHouse的一个Shard下的多个副本中的数据仅会备份元数据。
transaction.enable_public_ip	ClickHouse Server中Transaction需要一个用于标识自身的IP地址，默认使用私网IP地址。 设置此参数值为true，以使用公网IP地址标识自身，但需要所有节点均开启公网IP。

server-metrika

该服务配置项用于生成*metrika.xml*文件，其默认被ClickHouse Server的config所引用。您可以在EMR控制台ClickHouse服务的**配置**页面，在默认的**server-metrika**页签，查看以下参数。

配置项	描述
clickhouse_compression	为使用MergeTree相关引擎的表设置数据压缩，详细信息请参见Server Settings。默认值为空。 如果需要使用ClickHouse的压缩，请自行添加配置。
storage_configuration	用来指定自定义的磁盘信息。
zookeeper_servers	用来配置ClickHouse集群所使用的ZooKeeper信息。默认值为创建ClickHouse集群时同时创建的ZooKeeper的值。多个ZooKeeper节点时，请使用英文逗号（,）进行分隔，例如，`emr-header-1.cluster-12345:2181,emr-worker-1.cluster-12345:2181,emr-worker-2.cluster-12345:2181`。
quotas_default	ClickHouse允许配置不同的quota以灵活的使用不同的资源限制。修改该配置项可以修改名为default的quota设置。如果需要添加新的quota设置，您可以添加自定义设置。

相关文档

ClickHouse参数的详情信息，可以参见以下官方文档：

- Server Settings
- Settings
- MergeTree tables settings

后续步骤

如果需要修改或添加配置项，请参见管理配置项。

5.2.3.5. 访问权限控制

阿里云E-MapReduce（简称EMR）的ClickHouse集群中，主要提供了四种服务配置项以配置ClickHouse集群，包括客户端配置、服务端配置、用户权限配置和拓展配置。本文为您介绍ClickHouse服务的用户权限配置。

背景信息

用户访问权限配置在*server-users*和*server-metrika*文件中，包含users、profiles和quotas三部分配置。详细配置信息：

- users配置
- profiles配置
- quotas配置

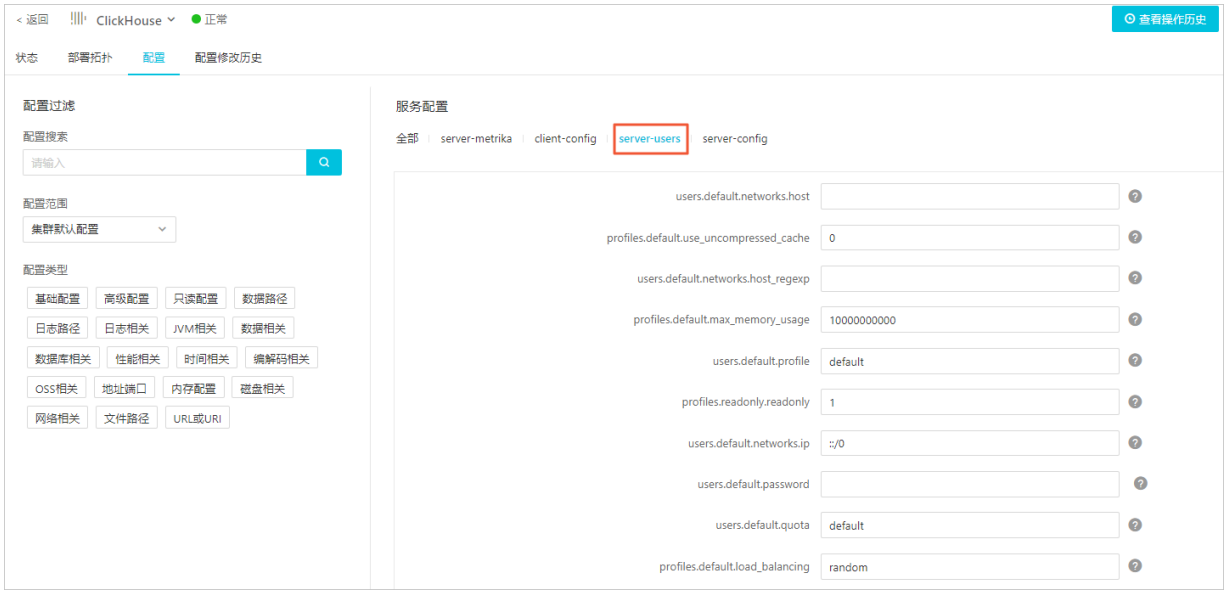
> **说明** ClickHouse服务的客户端配置、服务端配置和拓展配置的详细信息，请参见配置项说明。

前提条件

已创建E-MapReduce的ClickHouse集群，详情请参见创建集群。

users配置

您可以在ClickHouse服务配置页面的服务配置区域，查看或修改配置。users配置在*server-users*页签中。



参数	描述
users.default.networks.host	default用户允许访问的主机名，默认值为空。 多个主机名时，可以使用英文逗号（,）分隔。
users.default.networks.host_regexp	default用户允许访问的主机名的正则表达式，默认值为空。 多个表达式时，可以使用英文逗号（,）分隔。
users.default.networks.ip	default用户允许访问的IP地址。默认值为**::/0**， 表示允许所有IP地址访问。 多个IP地址时，可以使用英文逗号（,）分隔。 **注意** 请确保users.default.networks.ip、users.default.networks.host和users.default.networks.host_regexp三个参数中，至少有一个参数值不为空，否则可能会导致网络不通。
users.default.profile	default用户默认使用的profile名称。默认值为**default**。
users.default.password	ClickHouse Server中default用户的密码。默认值为空。 **注意** 不建议使用此配置。建议您添加自定义参数users.default.password_sha256_hex或users.default.password_double_sha1_hex以设置密码。 • users.default.password_sha256_hex：SHA256产生的密钥的16进制字符串。 • users.default.password_double_sha1_hex：通过两次SHA1产生的密钥的16进制字符串。
users.default.quota	default用户默认使用的quota配置。默认值为**default**。

profiles配置

您可以在ClickHouse服务配置页面的服务配置区域，查看或修改配置。profiles配置在*server-users*页签中。

参数	描述
profiles.default.max_memory_usage	用于设置名为default的profile中max_memory_usage的值。修改该参数可以设置单个查询时所能够使用的最大内存。 默认为10,000,000,000，单位为byte。
profiles.default.use_uncompressed_cache	用于设置名为default的profile中use_uncompressed_cache的值。 • 1（默认值）：表示使用未压缩数据块的缓存。 • 0：表示不使用未压缩数据块的缓存。
profiles.default.load_balancing	用于设置名为default的profile中*load_balancing*的值。可以设置在分布式查询处理中选择副本的策略。 策略详细信息，请参见Settings。
profiles.readonly.readonly	用于设置名为readonly的profile中readonly的值。 • 1：使用名为readonly的profile，只允许执行读操作。 • 0：不开启readonly。

quotas配置

您可以在ClickHouse服务配置页面的服务配置区域，查看或修改配置。quotas配置在*server-metrika*页签中。

quotas_default：ClickHouse允许配置不同的quota以灵活的使用不同的资源限制。修改该配置项可以修改名为default的quota设置（**users.default.quota**）。如果需要添加新的quota设置，你可以单击右上角的**自定义配置**，详细操作请参见添加组件参数。

5.2.3.6. 扩容ClickHouse集群

随着业务量的增长，当ClickHouse集群已不能满足业务需求时，需要扩容ClickHouse集群。ClickHouse集群支持分片扩容和副本扩容两种方式，当集群容量不能满足业务需求时，您可以进行分片扩容；当集群并发访问量不能满足业务需求时，您可以进行副本扩容。本文为您介绍如何扩容ClickHouse集群。

前提条件

已创建ClickHouse集群，详情请参见创建集群。

注意事项

- 扩容的机器数必须是分区或副本的倍数。
- 集群扩容，仅支持表结构迁移，不支持数据迁移。
- default数据库下的表结构不支持迁移。
- 分片扩容是直接在原有的集群上增加节点，并在新增节点上创建分布式表和本地表，扩容后新写入的数据按照原有的分布策略进行写入。

操作步骤

1. 进入集群管理页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**节点管理**。
2. 在**节点管理**页面，单击目标机器组操作列的**扩容**。
3. 在**扩容**对话框中，根据实际情况修改相应参数。

参数		描述
扩容方式	**分片(shard)扩容**	当集群容量不能满足业务需求时，可以进行分片扩容。
	副本(replica)扩容	当集群并发访问量不能满足业务需求时，可以进行副本扩容。
增加数量		扩容的机器数必须是分区或副本的倍数。
服务协议		阅读并同意服务条款后，选中即可。

4. 完成上述参数配置后，单击**确定**。

5.2.3.7. 扩容磁盘

当EMR的ClickHouse集群数据存储空间不足时，则需要进行扩容操作。ClickHouse磁盘扩容分为云盘扩容和云盘挂载两种，本文主要为您介绍如何挂载云盘。

背景信息

在使用EMR ClickHouse集群时，尽管在创建前已经做好数据规模的估算了，但随着业务的发展，EMR ClickHouse集群的存储可能依旧不够用，无论是临时扩容以应对临时的高峰（例如，应对大促时期的数据量），或是需要长期扩容以应对日益增长的数据，可能都需要对EMR ClickHouse集群的磁盘进行扩容。

如果ClickHouse集群当前所挂载的数据盘为ESSD云盘、SSD云盘或高效云盘等类型的云盘，则可以直接扩容云盘的容量而不需要修改任何配置。扩容详细信息，请参见扩容磁盘。

前提条件

已在EMR控制台创建了ClickHouse集群，详情请参见创建集群。

操作步骤

1. 进入实例的云盘页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**节点管理**。
   iv. 在**主机信息**区域，单击目标实例的**ECS ID**。
2. 在ECS控制台创建云盘，详情请参见创建云盘。
   创建云盘时，**是否挂载**选择为**挂载到ECS实例**。按量付费的集群，**释放设置**需要选择为**云盘随实例释放**。
3. 在目标实例的节点，挂载磁盘。
   i. 登录目标实例节点，详情请参见登录集群。

ii. 执行 `lsblk` 命令，查看未挂载的磁盘。

返回信息如下类似信息。

```
NAME   MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
vdf    253:80   0  40G  0 disk
vdd    253:48   0  80G  0 disk /mnt/disk3
vdb    253:16   0  80G  0 disk /mnt/disk1
vde    253:64   0  80G  0 disk /mnt/disk4
vdc    253:32   0  80G  0 disk /mnt/disk2
vda    253:0    0  80G  0 disk
└─vda1 253:1    0  80G  0 part /
```

MOUNTPOINT下显示为空的即是未挂载的磁盘。

iii. 执行以下命令，格式化磁盘。

```
mkfs.ext4 /dev/vdf
```

> 注意 /dev/vdf为未挂载磁盘的名称，请根据实际情况替换为前一步骤中查找到的未挂载磁盘。

iv. 执行以下命令，创建挂载目录。

```
mkdir -p /mnt/disk5
```

> 注意 命令中的disk5表示disk的名称，可以是5到n，n取决于您需要挂载多少块盘。本文示例是挂载disk5，请您根据实际情况修改。

v. 执行以下命令，挂载磁盘。

```
mount -t ext4 /dev/vdf /mnt/disk5
```

> 注意 当前操作仅保证系统未重启可用，重启后磁盘不会自动挂载。

vi. 将磁盘挂载信息写入到*/etc/fstab*中，使磁盘在操作系统重启后自动挂载。

a. 执行以下命令，打开*fstab*文件。

```
vim /etc/fstab
```

b. 添加以下内容至文件末尾。

```
/dev/vdf /mnt/disk5 ext4 defaults,noatime,nofail 0 0
```

4. 在挂载目录中创建*clickhouse*数据目录。

i. 执行以下命令，创建*clickhouse*数据目录。

```
mkdir -p /mnt/disk5/clickhouse
```

ii. 执行以下命令，修改目录权限。

```
chown -R clickhouse:clickhouse /mnt/disk5/clickhouse/
```

5. 在EMR集群中，对所有需要扩容的机器重复执行步骤1~步骤4。

登录Core节点可以参考以下步骤：

i. 在Master节点上切换到emr-user账号。

```
su emr-user
```

ii. 免密码登录到对应的Core节点。

```
ssh core-1-1
```

iii. 通过sudo获得root权限。

```
sudo su - root
```

6. 在EMR控制台，修改ClickHouse配置项。

i. 进入ClickHouse配置页面。
   a. 登录EMR on ECS控制台。
   b. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   c. 在**集群服务**页面，单击ClickHouse服务的**配置**。
ii. 在**服务配置**区域，单击**server-metrika**页签，修改参数**storage_configuration**的参数值。

   具体修改点为以下两处：
   a. 在**disks**中增加扩容的磁盘信息。

```
<disk5>
    <path>/mnt/disk5/clickhouse/</path>
    <keep_free_space_bytes>10485760</keep_free_space_bytes>
  </disk5>
```

   b. 在**single**中增加扩容的磁盘名称。

```
<disk>disk5</disk>
```

   以EMR-5.5.1版本为例，修改后**storage_configuration**的参数值信息如下所示。

```
<disks>
  <disk1>
    <path>/mnt/disk1/clickhouse/</path>
    <keep_free_space_bytes>10485760</keep_free_space_bytes>
  </disk1>
  <disk2>
    <path>/mnt/disk2/clickhouse/</path>
    <keep_free_space_bytes>10485760</keep_free_space_bytes>
  </disk2>
  <disk3>
    <path>/mnt/disk3/clickhouse/</path>
    <keep_free_space_bytes>10485760</keep_free_space_bytes>
  </disk3>
  <disk4>
    <path>/mnt/disk4/clickhouse/</path>
    <keep_free_space_bytes>10485760</keep_free_space_bytes>
  </disk4>
  <disk5>
    <path>/mnt/disk5/clickhouse/</path>
    <keep_free_space_bytes>10485760</keep_free_space_bytes>
  </disk5>
</disks>
<policies>
  <default>
    <volumes>
      <single>
        <disk>disk1</disk>
        <disk>disk2</disk>
        <disk>disk3</disk>
        <disk>disk4</disk>
        <disk>disk5</disk>
      </single>
    </volumes>
  </default>
</policies>
```

iii. 保存配置。
   a. 在ClickHouse服务的**配置**页面，单击**保存**。
   b. 在**确认修改**对话框中，输入执行原因，打开**自动更新配置**开关，单击**确定**。
iv. 部署客户端配置。
   a. 在ClickHouse服务的**配置**页面，单击**部署客户端配置**。
   b. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
   c. 在**确认**对话框中，单击**确定**。

7. 检查配置是否完成。
   i. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。

ii. 执行如下命令，启动ClickHouse客户端。

```
clickhouse-client -h core-1-1 -m
```

> **说明** 本示例登录core-1-1节点，如果您有多个Core节点，可以登录任意一个节点。

iii. 执行如下命令，查看磁盘信息。

```
select * from system.disks;
```

返回信息如下所示。

```
┌─name────┬─path──────────────────┬─free_space──┬─total_space─┬─keep_free_space─┬─type──┐
│ default │ /var/lib/clickhouse/  │ 83830616064 │ 84014424064 │               0 │ local │
│ disk1   │ /mnt/disk1/clickhouse/ │ 83820130304 │ 84003938304 │        10485760 │ local │
│ disk2   │ /mnt/disk2/clickhouse/ │ 83928371200 │ 84003938304 │        10485760 │ local │
│ disk3   │ /mnt/disk3/clickhouse/ │ 83928371200 │ 84003938304 │        10485760 │ local │
│ disk4   │ /mnt/disk4/clickhouse/ │ 83928371200 │ 84003938304 │        10485760 │ local │
│ disk5   │ /mnt/disk5/clickhouse/ │ 39782125568 │ 41996746752 │        10485760 │ local │
└─────────┴───────────────────────┴─────────────┴─────────────┴─────────────────┴───────┘
```

iv. 执行如下命令，查看磁盘存储策略。

```
select * from system.storage_policies;
```

返回信息如下所示。

```
┌─policy_name─┬─volume_name─┬─volume_priority─┬─disks─────────────────────────────────────┬─volume_type─┬─max_data_
part_size─┬─move_factor─┬─prefer_not_to_merge─┐
│  default    │ single      │               1│ ['disk1','disk2','disk3','disk4','disk5'] │ JBOD        │
0 │           0 │                   0 │
└─────────────┴─────────────┴─────────────────┴───────────────────────────────────────────┴─────────────┴──────────
──┴─────────────┴─────────────────────┘
```

当回显信息如上文所示时，表示磁盘扩容操作完成。

5.2.3.8. 缩容磁盘

无论是本地盘还是云盘，目前都不支持容量的降低，如果希望进行数据盘的缩容，只能通过卸载数据盘来实现。本文介绍卸载数据盘的流程和具体操作。

前提条件

已在EMR控制台创建了ClickHouse集群，详情请参见创建集群。

使用限制

- 仅支持按量付费类型的集群进行缩容操作，包年包月类型请提交工单处理。
- 本地盘不支持卸载。

注意事项

- 缩容前请暂停向当前ClickHouse Server写入数据。
- 请确保待卸载云盘中的数据皆已移到其他盘中，或云盘中的数据可丢弃。

操作流程

1. 步骤一：从ClickHouse Server中移除数据盘
2. 步骤二：从操作系统中移除数据盘
3. 步骤三：在ECS控制台卸载云盘

步骤一：从ClickHouse Server中移除数据盘

1. 启动ClickHouse客户端。
   i. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。

ii. 执行如下命令，启动ClickHouse客户端。

```
clickhouse-client -h core-1-1 -m
```

> **说明** 本示例登录core-1-1节点，如果您有多个Core节点，可以登录任意一个节点。

2. 执行如下命令，查看数据盘在ClickHouse中的名称。

```
select * from system.disks;
```

返回信息如下所示。

```
┌─name────┬─path──────────────────┬─free_space─┬─total_space─┬─keep_free_space─┬─type──┐
│ default │ /var/lib/clickhouse/  │ 83830616064 │ 84014424064 │               0 │ local │
│ disk1   │ /mnt/disk1/clickhouse/ │ 83820130304 │ 84003938304 │        10485760 │ local │
│ disk2   │ /mnt/disk2/clickhouse/ │ 83928371200 │ 84003938304 │        10485760 │ local │
│ disk3   │ /mnt/disk3/clickhouse/ │ 83928371200 │ 84003938304 │        10485760 │ local │
│ disk4   │ /mnt/disk4/clickhouse/ │ 83928371200 │ 84003938304 │        10485760 │ local │
│ disk5   │ /mnt/disk5/clickhouse/ │ 39782125568 │ 41996746752 │        10485760 │ local │
└─────────┴───────────────────────┴────────────┴─────────────┴─────────────────┴───────┘
```

> **说明** path列中包含了name列的值。例如，path列内容为*/mnt/disk5/clickhouse/*，则在ClickHouse中的名称为disk5。

3. 执行以下命令，查看待移除的数据盘上的part。

```
SELECT name,database,table FROM system.parts where active=1 and disk_name='<disk_name>';
```

本文以移除挂载点在disk5上的数据盘为例，所以例命令中的 `<disk_name>` 为disk5，返回信息如下所示。

```
SELECT
    name,
    database,
    table
FROM system.parts
WHERE (active = 1) AND (disk_name = 'disk5')
┌─name──────────┬─database─┬─table─────┐
│ 202204_33_33_0 │ system   │ query_log │
└────────────────┴──────────┴───────────┘
1 rows in set. Elapsed: 0.002 sec.
```

本示例返回信息中 `202204_33_33_0` 为part的名称，该part是属于default数据库中test表的。

4. 当所有数据均移动到其它磁盘上之后，检查其他磁盘上的data part数量。

i. 执行以下命令，查看磁盘data part数量。

```
SELECT database,table,disk_name,count(1) AS part_number FROM system.parts WHERE database != 'system' GROUP BY datab
ase,table,disk_name ORDER BY part_number DESC;
```

ii. 执行以下命令，按照part_number进行排序，对part_number数量较大的表执行合并操作。

```
optimize table <database_name>.<table_name> final;
```

等待合并完成并清理过期的data part。本示例命令为 `optimize table default.test final;` ，命令中的 `<database_name>` 为数据库名， `<table_name>` 为表名，请您根据实际情况修改。

5. 执行以下命令，将需移除的数据盘上的part移动到继续使用的磁盘上。

```
ALTER TABLE <database_name>.<table_name> MOVE PART '<part_name>' TO DISK '<disk_name>';
```

本示例命令为 `ALTER TABLE default.test MOVE PART 'all_1_140_3' TO DISK 'disk1';` 。命令中的 `<database_name>` 为数据库名， `<table_name>` 为表名， `<part_name>` 为步骤3获取到的part的名称， `<disk_name>` 为不会卸载的盘，请您根据实际情况修改。

> **说明** 如果有多个data part，请重复执行该命令。

6. 在EMR控制台，修改ClickHouse配置项。

i. 进入ClickHouse配置页面。
   a. 登录阿里云E-MapReduce控制台。
   b. 单击上方的**集群管理**页签。
   c. 在**集群管理**页面，单击相应集群所在行的**详情**。
   d. 在左侧导航栏中，选择**集群服务 > ClickHouse**。
   e. 在ClickHouse服务页面，单击**配置**页签。
ii. 将数据盘从ClickHouse配置中移除。
   a. 在ClickHouse服务的**配置**页面，单击**server-metrika**页签。
   b. 找到参数**storage_configuration**，删除参数值中待删除磁盘（本示例为disk5）的所有配置。
iii. 保存配置。
   a. 在ClickHouse服务的**配置**页面，单击**保存**。
   b. 在**确认修改**对话框中，输入执行原因，打开**自动更新配置**开关，单击**确定**。
iv. 部署客户端配置。
   a. 在ClickHouse服务的**配置**页面，单击**部署客户端配置**。
   b. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
   c. 在**确认**对话框中，单击**确定**。

7. 重启Clickhouse服务。
   i. 在ClickHouse服务的**配置**页面，选择**更多操作 > 重启**。
   ii. 在**重启ClickHouse服务**对话框中，输入执行原因，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

步骤二：从操作系统中移除数据盘

1. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。
2. 执行 `df -h` 命令，查看待移除的数据盘的设备信息。

   本示例返回信息如下。

```
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        7.6G     0  7.6G   0% /dev
tmpfs           7.6G  171M  7.5G   3% /dev/shm
tmpfs           7.6G  588K  7.6G   1% /run
tmpfs           7.6G     0  7.6G   0% /sys/fs/cgroup
/dev/vda1       118G   54G   60G  48% /
tmpfs           1.6G     0  1.6G   0% /run/user/0
/dev/vdb         79G  704M   78G   1% /mnt/disk1
/dev/vdc         79G   57M   79G   1% /mnt/disk2
/dev/vdd         79G  1.6G   77G   2% /mnt/disk3
/dev/vdf         40G   49M   38G   1% /mnt/disk5
```

   其中，本示例返回信息中的参数：
   - /dev/vdf：系统中待移除数据盘的设备名，需记录下设备名，待步骤三：在ECS控制台卸载云盘中核对。
   - /mnt/disk5：系统中待移除数据盘的挂载点，下一步骤中会使用。

3. 执行以下命令，卸载磁盘。

```
umount <disk_mounted>
```

   > **注意** 命令中的 `<disk_mounted>` 为待卸载磁盘的挂载点，本示例是*/mnt/disk5*，请您根据实际情况修改。

4. （可选）如果在*/etc/fstab*中配置了磁盘的自动挂载，则修改并删除其中涉及到的配置。

步骤三：在ECS控制台卸载云盘

1. 进入实例的云盘页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**节点管理**。
   iv. 在**主机信息**区域，单击目标实例的**ECS ID**。
   v. 在ECS控制台，单击**云盘**页签。
2. 在待卸载云盘所在行，选择**更多 > 卸载**。

如果无法确定需要卸载的数据盘，可以在ECS控制台的云盘页面，将鼠标悬浮于各个云盘ID之上，即可查看各个云盘的设备名称。

例如，在系统中的设备名为*/dev/vdf*，则云盘的设备名通常为*/dev/xvdf*。

3. 在弹出的对话框中，单击**确认卸载**。

5.2.4. 数据导入

5.2.4.1. 从Spark导入数据至ClickHouse

本文为您介绍如何将Spark中的数据导入至ClickHouse集群。

前提条件

- 已创建Hadoop集群，详情请参见创建集群。
- 已创建ClickHouse集群，详情请参见创建集群。

背景信息

关于Spark的更多介绍，请参见概述。

代码示例

代码示例如下。

```
package com.company.packageName
import java.util.Properties
import java.util.concurrent.ThreadLocalRandom
import scala.annotation.tailrec
import com.google.common.collect.ImmutableMap
import org.apache.spark.internal.Logging
import org.apache.spark.sql.{SaveMode, SparkSession}
case class Test(id: Int, key1: String, value1: Boolean, key2: Long, value2: Double)
object CKDataImporter extends Logging {
  private var dbName: String = "default"
  private var tableName: String = ""
  private var ckHost: String = ""
  private var ckPort: String = "8123"
  private var user: String = "default"
  private var password: String = ""
  private var local: Boolean = false
  def main(args: Array[String]): Unit = {
    parse(args.toList)
    checkArguments()
    val jdbcUrl = s"jdbc:clickhouse://$ckHost:$ckPort/$dbName"
    logInfo(s"Use jdbc: $jdbcUrl")
    logInfo(s"Use table: $tableName")
    val spark = getSparkSession
    // generate test data
    val rdd = spark.sparkContext.parallelize(1 to 1000).map(i => {
      val rand = ThreadLocalRandom.current()
      val randString = (0 until rand.nextInt(10, 20))
        .map(_ => rand.nextLong())
        .mkString("")
      Test(i, randString, rand.nextBoolean(), rand.nextLong(), rand.nextGaussian())
    })
    val df = spark.createDataFrame(rdd)
    df.write
      .mode(SaveMode.Append)
      .jdbc(jdbcUrl, tableName, getCKJdbcProperties(user, password))
  }
  private def printUsageAndExit(exitCode: Int = 0): Unit = {
    logError("Usage: java -jar /path/to/CKDataImporter.jar [options]")
    logError("  --dbName       设置ClickHouse数据库的名称，默认为default")
    logError("  --tableName    设置ClickHouse库中表的名称")
```

```
    logError("  --tableName    设置ClickHouse库中表的名称")
    logError("  --ckHost       设置ClickHouse地址")
    logError("  --ckPort       设置ClickHouse端口，默认为8123")
    logError("  --user         设置ClickHouse所使用的用户名")
    logError("  --password     设置ClickHouse用户的密码")
    logError("  --local        设置此程序使用Spark Local模式运行")
    System.exit(exitCode)
  }
  @tailrec
  private def parse(args: List[String]): Unit = args match {
    case ("--help" | "-h") :: _ =>
      printUsageAndExit()
    case "--dbName" :: value :: tail =>
      dbName = value
      parse(tail)
    case "--tableName" :: value :: tail =>
      tableName = value
      parse(tail)
    case "--ckHost" :: value :: tail =>
      ckHost = value
      parse(tail)
    case "--ckPort" :: value :: tail =>
      ckPort = value
      parse(tail)
    case "--user" :: value :: tail =>
      user = value
      parse(tail)
    case "--password" :: value :: tail =>
      password = value
      parse(tail)
    case "--local" :: tail =>
      local = true
      parse(tail)
    case Nil =>
    case _ =>
      printUsageAndExit(1)
  }
  private def checkArguments(): Unit = {
    if ("".equals(tableName) || "".equals(ckHost)) {
      printUsageAndExit(2)
    }
  }
  private def getCKJdbcProperties(
      user: String,
      password: String,
      batchSize: String = "1000",
      socketTimeout: String = "300000",
      numPartitions: String = "8",
      rewriteBatchedStatements: String = "true"): Properties = {
    val kvMap = ImmutableMap.builder()
      .put("driver", "ru.yandex.clickhouse.ClickHouseDriver")
      .put("user", user)
      .put("password", password)
      .put("batchsize", batchSize)
      .put("socket_timeout", socketTimeout)
      .put("numPartitions", numPartitions)
      .put("rewriteBatchedStatements", rewriteBatchedStatements)
      .build()
    val properties = new Properties
    properties.putAll(kvMap)
    properties
  }
  private def getSparkSession: SparkSession = {
    val builder = SparkSession.builder()
    if (local) {
      builder.master("local[*]")
    }
    builder.appName("ClickHouse-Data-Importer")
    builder.getOrCreate()
  }
}
```

操作流程

1. 步骤一：创建ClickHouse表
2. 步骤二：编译并打包
3. 步骤三：提交作业

步骤一：创建ClickHouse表

1. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。
2. 执行如下命令，启动ClickHouse客户端。

```
clickhouse-client -h core-1-1 -m
```

> **说明** 本示例登录core-1-1节点，如果您有多个Core节点，可以登录任意一个节点。

3. 创建ClickHouse信息。
   i. 执行如下命令，创建数据库*clickhouse_database_name*。

```
CREATE DATABASE clickhouse_database_name ON CLUSTER cluster_emr;
```

   阿里云EMR会为ClickHouse集群自动生成一个名为cluster_emr的集群。数据库名您可以自定义。

   ii. 执行如下命令，创建表*clickhouse_table_name_local*。

```
CREATE TABLE clickhouse_database_name.clickhouse_table_name_local ON CLUSTER cluster_emr (
  id            UInt32,
  key1            String,
  value1        UInt8,
  key2            Int64,
  value2        Float64
) ENGINE = ReplicatedMergeTree('/clickhouse/tables/{layer}-{shard}/clickhouse_database_name/clickhouse_table_name_local', '{replica}')
ORDER BY id;
```

> **说明** 表名您可以自定义，但请确保表名是以*_local*结尾。layer、shard和replica是阿里云EMR为ClickHouse集群自动生成的宏定义，可以直接使用。

   iii. 执行如下命令，创建与表*clickhouse_table_name_local*字段定义一致的表*clickhouse_table_name_all*。

> **说明** 表名您可以自定义，但请确保表名是以*_all*结尾。

```
CREATE TABLE clickhouse_database_name.clickhouse_table_name_all ON CLUSTER cluster_emr (
  id                    UInt32,
  key1                  String,
  value1                UInt8,
  key2                  Int64,
  value2                Float64
) ENGINE = Distributed(cluster_emr, clickhouse_database_name, clickhouse_table_name_local, rand());
```

步骤二：编译并打包

1. 下载并解压CKDataImporter示例到本地。
2. 在CMD命令行中，进入到下载文件中*pom.xml*所在的目录下，执行如下命令打包文件。

```
mvn clean package
```

根据您*pom.xml*文件中artifactId的信息，下载文件中的*target*目录下会出现*CKDataImporter-1.0.0.jar*的JAR包。

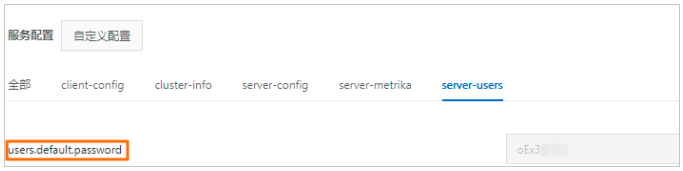
步骤三：提交作业

1. 使用SSH方式登录Hadoop集群，详情请参见登录集群。
2. 上传打包好的*CKDataImporter-1.0.0.jar*至Hadoop集群的根目录下。

> **说明** 本文示例中*CKDataImporter-1.0.0.jar*是上传至root根目录下，您也可以自定义上传路径。

3. 执行如下命令提交作业。

```
spark-submit --master yarn \
             --class com.aliyun.emr.CKDataImporter \
              CKDataImporter-1.0.0.jar \
             --dbName clickhouse_database_name \
             --tableName clickhouse_table_name_all \
             --ckHost ${clickhouse_host} \
             --password ${password};
```

参数	说明
dbName	ClickHouse集群数据库的名称，默认为default。本文示例为*clickhouse_database_name*。
tableName	ClickHouse集群数据库中表的名称。本文示例为*clickhouse_table_name_all*。
ckHost	ClickHouse集群的Master节点的内网IP地址或公网IP地址。IP地址获取方式，请参见获取主节点的IP地址。
password	ClickHouse用户的密码。 您可以在ClickHouse服务的**配置**页面，通过查看**users.default.password**参数，获取密码。 

获取主节点的IP地址

1. 进入节点管理页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**节点管理**。
2. 在**节点管理**页面，单击主实例组所在行的图标。

5.2.4.2. 从Flink导入数据至ClickHouse

本文为您介绍如何将Flink中的数据导入至ClickHouse集群。

前提条件

- 已创建Flink集群，详情请参见创建集群。
- 已创建ClickHouse集群，详情请参见创建集群。

背景信息

关于Flink的更多介绍，请参见Apache Flink。

代码示例

代码示例如下：

- 流处理

```
package com.company.packageName
import java.util.concurrent.ThreadLocalRandom
import scala.annotation.tailrec
import org.apache.flink.api.common.typeinfo.Types
import org.apache.flink.api.java.io.jdbc.JDBCAppendTableSink
```

```
import org.apache.flink.streaming.api.scala._
import org.apache.flink.table.api.scala.{StreamTableEnvironment, table2RowDataStream}
object StreamingJob {
  case class Test(id: Int, key1: String, value1: Boolean, key2: Long, value2: Double)
  private var dbName: String = "default"
  private var tableName: String = ""
  private var ckHost: String = ""
  private var ckPort: String = "8123"
  private var user: String = "default"
  private var password: String = ""
  def main(args: Array[String]) {
    parse(args.toList)
    checkArguments()
    // set up the streaming execution environment
    val env = StreamExecutionEnvironment.getExecutionEnvironment
    val tableEnv = StreamTableEnvironment.create(env)
    val insertIntoCkSql =
      s"""
        | INSERT INTO $tableName (
        |   id, key1, value1, key2, value2
        | ) VALUES (
        |   ?, ?, ?, ?, ?
        | )
        |""".stripMargin
    val jdbcUrl = s"jdbc:clickhouse://$ckHost:$ckPort/$dbName"
    println(s"jdbc url: $jdbcUrl")
    println(s"insert sql: $insertIntoCkSql")
    val sink = JDBCAppendTableSink
      .builder()
      .setDrivername("ru.yandex.clickhouse.ClickHouseDriver")
      .setDBUrl(jdbcUrl)
      .setUsername(user)
      .setPassword(password)
      .setQuery(insertIntoCkSql)
      .setBatchSize(1000)
      .setParameterTypes(Types.INT, Types.STRING, Types.BOOLEAN, Types.LONG, Types.DOUBLE)
      .build()
    val data: DataStream[Test] = env.fromCollection(1 to 1000).map(i => {
      val rand = ThreadLocalRandom.current()
      val randString = (0 until rand.nextInt(10, 20))
        .map(_ => rand.nextLong())
        .mkString("")
      Test(i, randString, rand.nextBoolean(), rand.nextLong(), rand.nextGaussian())
    })
    val table = table2RowDataStream(tableEnv.fromDataStream(data))
    sink.emitDataStream(table.javaStream)
    // execute program
    env.execute("Flink Streaming Scala API Skeleton")
  }
  private def printUsageAndExit(exitCode: Int = 0): Unit = {
    println("Usage: flink run com.company.packageName.StreamingJob /path/to/flink-clickhouse-demo-1.0.0.jar [options]")
    println("  --dbName      设置ClickHouse数据库的名称，默认为default")
    println("  --tableName   设置ClickHouse库中表的名称")
    println("  --ckHost      设置ClickHouse地址")
    println("  --ckPort      设置ClickHouse端口，默认为8123")
    println("  --user        设置ClickHouse所使用的用户名")
    println("  --password    设置ClickHouse用户的密码")
    System.exit(exitCode)
  }
  @tailrec
  private def parse(args: List[String]): Unit = args match {
    case ("--help" | "-h") :: _ =>
      printUsageAndExit()
    case "--dbName" :: value :: tail =>
      dbName = value
      parse(tail)
    case "--tableName" :: value :: tail =>
      tableName = value
      parse(tail)
    case "--ckHost" :: value :: tail =>
      ckHost = value
      parse(tail)
    case "--ckPort" :: value :: tail =>
```

```
    case "--ckPort" :: value :: tail =>
      ckPort = value
      parse(tail)
    case "--user" :: value :: tail =>
      user = value
      parse(tail)
    case "--password" :: value :: tail =>
      password = value
      parse(tail)
    case Nil =>
    case _ =>
      printUsageAndExit(1)
  }
  private def checkArguments(): Unit = {
    if ("".equals(tableName) || "".equals(ckHost)) {
      printUsageAndExit(2)
    }
  }
}
```

- 批处理

```
package com.company.packageName
import java.util.concurrent.ThreadLocalRandom
import scala.annotation.tailrec
import org.apache.flink.Utils
import org.apache.flink.api.common.typeinfo.Types
import org.apache.flink.api.java.io.jdbc.JDBCAppendTableSink
import org.apache.flink.api.scala._
import org.apache.flink.table.api.scala.{BatchTableEnvironment, table2RowDataSet}
object BatchJob {
  case class Test(id: Int, key1: String, value1: Boolean, key2: Long, value2: Double)
  private var dbName: String = "default"
  private var tableName: String = ""
  private var ckHost: String = ""
  private var ckPort: String = "8123"
  private var user: String = "default"
  private var password: String = ""
  def main(args: Array[String]) {
    parse(args.toList)
    checkArguments()
    // set up the batch execution environment
    val env = ExecutionEnvironment.getExecutionEnvironment
    val tableEnv = BatchTableEnvironment.create(env)
    val insertIntoCkSql =
      s"""
        | INSERT INTO $tableName (
        |   id, key1, value1, key2, value2
        | ) VALUES (
        |   ?, ?, ?, ?, ?
        | )
        |""".stripMargin
    val jdbcUrl = s"jdbc:clickhouse://$ckHost:$ckPort/$dbName"
    println(s"jdbc url: $jdbcUrl")
    println(s"insert sql: $insertIntoCkSql")
    val sink = JDBCAppendTableSink
      .builder()
      .setDrivername("ru.yandex.clickhouse.ClickHouseDriver")
      .setDBUrl(jdbcUrl)
      .setUsername(user)
      .setPassword(password)
      .setQuery(insertIntoCkSql)
      .setBatchSize(1000)
      .setParameterTypes(Types.INT, Types.STRING, Types.BOOLEAN, Types.LONG, Types.DOUBLE)
      .build()
    val data = env.fromCollection(1 to 1000).map(i => {
      val rand = ThreadLocalRandom.current()
      val randString = (0 until rand.nextInt(10, 20))
        .map(_ => rand.nextLong())
        .mkString("")
      Test(i, randString, rand.nextBoolean(), rand.nextLong(), rand.nextGaussian())
    })
    val table = table2RowDataSet(tableEnv.fromDataSet(data))
```

```
    sink.emitDataSet(Utils.convertScalaDatasetToJavaDataset(table))
    // execute program
    env.execute("Flink Batch Scala API Skeleton")
  }
  private def printUsageAndExit(exitCode: Int = 0): Unit = {
    println("Usage: flink run com.company.packageName.StreamingJob /path/to/flink-clickhouse-demo-1.0.0.jar [options]")
    println("  --dbName      设置ClickHouse数据库的名称，默认为default")
    println("  --tableName   设置ClickHouse库中表的名称")
    println("  --ckHost      设置ClickHouse地址")
    println("  --ckPort      设置ClickHouse端口，默认为8123")
    println("  --user        设置ClickHouse所使用的用户名")
    println("  --password    设置ClickHouse用户的密码")
    System.exit(exitCode)
  }
  @tailrec
  private def parse(args: List[String]): Unit = args match {
    case ("--help" | "-h") :: _ =>
      printUsageAndExit()
    case "--dbName" :: value :: tail =>
      dbName = value
      parse(tail)
    case "--tableName" :: value :: tail =>
      tableName = value
      parse(tail)
    case "--ckHost" :: value :: tail =>
      ckHost = value
      parse(tail)
    case "--ckPort" :: value :: tail =>
      ckPort = value
      parse(tail)
    case "--user" :: value :: tail =>
      user = value
      parse(tail)
    case "--password" :: value :: tail =>
      password = value
      parse(tail)
    case Nil =>
    case _ =>
      printUsageAndExit(1)
  }
  private def checkArguments(): Unit = {
    if ("".equals(tableName) || "".equals(ckHost)) {
      printUsageAndExit(2)
    }
  }
}
```

操作流程

1. 步骤一：创建ClickHouse表
2. 步骤二：编译并打包
3. 步骤三：提交作业

步骤一：创建ClickHouse表

1. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。
2. 执行如下命令，进入ClickHouse客户端。

   ```
   clickhouse-client -m
   ```

3. 创建ClickHouse信息。
   i. 执行如下命令，创建数据库*clickhouse_database_name*。

   ```
   CREATE DATABASE clickhouse_database_name ON CLUSTER cluster_emr;
   ```

   阿里云EMR会为ClickHouse集群自动生成一个名为cluster_emr的集群。数据库名您可以自定义。

ii. 执行如下命令，创建表*clickhouse_table_name_local*。

```
CREATE TABLE clickhouse_database_name.clickhouse_table_name_local ON CLUSTER cluster_emr (
  id            UInt32,
  key1            String,
  value1        UInt8,
  key2            Int64,
  value2        Float64
) ENGINE = ReplicatedMergeTree('/clickhouse/tables/{layer}-{shard}/clickhouse_database_name/clickhouse_table_name_local', '{replica}')
ORDER BY id;
```

> **说明** 表名您可以自定义，但请确保表名是以*_local*结尾。layer、shard和replica是阿里云EMR为ClickHouse集群自动生成的宏定义，可以直接使用。

iii. 执行如下命令，创建与表*clickhouse_table_name_local*字段定义一致的表*clickhouse_table_name_all*。

> **说明** 表名您可以自定义，但请确保表名是以*_all*结尾。

```
CREATE TABLE clickhouse_database_name.clickhouse_table_name_all ON CLUSTER cluster_emr (
  id                    UInt32,
  key1                  String,
  value1                UInt8,
  key2                  Int64,
  value2                Float64
) ENGINE = Distributed(cluster_emr, clickhouse_database_name, clickhouse_table_name_local, rand());
```

步骤二：编译并打包

1. 下载并解压flink-clickhouse-demo.tgz示例到本地。
2. 在CMD命令行中，进入到下载文件中*pom.xml*所在的目录下，执行如下命令打包文件。

```
mvn clean package
```

根据您*pom.xml*文件中artifactId的信息，下载文件中的*target*目录下会出现*flink-clickhouse-demo-1.0.0.jar*的JAR包。

步骤三：提交作业

1. 使用SSH方式登录Flink集群，详情请参见登录集群。
2. 上传打包好的*flink-clickhouse-demo-1.0.0.jar*至Flink集群的根目录下。

> **说明** 本文示例中*flink-clickhouse-demo-1.0.0.jar*是上传至root根目录下，您也可以自定义上传路径。

3. 执行如下命令提交作业。

   代码示例如下：

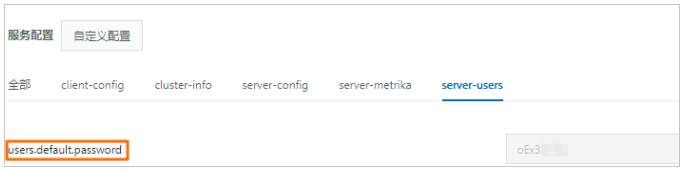
   - 流作业

```
flink run -m yarn-cluster \
          -c com.aliyun.emr.StreamingJob \
          flink-clickhouse-demo-1.0.0.jar \
          --dbName clickhouse_database_name \
          --tableName clickhouse_table_name_all \
          --ckHost ${clickhouse_host} \
          --password ${password};
```

   - 批作业

```
flink run -m yarn-cluster \
          -c com.aliyun.emr.BatchJob \
          flink-clickhouse-demo-1.0.0.jar \
          --dbName clickhouse_database_name \
          --tableName clickhouse_table_name_all \
          --ckHost ${clickhouse_host} \
          --password ${password};
```

参数	说明
dbName	ClickHouse集群数据库的名称，默认为default。本文示例为*clickhouse_database_name*。

参数	说明
tableName	ClickHouse集群数据库中表的名称。本文示例为*clickhouse_table_name_all*。
ckHost	ClickHouse集群的Master节点的内网IP地址或公网IP地址。ip地址获取方式，请参见获取主节点的IP地址。
password	ClickHouse用户的密码。 您可以在ClickHouse服务的**配置**页面，通过查看**users.default.password**参数，获取密码。 

获取主节点的IP地址

1. 进入节点管理页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**节点管理**。
2. 在**节点管理**页面，单击主实例组所在行的图标。

5.2.4.3. 从HDFS导入数据至ClickHouse

您可以通过HDFS表引擎或表函数读写数据。本文为您介绍如何将HDFS中的数据导入至ClickHouse集群。

前提条件

- 已创建Hadoop集群，详情请参见创建集群。
- 已创建ClickHouse集群，详情请参见创建集群。

注意事项

本文代码示例中HDFS URL中的9000为非HA模式下NameNode的端口，如果使用的是HA模式下的NameNode，则端口通常为8020。

使用HDFS表引擎读写数据

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name
(
  name1 [type1],
  name2 [type2],
  ...
)
Engine = HDFS(uri, format);
```

其中，涉及参数描述如下表所示。

参数	描述
db	数据库名。
table_name	表名。
name1/name2	列名。

参数	描述
`tyep1/type2`	列的类型。
`uri`	HDFS上文件的地址。 **说明** • 不可以是目录地址。 • 文件所属的目录需要存在，如果不存在，则写数据时会报错。
`format`	文件的类型。

1. 创建业务表和HDFS表
   i. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。
   ii. 执行以下命令，进入ClickHouse客户端。

```
clickhouse-client -h core-1-1 -m
```

> **说明** 本示例登录core-1-1节点，如果您有多个Core节点，可以登录任意一个节点。

   iii. 执行以下命令，创建数据库hdfs。

```
CREATE DATABASE IF NOT EXISTS hdfs ON CLUSTER cluster_emr;
```

   iv. 执行以下命令，创建表orders。

```
CREATE TABLE IF NOT EXISTS hdfs.orders ON CLUSTER cluster_emr
(
    `uid` UInt32,
    `date` DateTime,
    `skuId` UInt32,
    `order_revenue` UInt32
)
ENGINE = HDFS('hdfs://192.168.**.**:9000/orders.csv', 'CSV');
```

> **说明** 本文示例是将示例数据上传到了HDFS集群的根目录下。代码中的 `192.168.**.**` 为HDFS集群的emr-header-1节点的内网IP地址，您可以在EMR控制台**集群管理**页签中的**主机列表**页面查看。

   v. 执行以下命令，创建数据库product。

```
CREATE DATABASE IF NOT EXISTS product ON CLUSTER cluster_emr;
```

   vi. 执行以下命令，创建业务表orders。

```
CREATE TABLE IF NOT EXISTS product.orders ON CLUSTER cluster_emr
(
    `uid` UInt32,
    `date` DateTime,
    `skuId` UInt32,
    `order_revenue` UInt32
)
Engine = ReplicatedMergeTree('/cluster_emr/product/orders/{shard}', '{replica}')
PARTITION BY toYYYYMMDD(date)
ORDER BY toYYYYMMDD(date);
```

> **说明** 示例中的{shard}和{replica}是阿里云EMR为ClickHouse集群自动生成的宏定义，可以直接使用。

vii. 执行以下命令，创建业务表orders_all。

```
CREATE TABLE IF NOT EXISTS product.orders_all ON CLUSTER cluster_emr
(
    `uid` UInt32,
    `date` DateTime,
    `skuId` UInt32,
    `order_revenue` UInt32
)
Engine = Distributed(cluster_emr, product, orders, rand());
```

2. 使用HDFS表引擎导入数据

i. 下载并上传示例数据orders.csv至HDFS集群的目录下。

> **说明** 本文示例上传到了HDFS集群的根目录下。

ii. 执行以下命令，导入数据。

```
INSERT INTO product.orders_all
SELECT
  uid,
  date,
  skuId,
  order_revenue
FROM
  hdfs.orders;
```

iii. 执行以下命令，检查数据一致性。

```
SELECT
  a.*
FROM
  hdfs.orders a
LEFT ANTI JOIN
  product.orders_all
USING uid;
```

3. 使用HDFS表引擎导出数据

i. 执行以下命令，构造数据。

```
INSERT INTO product.orders_all VALUES \
  (60333391,'2021-08-04 11:26:01',49358700,89) \
  (38826285,'2021-08-03 10:47:29',25166907,27) \
  (10793515,'2021-07-31 02:10:31',95584454,68) \
  (70246093,'2021-08-01 00:00:08',82355887,97) \
  (70149691,'2021-08-02 12:35:45',68748652,1)  \
  (87307646,'2021-08-03 19:45:23',16898681,71) \
  (61694574,'2021-08-04 23:23:32',79494853,35) \
  (61337789,'2021-08-02 07:10:42',23792355,55) \
  (66879038,'2021-08-01 16:13:19',95820038,89);
```

ii. 执行以下命令，导出数据。

```
INSERT INTO hdfs.orders
SELECT
  uid,
  date,
  skuId,
  order_revenue
FROM
 product.orders_all;
```

iii. 执行以下命令，可以检查数据一致性。

```
SELECT
  a.*
FROM
  hdfs.orders
RIGHT ANTI JOIN
  product.orders_all a
USING uid;
```

语法

示例

使用HDFS表函数读写数据

```
hdfs(uri, format, structure);
```

其中，涉及参数描述如下表所示。

参数	描述
`uri`	HDFS上文件的地址。 **说明** • 不可以是目录地址。 • 文件所属的目录需要存在，如果不存在，则写数据时会报错。
`format`	文件的类型。
`structure`	表中字段的类型。例如，column1 UInt32，column2 String。

1. 创建数据库和业务表
   i. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。
   ii. 执行以下命令，进入ClickHouse客户端。

   ```
   clickhouse-client -h core-1-1 -m
   ```

   说明 本示例登录core-1-1节点，如果您有多个Core节点，可以登录任意一个节点。

   iii. 执行以下命令，创建数据库product。

   ```
   CREATE DATABASE IF NOT EXISTS product ON CLUSTER cluster_emr;
   ```

   iv. 执行以下命令，创建业务表orders。

   ```
   CREATE TABLE IF NOT EXISTS product.orders ON CLUSTER cluster_emr
   (
       `uid` UInt32,
       `date` DateTime,
       `skuId` UInt32,
       `order_revenue` UInt32
   )
   Engine = ReplicatedMergeTree('/cluster_emr/product/orders/{shard}', '{replica}')
   PARTITION BY toYYYYMMDD(date)
   ORDER BY toYYYYMMDD(date);
   ```

   v. 执行以下命令，创建业务表orders_all。

   ```
   CREATE TABLE IF NOT EXISTS product.orders_all ON CLUSTER cluster_emr
   (
       `uid` UInt32,
       `date` DateTime,
       `skuId` UInt32,
       `order_revenue` UInt32
   )
   Engine = Distributed(cluster_emr, product, orders, rand());
   ```

2. 使用HDFS表函数导入数据
   i. 下载并上传示例数据orders.csv至HDFS集群的目录下。

   说明 本文示例上传到了HDFS集群的根目录下。

ii. 执行以下命令，导入数据。

```
INSERT INTO product.orders_all
SELECT
  uid,
  date,
  skuId,
  order_revenue
FROM
  hdfs('hdfs://192.168.**.**:9000/orders.csv', 'CSV', 'uid UInt32, date DateTime, skuId UInt32, order_revenue UInt32');
```

iii. 执行以下命令，可以检查数据一致性。

```
SELECT
  a.*
FROM
  hdfs('hdfs://192.168.**.**:9000/orders.csv', 'CSV', 'uid UInt32, date DateTime, skuId UInt32, order_revenue UInt32') a
LEFT ANTI JOIN
  product.orders_all
USING uid;
```

3. 使用HDFS表函数导出数据

i. 执行以下命令，构造数据。

```
INSERT INTO product.orders_all VALUES \
  (60333391,'2021-08-04 11:26:01',49358700,89) \
  (38826285,'2021-08-03 10:47:29',25166907,27) \
  (10793515,'2021-07-31 02:10:31',95584454,68) \
  (70246093,'2021-08-01 00:00:08',82355887,97) \
  (70149691,'2021-08-02 12:35:45',68748652,1)  \
  (87307646,'2021-08-03 19:45:23',16898681,71) \
  (61694574,'2021-08-04 23:23:32',79494853,35) \
  (61337789,'2021-08-02 07:10:42',23792355,55) \
  (66879038,'2021-08-01 16:13:19',95820038,89);
```

ii. 执行以下命令，导出数据。

```
INSERT INTO FUNCTION
  hdfs('hdfs://192.168.**.**:9000/orders.csv', 'CSV', 'uid UInt32, date DateTime, skuId UInt32, order_revenue UInt32')
SELECT
  uid,
  date,
  skuId,
  order_revenue
FROM
  product.orders_all;
```

iii. 执行以下命令，可以检查数据一致性。

```
SELECT
  a.*
FROM
  hdfs('hdfs://192.168.**.**:9000/orders.csv', 'CSV', 'uid UInt32, date DateTime, skuId UInt32, order_revenue UInt32')
RIGHT ANTI JOIN
  product.orders_all a;
```

语法

示例

配置

EMR ClickHouse允许使用对HDFS进行配置：

- 全局生效的HDFS配置。

```
<hdfs>
  <dfs_default_replica>3</dfs_default_replica>
</hdfs>
```

HDFS参数的详细信息，请参见官网文档HDFS Configuration Reference。

> **说明** 查询参数时将下划线（_）替换为半角句号（.）即可。例如，您要查询EMR中的参数 `dfs_default_replica`，则可以在官网文档中搜索 `dfs.default.replica`。

- 仅对${user}用户生效的HDFS配置，用户配置与全局配置相同的键不同值时，会覆盖全局配置。

```
<hdfs_${user}>
  <dfs_default_replica>3</dfs_default_replica>
</hdfs_${user}>
```

5.2.4.4. 从OSS导入数据至ClickHouse

在EMR ClickHouse集群，您可以在通过OSS表引擎读写数据，或者通过OSS表函数读数据。本文为您介绍如何将OSS中的数据导入至ClickHouse集群。

前提条件

- 已在OSS上创建存储空间，详情请参见创建存储空间。
- 已创建ClickHouse集群，详情请参见创建集群。

使用OSS表引擎读写数据

创建OSS表的语法如下所示。

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
  name1 [type1] [NULL|NOT NULL] [DEFAULT|MATERIALIZED|ALIAS expr1] [compression_codec] [TTL expr1],
  name2 [type2] [NULL|NOT NULL] [DEFAULT|MATERIALIZED|ALIAS expr2] [compression_codec] [TTL expr2],
  ...
)
ENGINE = OSS(path, [access_key_id, access_key_secret,] format, [compression]);
```

其中，涉及参数描述如下表所示。

参数	描述
db	数据库名。
table_name	表名。
name1/name2	列名。
tyep1/type2	列的类型。
path	OSS路径。 ClickHouse集群访问OSS使用地址详情，请参见ECS实例通过OSS内网地址访问OSS资源。 path支持virtual hosted style和path style两种形式。推荐您使用virtual hosted style。 path支持使用以下通配符： • `*` 表示除了 '/' 以外任意个字符，包括空字符串。 • `?` 表示单个字符。 • `{str1,str2,...,strn}` 表示 `str1/str2/.../strn` 中任意一个字符串。 • `{N..M}` 表示从N到M的任意数字。N和M可以包含前导0，例如 {001..099}。 **注意** 根据通配符来决定哪些文件会被使用在 SELECT 时而非创建表时，因此如果是为了写入数据至OSS，不应该使用通配符。
access_key_id	阿里云账号的AccessKey ID。
access_key_secret	阿里云账号的AccessKey Secret。
format	path所指向的对象（文件）的格式。例如，CSV和XML等类型，详细信息请参见Formats for Input and Output Data。

参数	描述
compression	压缩类型。 该参数为可选参数，默认会根据文件扩展选择合适的压缩类型。 根据您创建的集群版本，设置压缩类型： • EMR-3.x系列版本：支持 `none` 、 `gzip/gz` 、 `brotli/b` 、 `deflate` 或 `auto` 。 • EMR-5.x系列版本：支持 `none` 、 `gzip/gz` 、 `brotli/b` 、 `lzma(xz)` 、 `auto` 或 `zstd/zst` 。

1. 下载并上传示例数据orders.csv至OSS。

   上传文件至OSS的详细操作，请参见上传文件。
2. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。
3. 执行以下命令，进入ClickHouse客户端。

   ```
   clickhouse-client -h core-1-1 -m
   ```

   > **说明** 本示例登录core-1-1节点，如果您有多个Core节点，可以登录任意一个节点。
4. 执行以下命令，创建数据库test。

   ```
   CREATE DATABASE IF NOT EXISTS test ON CLUSTER cluster_emr;
   ```
5. 执行以下命令，创建业务表。

   i. 创建复制表。

   ```
   CREATE TABLE test.orders ON CLUSTER cluster_emr
   (
     uid UInt32,
     date DateTime,
     skuId UInt32,
     order_revenue UInt32
   ) ENGINE = ReplicatedMergeTree('/clickhouse/cluster_emr/test/orders/{shard}', '{replica}')
   PARTITION BY toYYYYMMDD(date)
   ORDER BY toYYYYMMDD(date);
   ```

   ii. 创建分布式表。

   ```
   CREATE TABLE test.orders_all ON CLUSTER cluster_emr
   (
     uid UInt32,
     date DateTime,
     skuId UInt32,
     order_revenue UInt32
   ) ENGINE = Distributed(cluster_emr, test, orders, rand());
   ```
6. 执行以下命令，创建表orders_oss。

   ```
   CREATE TABLE test.orders_oss
   (
     uid UInt32,
     date DateTime,
     skuId UInt32,
     order_revenue UInt32
   ) ENGINE = OSS('http://test.oss-cn-beijing.aliyuncs.com/orders.csv', '<access_key_id>', '<access_key_secret>', 'CSV');
   ```

   > **说明** 示例中的数据目录*http://test.oss-cn-beijing.aliyuncs.com/orders.csv*，表示cn-beijing地域下名称为test的Bucket中的*orders.csv*文件。
7. 将OSS上的数据写入业务表orders_all。

```
INSERT INTO test.orders_all
SELECT
  uid,
  date,
  skuId,
  order_revenue
FROM
  test.orders_oss;
```

您可以通过以下命令检查数据一致性：

- 查看表orders_all的数据。

  ```
  SELECT count(1) FROM test.orders_all;
  ```

- 查看表orders_oss的数据。

  ```
  SELECT count(1) FROM test.orders_oss;
  ```

1. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。
2. 执行以下命令，进入ClickHouse客户端。

   ```
   clickhouse-client -h core-1-1 -m
   ```

   > **说明** 本示例登录core-1-1节点，如果您有多个Core节点，可以登录任意一个节点。

3. 执行以下命令，创建要写入数据的OSS表。

   ```
   CREATE TABLE test.orders_oss
   (
     uid UInt32,
     date DateTime,
     skuId UInt32,
     order_revenue UInt32
   ) ENGINE = OSS('http://test.oss-cn-beijing.aliyuncs.com/orders.csv', '<access_key_id>', '<access_key_secret>', 'CSV');
   ```

4. 执行以下命令，向表中写入数据。

   ```
   --假设业务表为test.orders_all
   INSERT INTO test.orders_oss
   SELECT
     uid,
     date,
     skuId,
     order_revenue
   FROM
     test.orders_all;
   ```

5. 在 OSS管理控制台上查看数据。

语法

示例：使用OSS表将数据导入至ClickHouse集群

示例：将ClickHouse集群数据导出至OSS

使用OSS表函数读数据

创建OSS表的语法如下所示。

```
oss(path, [access_key_id, access_key_secret,] format, structure, [compression])
```

其中，涉及参数描述如下表所示。

参数	描述

参数	描述
path	OSS路径。 ClickHouse集群访问OSS使用地址详情，请参见ECS实例通过OSS内网地址访问OSS资源。 path支持virtual hosted style和path style两种形式。推荐您使用virtual hosted style。 path支持使用以下通配符： • * 表示除了 '/' 以外任意个字符，包括空字符串。 • ? 表示单个字符。 • {str1,str2,...,strn} 表示 str1/str2/.../strn 中任意一个字符串。 • {N..M} 表示从N到M的任意数字。N和M可以包含前导0，例如 {001..099}。 注意 根据通配符来决定哪些文件会被使用在 SELECT 时而非创建表时，因此如果是为了写入数据至OSS，不应该使用通配符。
access_key_id	阿里云账号的AccessKey ID。
access_key_secret	阿里云账号的AccessKey Secret。
format	path所指向的对象（文件）的格式。例如，CSV和XML等类型，详细信息请参见Formats for Input and Output Data。
structure	表中字段的类型。例如，column1 UInt32、column2 String。
compression	压缩类型。 该参数为可选参数，默认会根据文件扩展选择合适的压缩类型。 根据您创建的集群版本，设置压缩类型： • EMR-3.x系列版本：支持 none 、 gzip/gz 、 brotli/b 、 deflate 或 auto 。 • EMR-5.x系列版本：支持 none 、 gzip/gz 、 brotli/b 、 lzma(xz) 、 auto 或 zstd/zst 。

1. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。
2. 执行以下命令，进入ClickHouse客户端。

```
clickhouse-client -h core-1-1 -m
```

> 说明 本示例登录core-1-1节点，如果您有多个Core节点，可以登录任意一个节点。

3. 执行以下命令，查询OSS表数据。

```
SELECT * FROM oss(
  'http://test.oss-cn-beijing.aliyuncs.com/orders.csv',
  '<your-access-key>',
  '<your-access-key-secret>',
  'CSV',
  'uid UInt32, date DateTime, skuId UInt32, order_revenue UInt32');
```

返回信息如下。

```
┌──uid─┬────────date─┬─skuId─┬─order_revenue─┐
│ 60333391 │ 2021-08-04 11:26:01 │ 49358700 │            89 │
│ 38826285 │ 2021-08-03 10:47:29 │ 25166907 │            27 │
│ 10793515 │ 2021-07-31 02:10:31 │ 95584454 │            68 │
│ 70246093 │ 2021-08-01 00:00:08 │ 82355887 │            97 │
│ 70149691 │ 2021-08-02 12:35:45 │ 68748652 │             1 │
│ 87307646 │ 2021-08-03 19:45:23 │ 16898681 │            71 │
│ 61694574 │ 2021-08-04 23:23:32 │ 79494853 │            35 │
│ 61337789 │ 2021-08-02 07:10:42 │ 23792355 │            55 │
│ 66879038 │ 2021-08-01 16:13:19 │ 95820038 │            89 │
└──────┴─────────────┴───────┴───────────────┘
```

4. 使用OSS表函数将数据导入至ClickHouse集群。

```
INSERT INTO test.orders_all
SELECT
  uid,
  date,
  skuId,
  order_revenue
FROM
  oss('http://test.oss-cn-beijing.aliyuncs.com/orders.csv',
      '<your-access-key>',
      '<your-access-secret>',
      'CSV',
      'uid UInt32, date DateTime, skuId UInt32, order_revenue UInt32');
```

语法

示例

OSS相关配置

1. 支持的profile

   如果使用MultipartUpload上传文件到OSS，则可以设置 `oss_min_upload_part_size` 参数以指定每个part最小的大小，默认值为512 MB，必须使用UInt64范围内的整数。

2. 设置方法

   - 在一次SQL中，代码设置如下。

     ```
     INSERT INTO OSS_TABLE
     SELECT
       ...
     FROM
       ...
     SETTINGS
       oss_min_upload_part_size=1073741824;
     ```

   - 在一次Session中，代码设置如下。

     ```
     SET oss_min_upload_part_size=1073741824;
     INSERT INTO OSS_TABLE
     SELECT
       ...
     FROM
       ...
     ;
     ```

   - 针对某一个表，代码设置如下。

     ```
     CREATE TABLE OSS_TABLE
     (
       ...
     ) ENGINE = OSS(...)
     SETTINGS
       oss_min_upload_part_size=1073741824;
     ```

   - 针对某一个用户，设置如下。

     在EMR控制台ClickHouse服务的**配置**页面，单击**server-users**页签，新增参数为**users.<YourUserName>.oss_min_upload_part_size**，参数值为1073741824的配置项。

EMR中的ClickHouse支持使用如下参数配置OSS，代码示例如下。

```
<oss>
    <endpoint-name>
        <endpoint>https://oss-cn-beijing.aliyuncs.com/bucket</endpoint>
        <access_key_id>ACCESS_KEY_ID</access_key_id>
        <secret_access_key>ACCESS_KEY_SECRET</secret_access_key>
    </endpoint-name>
</oss>
```

其中，相关参数描述如下。

参数	描述
endpoint-name	Endpoint的名称。

参数	描述
endpoint	OSS的访问域名，详情请参见OSS访问域名使用规则。
access_key_id	阿里云账号的AccessKey ID。
secret_access_key	阿里云账号的AccessKey Secret。

您也可以在EMR控制台ClickHouse服务的**配置**页面，单击**server-config**页签，通过以下两个方式新增自定义配置。

方式	操作
方法一	新增参数oss.<endpoint-name>.endpoint、oss.<endpoint-name>.access_key_id和oss.<endpoint-name>.secret_access_key及其对应的参数值。 **说明** 参数中的 `<endpoint-name>` 需要替换为Endpoint的名称。

方法二	新增参数为**oss**，参数值如下的配置项。
	`<endpoint-name>`
	`  <endpoint>https://oss-cn-beijing.aliyuncs.com/bucket</endpoint>`
	`  <access_key_id>ACCESS_KEY_ID</access_key_id>`
	`  <secret_access_key>ACCESS_KEY_SECRET</secret_access_key>`
	`</endpoint-name>`
	说明 参数值请替换为您实际的值。

如果您已进行如上配置，则在创建OSS表或使用OSS表函数时，可以使用如下命令。

- OSS表

```
CREATE TABLE OSS_TABLE
(
  column1 UInt32,
  column2 String
  ...
)
ENGINE = OSS(path, format, [compression]);
```

- OSS表函数

```
oss(path, format, structure, [compression]);
```

profile

configuration

5.2.4.5. 从RDS导入数据至ClickHouse

您可以通过RDS MySQL表引擎或表函数导入数据至ClickHouse集群。本文为您介绍如何将RDS中的数据导入至ClickHouse集群。

前提条件

- 已购买RDS，详情请参见创建RDS MySQL实例。
- 已创建ClickHouse集群，详情请参见创建集群。

使用RDS MySQL表引擎导入数据

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1] [TTL expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2] [TTL expr2],
    ...
) ENGINE = MySQL('host:port', 'database', 'table', 'user', 'password'[, replace_query, 'on_duplicate_clause']);
```

其中，涉及参数描述如下表所示。

参数	描述
db	数据库名。
table_name	表名。
cluster	集群标识。
name1/name2	列名。
tyep1/type2	列的类型。
host:port	RDS MySQL的地址，可以在RDS MySQL管理控制台中数据库连接中进行查看。
database	RDS MySQL中的数据库名。
table	RDS MySQL中的表名。
user	用户名，该用户具有访问上述RDS MySQL中库中的表的权限。
password	user 对应的密码。
replace_query	是否将INSERT INTO查询转换为REPLACE INTO的标志。设置为1，表示替换查询。
on_duplicate_clause	会被添加到INSERT语句中。例如， INSERT INTO t (c1,c2) VALUES ('a', 2) ON DUPLICATE KEY UPDATE c2 = c2 + 1 ，此时需要指定 on_duplicate_clause 为 UPDATE c2 = c2 + 1 。

1. 在RDS MySQL实例中，创建原始数据表并导入原始数据。
   i. 连接MySQL实例，详情请参见通过客户端、命令行连接RDS MySQL。
   ii. 执行以下命令，创建原始数据表。

```
CREATE TABLE `origin`.`orders` (
  `uid` int(10) unsigned DEFAULT NULL,
  `date` datetime DEFAULT NULL,
  `skuId` int(10) unsigned DEFAULT NULL,
  `order_revenue` int(10) unsigned DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

   iii. 执行以下命令，导入原始数据。

```
INSERT INTO `origin`.`orders` VALUES(60333391, '2021-08-04 11:26:01', 49358700, 89),
       (38826285, '2021-08-03 10:47:29', 25166907, 27),
       (10793515, '2021-07-31 02:10:31', 95584454, 68),
       (70246093, '2021-08-01 00:00:08', 82355887, 97),
       (70149691, '2021-08-02 12:35:45', 68748652, 1),
       (87307646, '2021-08-03 19:45:23', 16898681, 71),
       (61694574, '2021-08-04 23:23:32', 79494853, 35),
       (61337789, '2021-08-02 07:10:42', 23792355, 55),
       (66879038, '2021-08-01 16:13:19', 95820038, 89);
```

2. 在ClickHouse集群中，执行以下操作。
   i. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。
   ii. 执行以下命令，进入ClickHouse客户端。

```
clickhouse-client -h core-1-1 -m
```

   > **说明** 本示例登录core-1-1节点，如果您有多个Core节点，可以登录任意一个节点。

   iii. 执行以下命令，创建数据库mysql。

```
CREATE DATABASE IF NOT EXISTS mysql;
```

iv. 执行以下命令，创建表orders。

```
CREATE TABLE mysql.orders
(
    `uid` UInt32,
    `date` DateTime,
    `skuId` UInt32,
    `order_revenue` UInt32
)
ENGINE = MySQL('host:port', 'origin', 'orders', 'user', 'password');
```

v. 执行以下命令，创建数据库product。

```
CREATE DATABASE IF NOT EXISTS product ON CLUSTER cluster_emr;
```

vi. 执行以下命令，创建业务表orders。

```
CREATE TABLE IF NOT EXISTS product.orders ON CLUSTER cluster_emr
(
    `uid` UInt32,
    `date` DateTime,
    `skuId` UInt32,
    `order_revenue` UInt32
)
Engine = ReplicatedMergeTree('/cluster_emr/product/orders/{shard}', '{replica}')
PARTITION BY toYYYYMMDD(date)
ORDER BY toYYYYMMDD(date);
```

vii. 执行以下命令，创建业务表orders_all。

```
CREATE TABLE IF NOT EXISTS product.orders_all ON CLUSTER cluster_emr
(
    `uid` UInt32,
    `date` DateTime,
    `skuId` UInt32,
    `order_revenue` UInt32
)
Engine = Distributed(cluster_emr, product, orders, rand());
```

viii. 执行以下命令，导入数据。

```
INSERT INTO product.orders_all
SELECT
  uid,
  date,
  skuId,
  order_revenue
FROM
  mysql.orders;
```

ix. 执行以下命令，查询数据。

```
SELECT a.*
FROM mysql.orders AS a
ANTI LEFT JOIN product.orders_all USING (uid);
```

> **说明** 查询数据为空时正常。

语法

示例

使用RDS MySQL表函数导入数据

```
mysql('host:port', 'database', 'table', 'user', 'password'[, replace_query, 'on_duplicate_clause'])
```

其中，涉及参数描述如下表所示。

参数	描述
host:port	RDS MySQL的地址，您可以在RDS MySQL管理控制台中的数据库连接中查看。
database	RDS MySQL中的数据库名。

参数	描述
table	RDS MySQL中的表名。
user	用户名，该用户具有访问上述RDS MySQL中库中的表的权限。
password	user 对应的密码。
replace_query	是否将INSERT INTO查询转换为REPLACE INTO的标志。设置为1，表示替换查询。
on_duplicate_clause	会被添加到INSERT语句中。例如， INSERT INTO t (c1,c2) VALUES ('a', 2) ON DUPLICATE KEY UPDATE c2 = c2 + 1 ，此时需要指定 on_duplicate_clause 为 UPDATE c2 = c2 + 1 。

1. 在RDS MySQL实例中，创建表并插入数据。
   i. 连接MySQL实例，详情请参见通过客户端、命令行连接RDS MySQL。
   ii. 执行以下命令，创建表orders。

```
CREATE TABLE `origin`.`orders` (
  `uid` int(10) unsigned DEFAULT NULL,
  `date` datetime DEFAULT NULL,
  `skuId` int(10) unsigned DEFAULT NULL,
  `order_revenue` int(10) unsigned DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

   iii. 执行以下命令，插入数据。

```
INSERT INTO `origin`.`orders` VALUES(60333391, '2021-08-04 11:26:01', 49358700, 89),
       (38826285, '2021-08-03 10:47:29', 25166907, 27),
       (10793515, '2021-07-31 02:10:31', 95584454, 68),
       (70246093, '2021-08-01 00:00:08', 82355887, 97),
       (70149691, '2021-08-02 12:35:45', 68748652, 1),
       (87307646, '2021-08-03 19:45:23', 16898681, 71),
       (61694574, '2021-08-04 23:23:32', 79494853, 35),
       (61337789, '2021-08-02 07:10:42', 23792355, 55),
       (66879038, '2021-08-01 16:13:19', 95820038, 89);
```

2. 在ClickHouse集群中，执行以下操作。
   i. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。
   ii. 执行以下命令，进入ClickHouse客户端。

```
clickhouse-client -h core-1-1 -m
```

> **说明** 本示例登录core-1-1节点，如果您有多个Core节点，可以登录任意一个节点。

   iii. 执行以下命令，创建数据库product。

```
CREATE DATABASE IF NOT EXISTS product ON CLUSTER cluster_emr;
```

   iv. 执行以下命令，创建表orders。

```
CREATE TABLE IF NOT EXISTS product.orders ON CLUSTER cluster_emr
(
    `uid` UInt32,
    `date` DateTime,
    `skuId` UInt32,
    `order_revenue` UInt32
)
Engine = ReplicatedMergeTree('/cluster_emr/product/orders/{shard}', '{replica}')
PARTITION BY toYYYYMMDD(date)
ORDER BY toYYYYMMDD(date);
```

v. 执行以下命令，创建表orders_all。

```
CREATE TABLE IF NOT EXISTS product.orders_all ON CLUSTER cluster_emr
(
    `uid` UInt32,
    `date` DateTime,
    `skuId` UInt32,
    `order_revenue` UInt32
)
Engine = Distributed(cluster_emr, product, orders, rand());
```

vi. 执行以下命令，导入数据。

```
INSERT INTO product.orders_all
SELECT
  uid,
  date,
  skuId,
  order_revenue
FROM
  mysql('host:port', 'origin', 'orders', 'user', 'password');
```

vii. 执行以下命令，查询数据。

```
SELECT a.*
FROM
  mysql('host:port', 'origin', 'orders', 'user', 'password') AS a
ANTI LEFT JOIN product.orders_all USING (uid);
```

> **说明** 查询数据为空时正常。

如果您需要导出数据，则将业务表数据写入MySQL表函数即可。写入命令如下。

```
INSERT INTO FUNCTION
  mysql('host:port', 'origin', 'orders', 'user', 'password')
FROM
  product.orders_all;
```

语法

示例

5.2.4.6. 从Kafka导入数据至ClickHouse

您可以通过Kafka表引擎导入数据至ClickHouse集群。本文为您介绍如何将Kafka中的数据导入至ClickHouse集群。

前提条件

- 已创建DataFlow集群，且选择了Kafka服务，详情请参见创建集群。
- 已创建ClickHouse集群，详情请参见创建集群。

使用限制

DataFlow集群和ClickHouse集群需要在同一VPC下。

语法

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2],
    ...
) ENGINE = Kafka()
SETTINGS
    kafka_broker_list = 'host1:port1,host2:port2',
    kafka_topic_list = 'topic1,topic2,...',
    kafka_group_name = 'group_name',
    kafka_format = 'data_format';
```

其中，涉及参数描述如下表所示。

参数	描述
db	数据库名。
table_name	表名。
cluster	集群标识。
name1/name2	列名。
tyep1/type2	列的类型。
kafka_broker_list	Kafka Broker的地址及端口。 DataFlow集群所有节点的内网IP地址及端口，您可以在EMR控制台**集群管理**页签中的**主机列表**页面查看。
kafka_topic_list	订阅的Topic名称。
kafka_group_name	Kafka consumer的分组名称。
kafka_format	数据的类型。例如，CSV和JSONEachRow等，详细信息请参见Formats for Input and Output Data。

示例

1. 在ClickHouse集群中执行以下操作。
   i. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。
   ii. 执行如下命令，进入ClickHouse客户端。

   ```
   clickhouse-client -h core-1-1 -m
   ```

   > **说明** 本示例登录core-1-1节点，如果您有多个Core节点，可以登录任意一个节点。

   iii. 执行如下命令，创建数据库kafka。

   ```
   CREATE DATABASE IF NOT EXISTS kafka ON CLUSTER cluster_emr;
   ```

   > **说明** 数据库名您可以自定义。本文示例中的 `cluster_emr` 是集群默认的标识，如果您修改过，请填写正确的集群标识，您也可以在EMR控制台ClickHouse服务的**配置**页面，在搜索区域搜索**clickhouse_remote_servers**参数查看。

iv. 执行如下命令，创建Kafka表。

```
CREATE TABLE IF NOT EXISTS kafka.consumer ON CLUSTER cluster_emr
(
    `uid` UInt32,
    `date` DateTime,
    `skuId` UInt32,
    `order_revenue` UInt32
)
ENGINE = Kafka()
SETTINGS
  kafka_broker_list = '192.168.**.**:9092,192.168.**.**:9092,192.168.**.**:9092',
  kafka_topic_list = 'clickhouse_test',
  kafka_group_name = 'clickhouse_test',
  kafka_format = 'CSV';
```

`kafka_broker_list` 为DataFlow集群所有节点的内网IP地址及端口，您可以在EMR控制台的**节点管理**页面查看。其余参数含义请参见语法。

v. 执行如下命令，创建数据库product。

```
CREATE DATABASE IF NOT EXISTS product ON CLUSTER cluster_emr;
```

vi. 执行以下命令，创建本地表。

```
CREATE TABLE IF NOT EXISTS product.orders ON CLUSTER cluster_emr
(
    `uid` UInt32,
    `date` DateTime,
    `skuId` UInt32,
    `order_revenue` UInt32
)
Engine = ReplicatedMergeTree('/cluster_emr/product/orders/{shard}', '{replica}')
PARTITION BY toYYYYMMDD(date)
ORDER BY toYYYYMMDD(date);
```

vii. 执行以下命令，创建分布式表。

```
CREATE TABLE IF NOT EXISTS product.orders_all ON CLUSTER cluster_emr
(
    `uid` UInt32,
    `date` DateTime,
    `skuId` UInt32,
    `order_revenue` UInt32
)
Engine = Distributed(cluster_emr, product, orders, rand());
```

viii. 执行以下命令，创建MATERIALIZED VIEW自动导数据。

```
CREATE MATERIALIZED VIEW IF NOT EXISTS product.kafka_load ON CLUSTER cluster_emr TO product.orders AS
SELECT *
FROM kafka.consumer;
```

2. 在DataFlow集群中执行以下操作。
   i. 使用SSH方式登录DataFlow集群，详情请参见登录集群。
   ii. 在DataFlow集群的命令行窗口，执行如下命令运行Kafka的生产者。

```
/usr/lib/kafka-current/bin/kafka-console-producer.sh --broker-list 192.168.**.**:9092,192.168.**.**:9092,192.168.**.**:9092 --topic clickhouse_test
```

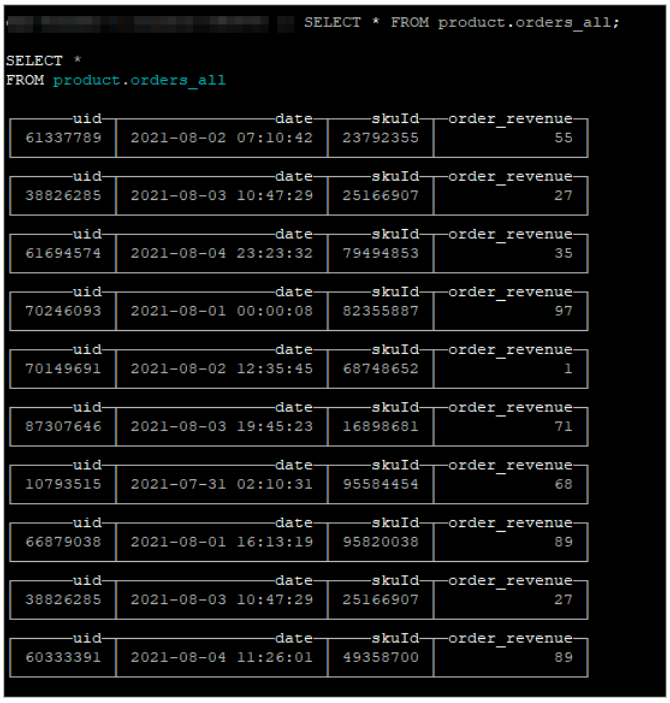
   iii. 执行以下命令，输入测试数据。

```
38826285,2021-08-03 10:47:29,25166907,27
10793515,2021-07-31 02:10:31,95584454,68
70246093,2021-08-01 00:00:08,82355887,97
70149691,2021-08-02 12:35:45,68748652,1
87307646,2021-08-03 19:45:23,16898681,71
61694574,2021-08-04 23:23:32,79494853,35
61337789,2021-08-02 07:10:42,23792355,55
66879038,2021-08-01 16:13:19,95820038,89
```

3. 在ClickHouse命令窗口中，执行以下命令，可以查看从Kafka中导入至ClickHouse集群的数据。

   您可以校验查询到的数据与源数据是否一致。

```
SELECT * FROM product.orders_all;
```



5.2.5. 冷热分离

5.2.5.1. 使用HDFS进行数据冷热分离

本文为您介绍在阿里云E-MapReduce的ClickHouse集群上，如何通过HDFS进行数据的冷热分离。通过本文操作，您既可以在保证集群读写性能的基础上，自动维护集群上的冷热数据，又可以充分利用计算和存储资源，以降低成本。

前提条件

- 已在EMR控制台上创建EMR-5.5.0及以上版本的ClickHouse集群，详情请参见创建集群。
- 在同一VPC下具有一个HDFS服务（例如，EMR Hadoop集群）。
- 拥有HDFS服务的读写权限。

使用限制

本文操作仅适用于EMR-5.5.0及以上版本的ClickHouse集群。

操作流程

1. 步骤一：在EMR控制台添加磁盘
2. 步骤二：验证配置
3. 步骤三：进行冷热分离

步骤一：在EMR控制台添加磁盘

1. 进入ClickHouse配置页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 在**集群服务**页面，单击ClickHouse服务区域的**配置**。
2. 在**服务配置**区域，单击**server-metrika**页签。
3. 修改**storage_configuration**的参数值。
   i. 在**disks**中添加一个HDFS类型的磁盘。
   详细信息如下。

```
<disk_hdfs>
    <type>hdfs</type>
    <endpoint>hdfs://${your-hdfs-url}</endpoint>
    <min_bytes_for_seek>1048576</min_bytes_for_seek>
    <thread_pool_size>16</thread_pool_size>
    <objects_chunk_size_to_delete>1000</objects_chunk_size_to_delete>
</disk_hdfs>
```

相关参数描述如下。

参数	是否必填	描述
disk_hdfs	是	磁盘的名称，您可以自定义。
type	是	磁盘的类型，固定值为hdfs。
endpoint	是	HDFS服务的目录地址。 注意 HDFS的地址通常是NameNode的地址，如果NameNode是HA模式，其端口通常为8020，否则为9000。
min_bytes_for_seek	否	最小使用Seek的Byte数量，低于该值时会用Skip代Seek。默认值为1048576。
thread_pool_size	否	用于Disk用于执行restore时所使用的线程池的大小。默认值为16。
objects_chunk_size_to_delete	否	一次最多可以删除HDFS文件的数量。默认为1000。

ii. 在**policies**中添加一个新的策略。

策略内容如下。

```
<hdfs_ttl>
    <volumes>
      <local>
        <!-- 包含默认存储策略下所有的磁盘 -->
        <disk>disk1</disk>
        <disk>disk2</disk>
        <disk>disk3</disk>
        <disk>disk4</disk>
      </local>
      <remote>
        <disk>disk_hdfs</disk>
      </remote>
    </volumes>
    <move_factor>0.2</move_factor>
</hdfs_ttl>
```

> **说明** 该部分内容也可以直接添加在default策略中。

4. 保存配置。
   i. 在ClickHouse服务的**配置**页面，单击**保存**。
   ii. 在**确认修改**对话框中，输入执行原因，打开**自动更新配置**开关，单击**确定**。
5. 部署客户端配置。
   i. 在ClickHouse服务的**配置**页面，单击**部署客户端配置**。
   ii. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

步骤二：验证配置

1. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。
2. 执行如下命令，启动ClickHouse客户端。

```
clickhouse-client -h core-1-1 -m
```

> **说明** 本示例登录core-1-1节点，如果您有多个Core节点，可以登录任意一个节点。

3. 执行如下命令，查看磁盘信息。

```
select * from system.disks;
```

返回信息如下所示。

```
┌─name─────┬─path──────────────────────────────────┬──────────free_space─┬─────────total_space─┬─keep_free_space─┬─typ
e─┐
│ default  │ /var/lib/clickhouse/                  │          83868921856 │          84014424064 │               0 │ loc
al │
│ disk1    │ /mnt/disk1/clickhouse/                │          83858436096 │          84003938304 │        10485760 │ loc
al │
│ disk2    │ /mnt/disk2/clickhouse/                │          83928215552 │          84003938304 │        10485760 │ loc
al │
│ disk3    │ /mnt/disk3/clickhouse/                │          83928301568 │          84003938304 │        10485760 │ loc
al │
│ disk4    │ /mnt/disk4/clickhouse/                │          83928301568 │          84003938304 │        10485760 │ loc
al │
│ disk_hdfs│ /var/lib/clickhouse/disks/disk_hdfs/ │ 18446744073709551615 │ 18446744073709551615 │               0 │ hdf
s  │
└──────────┴───────────────────────────────────────┴──────────────────────┴──────────────────────┴─────────────────┴────
────┘
```

4. 执行如下命令，查看磁盘存储策略。

```
select * from system.storage_policies;
```

返回信息如下所示。

```
┌─policy_name─┬─volume_name─┬─volume_priority─┬─disks─────────────────────────────┬─volume_type─┬─max_data_part_size─
┬─move_factor─┬─prefer_not_to_merge─┐
│ default     │ single      │               1 │ ['disk1','disk2','disk3','disk4'] │JBOD         │
0 │           0 │                   0 │
│ hdfs_ttl    │ local       │               1 │ ['disk1','disk2','disk3','disk4'] │JBOD         │
0 │         0.2 │                   0 │
│ hdfs_ttl    │ remote      │               2 │ ['disk_hdfs']                     │JBOD         │
0 │         0.2 │                   0 │
└─────────────┴─────────────┴─────────────────┴───────────────────────────────────┴─────────────┴────────────────────
──────────┴─────────────────────┘
```

当回显信息如上文所示时，表示磁盘扩容操作完成。

步骤三：进行冷热分离

对已有的表进行改造

1. 查看当前的存储策略。
   i. 在ClickHouse客户端执行如下命令，查看磁盘信息。

   ```
   SELECT
     storage_policy
   FROM system.tables
   WHERE database='<database_name>' AND name='<table_name>';
   ```

   命令中的 `<database_name>` 为数据库名，`<table_name>` 为表名。

   如果返回信息如下所示，则需要参见下一步骤添加一个volume。

   ```
   <default>
     <volumes>
       <single>
         <disk>disk1</disk>
         <disk>disk2</disk>
         <disk>disk3</disk>
         <disk>disk4</disk>
       </single>
     </volumes>
   </default>
   ```

2. 扩展当前的存储策略。

   在EMR控制台ClickHouse服务的配置页签，增加volume内容，详细信息如下。

   ```
   <default>
     <volumes>
       <single>
         <disk>disk1</disk>
         <disk>disk2</disk>
         <disk>disk3</disk>
         <disk>disk4</disk>
       </single>
       <!-- 以下是新增的volume remote -->
       <remote>
         <disk>disk_hdfs</disk>
       </remote>
     </volumes>
     <!-- 多个volume时需要指定move_factor -->
     <move_factor>0.2</move_factor>
   </default>
   ```

3. 执行以下命令，修改TTL。

   ```
   ALTER TABLE <yourDataName>.<yourTableName>
     MODIFY TTL toStartOfMinute(addMinutes(t, 5)) TO VOLUME 'remote';
   ```

4. 执行以下命令，查看各个part的分布。

   ```
   select partition,name,path from system.parts where database='<yourDataName>' and table='<yourTableName>' and active=1
   ```

   返回信息如下。

```
┌─partition───────────┬─name────────────────┬─path──────────────────────────────────────────────────────────────────
───────────────────────────┐
│ 2022-01-12 11:30:00 │ 1641958200_1_96_3   │ /var/lib/clickhouse/disks/disk_hdfs/store/156/156008ff-41bf-460c-8848-e3
4fad88c25d/1641958200_1_96_3/ │
│ 2022-01-12 11:35:00 │ 1641958500_97_124_2 │ /mnt/disk3/clickhouse/store/156/156008ff-41bf-460c-8848-e34fad88c25d/164
1958500_97_124_2/             │
│ 2022-01-12 11:35:00 │ 1641958500_125_152_2 │ /mnt/disk4/clickhouse/store/156/156008ff-41bf-460c-8848-e34fad88c25d/164
1958500_125_152_2/            │
│ 2022-01-12 11:35:00 │ 1641958500_153_180_2 │ /mnt/disk1/clickhouse/store/156/156008ff-41bf-460c-8848-e34fad88c25d/164
1958500_153_180_2/            │
│ 2022-01-12 11:35:00 │ 1641958500_181_186_1 │ /mnt/disk4/clickhouse/store/156/156008ff-41bf-460c-8848-e34fad88c25d/164
1958500_181_186_1/            │
│ 2022-01-12 11:35:00 │ 1641958500_187_192_1 │ /mnt/disk3/clickhouse/store/156/156008ff-41bf-460c-8848-e34fad88c25d/164
1958500_187_192_1/            │
└─────────────────────┴─────────────────────┴───────────────────────────────────────────────────────────────────────
───────────────────────────┘
6 rows in set. Elapsed: 0.002 sec.
```

说明 如果返回信息如上所示，则表明数据根据时间做了冷热分离。热数据存放在本地盘中，冷数据存放在HDFS中。

其中，*/var/lib/clickhouse/disks/disk_hdfs*是disk_hdfs元数据的目录，*/mnt/disk{1..4}/clickhouse*是本地盘路径。

创建新的表

- 创建语法

```
CREATE TABLE <yourDataName>.<yourTableName> [ON CLUSTER cluster_emr]
(
  column1 Type1,
  column2 Type2,
  ...
) Engine = MergeTree() --也可以使用Replicated*MergeTree()。
PARTITION BY <yourPartitionKey>
ORDER BY <yourPartitionKey>
TTL <yourTtlKey> TO VOLUME 'remote'
SETTINGS storage_policy='hdfs_ttl';
```

说明 命令中的<yourPartitionKey>为ClickHouse的分区键。<yourTtlKey>为您设置的TTL信息。

- 示例

```
CREATE TABLE test.test
(
    `id` UInt32,
    `t` DateTime
)
ENGINE = MergeTree()
PARTITION BY toStartOfFiveMinute(t)
ORDER BY id
TTL toStartOfMinute(addMinutes(t, 5)) TO VOLUME 'remote'
SETTINGS storage_policy='hdfs_ttl';
```

说明 本示例中，表格会将5分钟内的数据存放在本地，过了5分钟后，数据会被移动到remote volume中，也就是HDFS中。

相关配置

- server-config

  merge_tree.allow_remote_fs_zero_copy_replication：设置为true，以在Replicated*MergeTree使用DiskHDFS等远程存储时利用其自身的多副本进行备份，ClickHouse的一个Shard下的多个副本中的数据仅会备份元数据。

- server-users

  profile.${your-profile-name}.hdfs_replication：设置数据在HDFS上存储的副本个数。

5.2.5.2. 使用OSS进行数据冷热分离

本文为您介绍在阿里云E-MapReduce的ClickHouse集群上，如何通过OSS进行数据的冷热分离。通过本文操作，您既可以在保证集群读写性能的基础上，自动维护集群上的冷热数据，又可以充分利用计算和存储资源，以降低成本。

前提条件

已在EMR控制台上创建EMR-5.5.0及以上版本的ClickHouse集群，详情请参见创建集群。

使用限制

本文操作仅适用于EMR-5.5.0及以上版本的ClickHouse集群。

操作流程

1. 步骤一：在EMR控制台添加磁盘
2. 步骤二：验证配置
3. 步骤三：进行冷热分离

步骤一：在EMR控制台添加磁盘

1. 进入ClickHouse配置页面。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 在**集群服务**页面，单击ClickHouse服务区域的**配置**。
2. 在**服务配置**区域，单击**server-metrika**页签。
3. 修改**storage_configuration**的参数值。
   i. 在**disks**中添加一个OSS类型的磁盘。
   详细信息如下。

```
<disk_oss>
    <type>oss</type>
    <endpoint>http(s)://${yourBucketName}.${yourEndpoint}/${yourFlieName}</endpoint>
    <access_key_id>${yourAccessKeyId}</access_key_id>
    <secret_access_key>${yourAccessKeySecret}</secret_access_key>
    <send_metadata>false</send_metadata>
    <metadata_path>${yourMetadataPath}</metadata_path>
    <cache_enabled>true</cache_enabled>
    <cache_path>${yourCachePath}</cache_path>
    <skip_access_check>false</skip_access_check>
    <min_bytes_for_seek>1048576</min_bytes_for_seek>
    <thread_pool_size>16</thread_pool_size>
    <list_object_keys_size>1000</list_object_keys_size>
  </disk_oss>
```

相关参数描述如下。

参数	是否必填	描述
disk_oss	是	磁盘的名称，您可以自定义。
type	是	磁盘的类型，固定值为oss。
endpoint	是	OSS服务的地址。格式为*http(s)://${yourBucketName}.${yourEndpoint}/${yourFlieName}*。 **说明** endpoint参数值必须以HTTP或HTTPS开头，其中，`${yourBucketName}` 为OSS的Bucket名称，`${yourEndpoint}` 为OSS的访问域名，`{yourFlieName}` 为OSS上的文件名称。例如 `http://clickhouse.oss-cn-hangzhou-internal.aliyuncs.com/test` 。
access_key_id	是	阿里云账号的AccessKey ID。 获取方法请参见获取AccessKey。
secret_access_key	是	阿里云账号的AccessKey Secret。 用于加密签名字符串和OSS，用来验证签名字符串的密钥。获取方法请参见获取AccessKey。
send_metadata	否	在操作OSS文件时，是否添加元数据信息。参数取值如下： ■ true：添加元数据信息。 ■ false（默认值）：不添加元数据信息。

参数	是否必填	描述
metadata_path	否	用于存放本地文件与OSS文件的映射关系。 默认值为*${path}/disks/<disk_name>/*。 **说明** `<disk_name>` 是磁盘的名称，对应参数disk_oss。
cache_enabled	否	是否开启缓存。参数取值如下： ■ true（默认值）：开启缓存。 ■ false：不开启缓存。 Cache具体作用如下： ■ OSS目前所使用的Cache仅用于本地缓存以*.idx*、*.mrk*、*.mrk2*、*.mrk3*、*.txt*和*.dat*为后缀的文件。除了这些文件外，仍然会直接读OSS而非读缓存。 ■ 本地Cache没有容量限制，最大容量即为存储的磁盘容量。 ■ 本地Cache不会以LRU（Least Recently Used）算法之类的策略清理缓存，而是随着文件的生命周期而存在。 ■ 数据第一次被读取时，如果在本地Cache中不存在，则会从OSS中将文件下载至Cache中。 ■ 第一次写入数据时，仅会先写入本地Cache，之后才会从本地Cache写入至OSS中。 ■ 如果一个文件在OSS中被删除，则也会将本地Cache清除；如果一个文件在OSS中被重命名，则也会在本地Cache中重命名。
cache_path	否	缓存路径。 默认值为*${path}/disks/<disk_name>/cache/*。
skip_access_check	否	在加载磁盘时，是否检查具有对磁盘的读写权限。参数取值如下： ■ true（默认值）：检查。 ■ false：不检查。
min_bytes_for_seek	否	最小使用Seek的Byte数量，低于该值时会用Skip代替Seek。默认值为1048576。
thread_pool_size	否	磁盘用于执行 `restore` 命令时所使用的线程池的大小。默认值为16。
list_object_keys_size	否	在某一个key下，单次能够列出的对象最大数目。默认值为1000。

ii. 在**policies**中添加一个新的策略。

策略内容如下。

```
<oss_ttl>
    <volumes>
      <local>
        <!-- 包含默认存储策略下所有的磁盘 -->
        <disk>disk1</disk>
        <disk>disk2</disk>
        <disk>disk3</disk>
        <disk>disk4</disk>
      </local>
      <remote>
        <disk>disk_oss</disk>
      </remote>
    </volumes>
    <move_factor>0.2</move_factor>
</oss_ttl>
```

说明 该部分内容也可以直接添加在default策略中。

4. 保存配置。
   i. 在ClickHouse服务的**配置**页面，单击**保存**。
   ii. 在**确认修改**对话框中，输入执行原因，打开**自动更新配置**开关，单击**确定**。
5. 部署客户端配置。
   i. 在ClickHouse服务的**配置**页面，单击**部署客户端配置**。

ii. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。

iii. 在**确认**对话框中，单击**确定**。

步骤二：验证配置

1. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。
2. 执行如下命令，启动ClickHouse客户端。

```
clickhouse-client -h core-1-1 -m
```

> **说明** 本示例登录core-1-1节点，如果您有多个Core节点，可以登录任意一个节点。

3. 执行如下命令，查看磁盘信息。

```
select * from system.disks;
```

返回信息如下所示。

```
┌─name─────┬─path──────────────────────────────┬──────────free_space─┬──────────total_space─┬─keep_free_space─┬─type
──┐
│ default  │ /var/lib/clickhouse/              │          83868921856 │          84014424064 │               0 │ loca
l  │
│ disk1    │ /mnt/disk1/clickhouse/            │          83858436096 │          84003938304 │        10485760 │ loca
l  │
│ disk2    │ /mnt/disk2/clickhouse/            │          83928215552 │          84003938304 │        10485760 │ loca
l  │
│ disk3    │ /mnt/disk3/clickhouse/            │          83928301568 │          84003938304 │        10485760 │ loca
l  │
│ disk4    │ /mnt/disk4/clickhouse/            │          83928301568 │          84003938304 │        10485760 │ loca
l  │
│ disk_oss │ /var/lib/clickhouse/disks/disk_oss/ │ 18446744073709551615 │ 18446744073709551615 │               0 │ oss
  │
└──────────┴───────────────────────────────────┴──────────────────────┴──────────────────────┴─────────────────┴─────
──┘
```

4. 执行如下命令，查看磁盘存储策略。

```
select * from system.storage_policies;
```

返回信息如下所示。

```
┌─policy_name─┬─volume_name─┬─volume_priority─┬─disks─────────────────────────────┬─volume_type─┬─max_data_part_size─┬─
move_factor─┬─prefer_not_to_merge─┐
│ default     │ single      │               1 │ ['disk1','disk2','disk3','disk4'] │JBOD         │
0 │           0 │                   0 │
│ oss_ttl     │ local       │               1 │ ['disk1','disk2','disk3','disk4'] │JBOD         │
0 │         0.2 │                   0 │
│ oss_ttl     │ remote      │               2 │ ['disk_oss']                      │JBOD         │                  0
│         0.2 │                   0 │
└─────────────┴─────────────┴─────────────────┴───────────────────────────────────┴─────────────┴────────────────────┴─
─────────────┴─────────────────────┘
```

当回显信息如上文所示时，表示磁盘扩容操作完成。

步骤三：进行冷热分离

对已有的表进行改造

1. 在ClickHouse客户端执行如下命令，查看当前的存储策略。

```
SELECT
  storage_policy
FROM system.tables
WHERE database='<yourDatabaseName>' AND name='<yourTableName>';
```

本文示例中命令中的 `<yourDataName>` 为数据库名， `<yourTableName>` 为表名。

如果返回信息如下所示，则需要参见下一步骤添加一个volume。

```
<default>
  <volumes>
    <single>
      <disk>disk1</disk>
      <disk>disk2</disk>
      <disk>disk3</disk>
      <disk>disk4</disk>
    </single>
  </volumes>
</default>
```

2. 扩展当前的存储策略。

   在EMR控制台ClickHouse服务的配置页签，增加volume内容，详细信息如下。

```
<default>
  <volumes>
    <single>
      <disk>disk1</disk>
      <disk>disk2</disk>
      <disk>disk3</disk>
      <disk>disk4</disk>
    </single>
    <!-- 以下是新增的volume remote -->
    <remote>
      <disk>disk_oss</disk>
    </remote>
  </volumes>
  <!-- 多个volume时需要指定move_factor -->
  <move_factor>0.2</move_factor>
</default>
```

3. 执行以下命令，修改TTL。

```
ALTER TABLE <yourDataName>.<yourTableName>
  MODIFY TTL toStartOfMinute(addMinutes(t, 5)) TO VOLUME 'remote';
```

4. 执行以下命令，查看各个part的分布。

```
select partition,name,path from system.parts where database='<yourDataName>' and table='<yourTableName>' and active=1
```

   返回信息如下。

```
┌─partition───────────┬─name──────────────────────┬─path──────────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────┐
│ 2022-01-11 19:55:00 │ 1641902100_1_90_3_193     │ /var/lib/clickhouse/disks/disk_oss/store/fc5/fc50a391-4c16-406b-a396
-6e1104873f68/1641902100_1_90_3_193/  │
│ 2022-01-11 19:55:00 │ 1641902100_91_96_1_193    │ /var/lib/clickhouse/disks/disk_oss/store/fc5/fc50a391-4c16-406b-a396
-6e1104873f68/1641902100_91_96_1_193/ │
│ 2022-01-11 20:00:00 │ 1641902400_97_124_2_193   │ /mnt/disk3/clickhouse/store/fc5/fc50a391-4c16-406b-a396-6e1104873f68
/1641902400_97_124_2_193/             │
│ 2022-01-11 20:00:00 │ 1641902400_125_152_2_193 │ /mnt/disk2/clickhouse/store/fc5/fc50a391-4c16-406b-a396-6e1104873f68
/1641902400_125_152_2_193/            │
│ 2022-01-11 20:00:00 │ 1641902400_153_180_2_193 │ /mnt/disk4/clickhouse/store/fc5/fc50a391-4c16-406b-a396-6e1104873f68
/1641902400_153_180_2_193/            │
│ 2022-01-11 20:00:00 │ 1641902400_181_186_1_193 │ /mnt/disk3/clickhouse/store/fc5/fc50a391-4c16-406b-a396-6e1104873f68
/1641902400_181_186_1_193/            │
│ 2022-01-11 20:00:00 │ 1641902400_187_192_1_193 │ /mnt/disk4/clickhouse/store/fc5/fc50a391-4c16-406b-a396-6e1104873f68
/1641902400_187_192_1_193/            │
└─────────────────────┴───────────────────────────┴───────────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────┘

7 rows in set. Elapsed: 0.002 sec.
```

   说明 如果结果类似上所示，则表明数据根据时间做了冷热分离。热数据存放在本地盘中，冷数据存放在OSS中。

   其中，*/var/lib/clickhouse/disks/disk_oss*是disk_oss的metadata_path的默认值。*/mnt/disk{1..4}/clickhouse*是本地盘路径。

创建新的表

- 创建语法

```
CREATE TABLE <yourDataName>.<yourTableName> [ON CLUSTER cluster_emr]
(
  column1 Type1,
  column2 Type2,
  ...
) Engine = MergeTree() -- or Replicated*MergeTree()
PARTITION BY <yourPartitionKey>
ORDER BY <yourPartitionKey>
TTL <yourTtlKey> TO VOLUME 'remote'
SETTINGS storage_policy='oss_ttl';
```

说明 命令中的<yourPartitionKey>为ClickHouse的分区键。<yourTtlKey>为您设置的TTL信息。

- 示例

```
CREATE TABLE test.test
(
    `id` UInt32,
    `t` DateTime
)
ENGINE = MergeTree()
PARTITION BY toStartOfFiveMinute(t)
ORDER BY id
TTL toStartOfMinute(addMinutes(t, 5)) TO VOLUME 'remote'
SETTINGS storage_policy='oss_ttl';
```

说明 本示例中，表格会将5分钟内的数据存放在本地，过了5分钟后，数据会被移动到remote volume中，也就是OSS中。

相关配置

- **server-config**

  merge_tree.allow_remote_fs_zero_copy_replication：设置为true，以在Replicated*MergeTree使用DiskOSS等远程存储时利用OSS的多副本进行备份，ClickHouse的一个Shard下的多个副本中的数据仅会备份元数据。

- **server-users**
  - profile.${your-profile-name}.oss_min_upload_part_size：Write Buffer中的数据量高于该参数值时，会将数据写到OSS中。
  - profile.${your-profile-name}.oss_max_single_part_upload_size}：Write Buffer中的数据量高于该参数值时，使用MultipartUpload，详情请参见分片上传（MultipartUpload）。

5.2.6. 事务使用

事务是阿里云E-MapReduce（简称EMR）产品自研的特性，目前处于Experimental阶段。本文为您介绍事务的状态、参数以及事务的相关操作。

前提条件

已在EMR控制台上创建ClickHouse集群，且集群中已配置了可使用的ZooKeeper服务，创建详情请参见创建集群。

使用限制

本文操作适用于EMR-3.39.0及以上版本和EMR-5.5.0及以上版本的ClickHouse集群。

注意事项

- 事务仅支持Insert，且Insert的对象必须是以Replicated*MergeTree或者*MergeTree为存储引擎的表。
- 当前的事务为单机事务。
- 本功能处在Experimental阶段，请谨慎使用。

事务API

事务允许的操作如下表所示。

操作	描述
begin	开启一个事务。
write_data	在一个事务内写数据。
commit	提交一个事务。

操作	描述
rollback	回滚一个未提交的事务。

事务状态如下表所示。

状态	描述
UNKNOWN	事务未开启，此时仅允许begin操作。
INITIALIZED	事务已开启，此时允许所有操作。
COMMITTING	事务正在被提交，不允许执行begin或write_data两种操作。
COMMITTED	事务已经被提交，不再允许任何操作。
ABORTING	事务正在被回滚，不再允许任何操作。
ABORTED	事务已经被回滚，不再允许任何操作。

错误码	描述
0	执行成功。
11001	未知的事务操作类型。
11002	未知的事务状态。
11003	多个机器存在一个相同的事务处理。
11004	重定向至其他机器。
11005	不支持的存储引擎。
11006	事务未在任何机器上处理。
11201	当前事务状态为COMMITTING，但收到了begin请求。
11202	当前事务状态为COMMITTED，但收到了begin请求。
11203	当前事务状态为ABORTING，但收到了begin请求。
11204	当前事务状态为ABORTED，但收到了begin请求。
11301	当前事务状态为UNKNOWN，但收到了commit请求。
11302	当前事务状态为ABORTING，但收到了commit请求。
11303	当前事务状态为ABORTED，但收到了commit请求。
11401	当前事务状态为UNKNOWN，但收到了rollback请求。
11402	当前事务状态为COMMITTED，但收到了rollback请求。
11501	当前事务状态为UNKNOWN，但收到了write_data请求。
11502	当前事务状态为COMMITTING，但收到了write_data请求。
11503	当前事务状态为COMMITTED，但收到了write_data请求。
11504	当前事务状态为ABORTING，但收到了write_data请求。

名称	类型	是否必选	描述
action	字符串	是	事务的操作。
id	字符串	是	事务的ID。
stacktrace	布尔值	否	是否在执行出错时向Client发送详细的堆栈信息，默认值为false。

名称	类型	是否必选	描述
Host	字符串	是	ClickHouse HTTP Server的可访问地址，通常由*ip:port*或*hostname:port*组成，其中port通常为8123。
Authorization	字符串	否	ClickHouse用以验证身份的字段，使用HTTP Basic Authentication作为验证手段。 注意 如果指定了该验证方式，则不能指定X-ClickHouse-User和X-ClickHouse-Key，否则会报错。
X-ClickHouse-User	字符串	否	ClickHouse用户名。 注意 通常不推荐该验证方式，因为该验证方式为明文。指定该验证方式，则不能指定 Authorization，否则会报错。
X-ClickHouse-Key	字符串	否	ClickHouse用户密码。 注意 通常不推荐该验证方式，因为该验证方式为明文。指定该验证方式，则不能指定 Authorization，否则会报错。
X-ClickHouse-TransactionId	字符串	否	ClickHouse事务ID，仅用于事务写。

名称	类型	描述
Location	字符串	重定向的位置。

Response Body会以JSON的形式返回，JSON内含字段如下。

名称	类型	描述
code	整型	错误码。
message	字符串	错误信息。

ACTION TYPE

STATE TYPE

错误码

HTTP PARAM

Request Header

Response Header

Response Body

使用示例

在emr-header-1节点上开始一个ID为1f6676e3-496f-409e-b64e-0eb22b1c****的事务，执行命令如下。

```
POST /transaction?action=begin&id=1f6676e3-496f-409e-b64e-0eb22b1c**** HTTP/1.1
Host: emr-header-1:8123
Authorization: Basic ZGVmYXVsdDo=
```

- 执行成功时，返回如下信息。

```
HTTP/1.1 200 OK
Date: Fri, 14 Jan 2022 07:45:32 GMT
Connection: Close
Content-Type: application/json;charset=UTF-8
{
  "code": 0,
  "message": "Success."
}
```

- 执行失败时，返回如下类似信息。

```
HTTP/1.1 200 OK
Date: Fri, 14 Jan 2022 08:00:43 GMT
Connection: Close
Content-Type: application/json;charset=UTF-8
{
  "code": 11202,
  "message": "Received the request of transaction begin after the transaction was committed."
}
```

- 重定向时，返回如下信息。

```
HTTP/1.1 307 OK
Date: Fri, 14 Jan 2022 07:47:18 GMT
Connection: Close
Location: http://192.168.**.**:8123/transaction?action=begin&id=1f6676e3-496f-409e-b64e-0eb22b1c****
Content-Type: application/json;charset=UTF-8
{
  "code": 11004,
  "message": "Redirect to other replica."
}
```

> **说明** 向其他节点上发送请求会被重定向至emr-header-1，其IP地址为192.168.**.**。

在emr-header-1节点上，向以下表结构中写入数据，写的过程包含在事务1f6676e3-496f-409e-b64e-0eb22b1c****中。

```
CREATE TABLE test.test ON CLUSTER cluster_emr
(
    `id` UInt32,
    `t` DateTime
)
ENGINE = MergeTree
PARTITION BY toStartOfFiveMinute(t)
ORDER BY id;
```

执行命令如下。

```
POST /?query=INSERT%20INTO%20test.test%20VALUES%20(1,'2022-01-13%2020:03:01') HTTP/1.1
Authorization: Basic ZGVmYXVsdDo=
Host: emr-header-1.cluster-276031:8123
Content-Length: 0
X-ClickHouse-TransactionId: 31a8b711-c174-481a-be9d-e2c229c8****
```

- 执行成功时，返回如下信息。

```
HTTP/1.1 200 OK
Date: Fri, 14 Jan 2022 07:53:20 GMT
Connection: Keep-Alive
Content-Type: text/tab-separated-values; charset=UTF-8
X-ClickHouse-Server-Display-Name: emr-header-1.cluster-276031
Transfer-Encoding: chunked
X-ClickHouse-Query-Id: 1135ae61-a38d-400d-94e1-df3a67cd****
X-ClickHouse-Format: TabSeparated
X-ClickHouse-Timezone: Asia/Shanghai
Keep-Alive: timeout=3
X-ClickHouse-Summary: {"read_rows":"0","read_bytes":"0","written_rows":"0","written_bytes":"0","total_rows_to_read":"0"}
```

- 执行失败时，返回如下类似信息。

```
HTTP/1.1 500 Internal Server Error
Date: Fri, 14 Jan 2022 08:01:54 GMT
Connection: Keep-Alive
Content-Type: text/tab-separated-values; charset=UTF-8
X-ClickHouse-Server-Display-Name: emr-header-1.cluster-276031
Transfer-Encoding: chunked
X-ClickHouse-Query-Id: c9a65f05-d52e-4f2a-a892-77f11799****
X-ClickHouse-Format: TabSeparated
X-ClickHouse-Timezone: Asia/Shanghai
X-ClickHouse-Exception-Code: 11503
Keep-Alive: timeout=3
X-ClickHouse-Summary: {"read_rows":"0","read_bytes":"0","written_rows":"0","written_bytes":"0","total_rows_to_read":"0"}
Code: 11503, e.displayText() = DB::Exception: Received the request of writing data after the transaction was committed. (
version 21.3.13.1)
```

- 重定向时，返回如下信息。

```
HTTP/1.1 307 Temporary Redirect
Date: Fri, 14 Jan 2022 07:52:15 GMT
Connection: Keep-Alive
Content-Type: text/plain; charset=UTF-8
X-ClickHouse-Server-Display-Name: emr-worker-1.cluster-276031
Transfer-Encoding: chunked
Location: http://192.168.**.**:8123/?query=INSERT%20INTO%20test.test%20VALUES%20(1,'2022-01-13%2020:04:21')
X-ClickHouse-Exception-Code: 11004
Code: 11004, e.displayText() = DB::Exception: Failed to check replica under transaction path in zookeeper. (version 21.3.
13.1)
```

> **说明** 向其他节点上发送请求会被重定向至emr-header-1，其IP地址为192.168.**.**。

在emr-header-1节点上提交一个ID为1f6676e3-496f-409e-b64e-0eb22b1c****的事务，执行命令如下。

```
POST /transaction?action=commit&id=1f6676e3-496f-409e-b64e-0eb22b1c**** HTTP/1.1
Host: emr-header-1:8123
Authorization: Basic ZGVmYXVsdDo=
```

- 提交成功时，返回如下信息。

```
HTTP/1.1 200 OK
Date: Fri, 14 Jan 2022 07:55:09 GMT
Connection: Close
Content-Type: application/json;charset=UTF-8
{
  "code": 0,
  "message": "Success."
}
```

- 提交失败时，返回如下类似信息。

```
HTTP/1.1 200 OK
Date: Fri, 14 Jan 2022 08:03:39 GMT
Connection: Close
Content-Type: application/json;charset=UTF-8
{
  "code": 11303,
  "message": "Received the request of transaction commit after the transaction was aborted."
}
```

- 重定向时，返回如下信息。

```
HTTP/1.1 307 OK
Date: Fri, 14 Jan 2022 07:56:30 GMT
Connection: Close
Location: http://192.168.**.**:8123/transaction?action=commit&id=1f6676e3-496f-409e-b64e-0eb22b1c****
Content-Type: application/json;charset=UTF-8
{
  "code": 11004,
  "message": "Redirect to other replica."
}
```

> **说明** 向其他节点上发送请求会被重定向至emr-header-1，其IP地址为192.168.**.**。

在emr-header-1节点上回滚一个ID为1f6676e3-496f-409e-b64e-0eb22b1c****的事务，执行命令如下。

```
POST /transaction?action=rollback&id=1f6676e3-496f-409e-b64e-0eb22b1c**** HTTP/1.1
Host: emr-header-1:8123
Authorization: Basic ZGVmYXVsdDo=s
```

- 回滚成功时，返回如下信息。

```
HTTP/1.1 200 OK
Date: Fri, 14 Jan 2022 08:03:08 GMT
Connection: Close
Content-Type: application/json;charset=UTF-8
{
  "code": 0,
  "message": "Success."
}
```

- 回滚失败时，返回如下类似信息。

```
HTTP/1.1 200 OK
Date: Fri, 14 Jan 2022 07:58:41 GMT
Connection: Close
Content-Type: application/json;charset=UTF-8
{
  "code": 11402,
  "message": "Received the request of transaction rollback after the transaction was committed."
}
```

- 重定向时，返回如下信息。

```
HTTP/1.1 307 OK
Date: Fri, 14 Jan 2022 07:59:33 GMT
Connection: Close
Location: http://192.168.**.**:8123/transaction?action=rollback&id=1f6676e3-496f-409e-b64e-0eb22b1c****
Content-Type: application/json;charset=UTF-8
{
  "code": 11004,
  "message": "Redirect to other replica."
}
```

> **说明** 向其他节点上发送请求会被重定向至emr-header-1，其IP地址为192.168.**.**。

开始事务

事务写

提交事务

回滚事务

相关配置

参数	描述
transaction.enable_public_ip	ClickHouse Server中Transaction需要一个用于标识自身的IP地址，默认使用私网IP地址。 在EMR控制台ClickHouse服务的**server-config**页签中，设置此参数值为true，以使用公网IP地址标识自身，但需要所有节点均开启公网IP。
transaction.path_prefix	Transaction在ZooKeeper中的目录前缀。默认为*/clickhouse/transactions*。
transaction.transaction_timeout_ms	Transaction超时时间，默认值为86400000毫秒（1天）。
transaction.auxiliary_zookeepers	Transaction使用的ZooKeeper名称，默认为default，对应了配置在EMR控制台ClickHouse服务的**server-metrika**页签中参数**zookeeper_servers**的Zookeeper。
transaction.distributed_lock_timeout_ms	分布式锁超时的时间，默认值为120000毫秒（2分钟）。

5.2.7. 常见问题

本文汇总了ClickHouse使用时的常见问题。

- 如何创建ClickHouse用户？
- 数据丢失，如何处理？
- 如何扩缩容磁盘？
- 报错提示Memory limit (for total) exceeded时，该如何处理？
- 报错提示Memory limit (for query) exceeded时，该如何处理？
- 报错提示Memory limit (for user) exceeded时，该如何处理？

如何创建ClickHouse用户？

您可以通过以下两种方法创建ClickHouse用户：

- 通过在EMR控制台新增自定义配置项创建ClickHouse用户

  在EMR控制台ClickHouse服务的**配置**页面，单击**server-users**页签，新增配置项，参数为**users.<YourUserName>.password**或**users.<YourUserName>.password_sha256_hex**或**users.<YourUserName>.password_double_sha1_hex**，参数值您可以自定义配置，保存该配置项并重启服务，即可创建用户。

  参数中的<YourUserName>需要替换为您待创建用户的名称。

  > **说明** 添加组件参数详情，请参见管理组件参数。重启服务详情，请参见重启服务。

- 通过ClickHouse客户端创建ClickHouse用户

  i. 在EMR控制台ClickHouse服务的**配置**页面，单击**server-users**页签，新增参数为**users.default.access_management**，参数值为1的配置项，保存该配置并重启服务。使用默认用户连接ClickHouse集群。

  ii. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。

  iii. 执行以下命令，进入ClickHouse客户端。

  ```
  clickhouse-client -h core-1-1 -m
  ```

  > **说明** 本示例登录core-1-1节点，如果您有多个Core节点，可以登录任意一个节点。

  iv. 执行以下命令，创建用户。

  ```
  CREATE USER IF NOT EXISTS user_test ON CLUSTER new_cluster_emr IDENTIFIED WITH plaintext_password BY '123456';
  ```

  > **说明** 本示例创建的用户为user_test，密码为123456，您可以根据实际情况修改。

  创建用户命令格式如下。

  ```
  CREATE USER [IF NOT EXISTS | OR REPLACE] name1 [ON CLUSTER cluster_name1]
          [, name2 [ON CLUSTER cluster_name2] ...]
      [NOT IDENTIFIED | IDENTIFIED {[WITH {no_password | plaintext_password | sha256_password | sha256_hash | double_sh
  a1_password | double_sha1_hash}] BY {'password' | 'hash'}} | {WITH ldap SERVER 'server_name'} | {WITH kerberos [REALM
  'realm']}]
      [HOST {LOCAL | NAME 'name' | REGEXP 'name_regexp' | IP 'address' | LIKE 'pattern'} [,...] | ANY | NONE]
      [DEFAULT ROLE role [,...]]
      [GRANTEES {user | role | ANY | NONE} [,...] [EXCEPT {user | role} [,...]]]
      [SETTINGS variable [= value] [MIN [=] min_value] [MAX [=] max_value] [READONLY | WRITABLE] | PROFILE 'profile_nam
  e'] [,...];
  ```

  v. 执行以下命令，查看已有的用户。

  ```
  SHOW USERS;
  ```

  返回已有的用户。

  ```
  ┌─name─────┐
  │ default  │
  │ user_test │
  │ user_test2 │
  └──────────┘
  ```

数据丢失，如何处理？

- 问题现象：向ClickHouse写入A条数据，但实际读出来只有B条，且B小于A。
- 问题原因：通常情况下，ClickHouse中的数据不会丢失。但是在一个shard存在至少两个副本，本地表的表引擎为*MergeTree，同时使用分布式表进行读数的场景下，可能会出现读出来的数据少于写入数据的情况。

无论是通过分布式表写数据，还是直接写入本地表，每个机器上都会存在数据；而通过分布式表读数据时，默认每个shard仅会使用一个连接，此时的连接数是会少于机器数的，所以存在一些机器上的数据无法被读取的情况。代码示例如下。

```
CREATE TABLE db.table_local
(
  ...
)
Engine = MergeTree()
CREATE TABLE db.table_distributed
(
  ...
)
Engine = Distributed(cluster_emr, db, table_local, rand());
```

- 处理方法：

方式	操作
方式一（推荐）	删除*db.table_local*并后重新创建表，使用复制表作为本地表。
方式二（不推荐）	您可以在EMR控制台ClickHouse服务的**配置**页面，单击**server-users**页签，单击右上角的**自定义配置**，新增参数profiles.<your_profile_name>.max_parallel_replicas，参数值至少为每个shard下replica的数量，并确保users.<your_clickhouse-client_name>.profile值为*<your_profile_name>*。 **说明** *<your_profile_name>*和*<your_clickhouse-client_name>*均需要替换为您实际的名称。各参数详细信息，请参见访问权限控制。

如果以上方法还是无法解决您的问题，请提交工单或购买专家服务处理。

如何扩缩容磁盘？

具体的操作步骤请参见扩容磁盘和缩容磁盘。

报错提示Memory limit (for total) exceeded时，该如何处理？

- 问题原因：内存超过了server可使用的总内存。
- 处理方法：在EMR控制台ClickHouse服务的**配置**页面，单击**server-config**页签，单击右上角的**自定义配置**，新增参数max_server_memory_usage，该参数可以配置的最大值为 **机器物理内存大小** * max_server_memory_usage_to_ram_ratio 。

说明 ClickHouse中max_server_memory_usage_to_ram_ratio参数的默认值为0.9，如果您需要调整该参数值，可以新增max_server_memory_usage_to_ram_ratio参数，参数值您可以根据实际情况调整。

报错提示Memory limit (for query) exceeded时，该如何处理？

- 问题原因：内存超过了单次Query可使用的最大内存。
- 处理方法：

场景	操作
全局方式	在EMR控制台ClickHouse服务的**配置**页面，单击**server-config**页签，单击右上角的**自定义配置**，新增参数profiles.<your_profile_name>.max_memory_usage，并确保users.<your_clickhouse-client_name>.profile值为*<your_profile_name>*。 **说明** *<your_profile_name>*和*<your_clickhouse-client_name>*均需要替换为您实际的名称。各参数详细信息，请参见访问权限控制。
针对使用clickhouse-client	在EMR控制台ClickHouse服务的**配置**页面，单击**client-config**页签，单击右上角的**自定义配置**，新增参数max_memory_usage。
针对某一次会话Session	可以直接 SET max_memory_usage=xxxx ，该配置在Session生命周期内均会生效。
针对某一次Query	可以在SQL中添加配置，该配置仅对当前Query生效。 例如， SELECT column FROM table SETTINGS max_memory_usage=xxxx 。

报错提示Memory limit (for user) exceeded时，该如何处理？

- 问题原因：内存超过了单个用户可使用的最大内存。

- 处理方法：

场景	操作
全局方式	在EMR控制台ClickHouse服务的**配置**页面，单击**server-users**页签，单击右上角的**自定义配置**，新增参数profiles.<your_profile_name>.max_memory_usage_for_user，并确保users.<your_clickhouse-client_name>.profile值为*<your_profile_name>*。 **说明** *<your_profile_name>*和*<your_clickhouse-client_name>*均需要替换为您实际的名称。各参数详细信息，请参见访问权限控制。
针对使用clickhouse-client	在EMR控制台ClickHouse服务的**配置**页面，单击**client-config**页签，单击右上角的**自定义配置**，新增参数max_memory_usage_for_user。
针对某一次会话Session	可以直接使用命令`SET max_memory_usage_for_user=xxxx`，该配置在Session生命周期内均会生效。
针对某一次Query	可以在SQL中添加配置，该配置仅对当前Query生效。 例如，`SELECT column FROM table SETTINGS max_memory_usage_for_user=xxxx`。

5.3. StarRocks

5.3.1. StarRocks概述

本文为您介绍什么是StarRocks，以及StarRocks的特性和应用场景。

StarRocks介绍

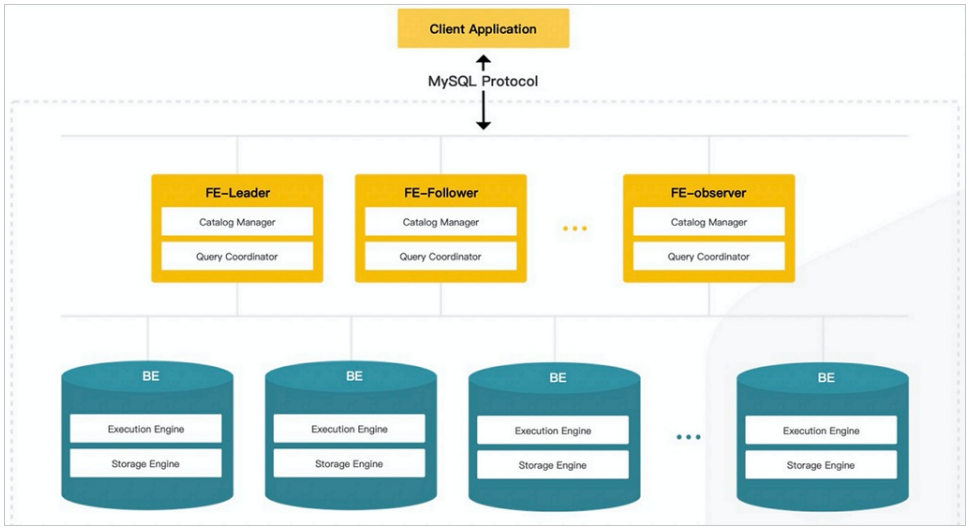
EMR StarRocks提供了99元首月试用的优惠活动，如果您感兴趣可以申请测试，详情请参见StarRocks测试指南。

- StarRocks是新一代极速全场景MPP（Massively Parallel Processing）数据库，致力于构建极速和统一分析体验。
- StarRocks兼容MySQL协议，可使用MySQL客户端和常用BI工具对接StarRocks来分析数据。
- StarRocks采用分布式架构：
  - 对数据表进行水平划分并以多副本存储。
  - 集群规模可以灵活伸缩，支持10 PB级别的数据分析。
  - 支持MPP框架，并行加速计算。
  - 支持多副本，具有弹性容错能力。

StarRocks特性

StarRocks的架构设计融合了MPP数据库，以及分布式系统的设计思想，其特性如下所示。

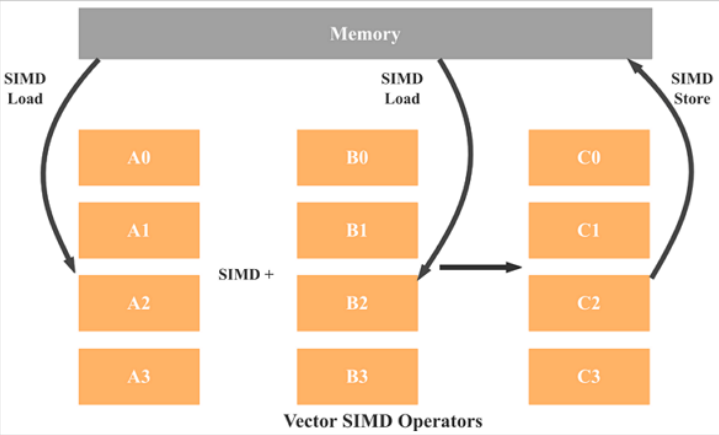
架构精简



- StarRocks内部通过MPP计算框架完成SQL的具体执行工作。MPP框架能够充分的利用多节点的计算能力，整个查询可以并行执行，从而实现良好的交互式分析体验。

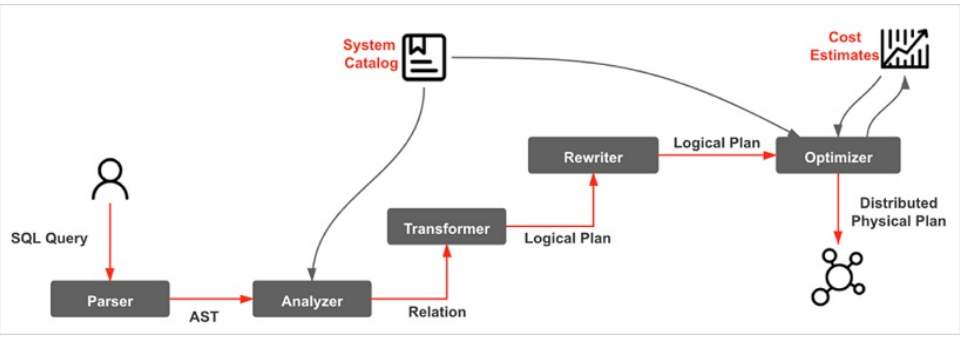
- StarRocks集群不需要依赖任何其他组件，易部署、易维护和极简的架构设计，降低了StarRocks系统的复杂度和维护成本，同时也提升了系统的可靠性和扩展性。管理员只需要专注于StarRocks系统，无需学习和管理任何其他外部系统。

全面向量化引擎



StarRocks的计算层全面采用了向量化技术，将所有算子、函数、扫描过滤和导入导出模块进行了系统性优化。通过列式的内存布局、适配CPU的SIMD指令集等手段，充分发挥了现代CPU的并行计算能力，从而实现亚秒级别的多维分析能力。

智能查询优化



StarRocks通过CBO优化器（Cost Based Optimizer）可以对复杂查询自动优化。无需人工干预，就可以通过统计信息合理估算执行成本，生成更优的执行计划，大大提高了AdHoc和ETL场景的数据分析效率。

联邦查询

StarRocks支持使用外表的方式进行联邦查询，当前可以支持Hive、MySQL、Elasticsearch、Iceberg和Hudi类型的外表，您无需通过数据导入，可以直接进行数据查询加速。

高效更新

StarRocks支持明细模型、聚合模型、主键模型和更新模型，其中主键模型可以按照主键进行Upsert或Delete操作，通过存储和索引的优化可以在并发更新的同时实现高效的查询优化，更好的服务实时数仓的场景。

智能物化视图

- StarRocks支持智能的物化视图。您可以通过创建物化视图，预先计算生成预聚合表用于加速聚合类查询请求。
- StarRocks的物化视图能够在数据导入时自动完成汇聚，与原始表数据保持一致。
- 查询的时候，您无需指定物化视图，StarRocks能够自动选择最优的物化视图来满足查询请求。

标准SQL

- StarRocks支持标准的SQL语法，包括聚合、JOIN、排序、窗口函数和自定义函数等功能。
- StarRocks可以完整支持TPC-H的22个SQL和TPC-DS的99个SQL。
- StarRocks兼容MySQL协议语法，可以使用现有的各种客户端工具、BI软件访问StarRocks，对StarRocks中的数据进行拖拽式分析。

流批一体

- StarRocks支持实时和批量两种数据导入方式。
- StarRocks支持的数据源有Kafka、HDFS和本地文件。
- StarRocks支持的数据格式有ORC、Parquet和CSV等。
- StarRocks可以实时消费Kafka数据来完成数据导入，保证数据不丢不重（exactly once）。
- StarRocks也可以从本地或者远程（HDFS）批量导入数据。

高可用易扩展

- StarRocks的元数据和数据都是多副本存储，并且集群中服务有热备，多实例部署，避免了单点故障。
- 集群具有自愈能力，可弹性恢复，节点的宕机、下线和异常都不会影响StarRocks集群服务的整体稳定性。
- StarRocks采用分布式架构，存储容量和计算能力可近乎线性水平扩展。StarRocks单集群的节点规模可扩展到数百节点，数据规模可达到10 PB级别。
- 扩缩容期间无需停服，可以正常提供查询服务。
- StarRocks中表模式热变更，可通过一条简单SQL命令动态地修改表的定义，例如增加列、减少列和新建物化视图等。同时，处于模式变更中的表也可以正常导入和查询数据。

应用场景

StarRocks可以满足企业级用户的多种分析需求，具体的业务场景如下所示：

- OLAP多维分析
  - 用户行为分析
  - 用户画像、标签分析、圈人
  - 高维业务指标报表
  - 自助式报表平台
  - 业务问题探查分析
  - 跨主题业务分析
  - 财务报表
  - 系统监控分析
- 实时数仓
  - 电商大促数据分析
  - 教育行业的直播质量分析
  - 物流行业的运单分析
  - 金融行业绩效分析、指标计算
  - 广告投放分析
  - 管理驾驶舱
  - 探针分析APM（Application Performance Management）
- 高并发查询
  - 广告主报表分析
  - 零售行业渠道人员分析
  - SaaS行业面向用户分析报表
  - Dashboard多页面分析
- 统一分析

  通过使用一套系统解决多维分析、高并发查询、预计算、实时分析和Ad-Hoc查询等场景，降低系统复杂度和多技术栈开发与维护成本。

5.3.2. 快速入门

5.3.2.1. 创建集群

本文为您介绍创建StarRocks集群的详细操作步骤和相关配置。

前提条件

已在目标地域创建一个专有网络和交换机，详情请参见创建和管理专有网络和创建和管理交换机。

操作步骤

1. 进入创建集群页面。
   i. 登录EMR on ECS控制台。

ii. （可选）在顶部菜单栏处，根据实际情况选择地域和资源组。
   - 地域：创建的集群将会在对应的地域内，一旦创建不能修改。
   - 资源组：默认显示账号全部资源。

iii. 单击上方的**创建集群**，进行创建。

2. 配置集群信息。

创建集群时，您需要对集群进行软件配置、硬件配置和基础配置。

> 注意　集群创建完成后，除了集群名称以外，其他配置均无法修改，所以在创建时请仔细确认各项配置。

i. 软件配置。

配置项	示例	描述
地域	华东1（杭州）	创建的集群将会在对应的地域内，一旦创建不能修改。
业务场景	数据分析场景	选择**数据分析场景**。
产品版本	EMR-3.39.1	默认最新的软件版本。
服务高可用	关闭	默认关闭。开启后集群会有三个Master节点来支持ResourceManager和NameNode的高可用。 您也可以修改节点数量。
可选服务	StarRocks	根据您的实际需求选择其他的一些组件，被选中的组件会默认启动相关的服务进程。
高级设置	不开启	**软件自定义配置**：可指定JSON文件对集群中的基础软件（例如Hadoop、Spark和Hive等）进行配置。默认不开启。

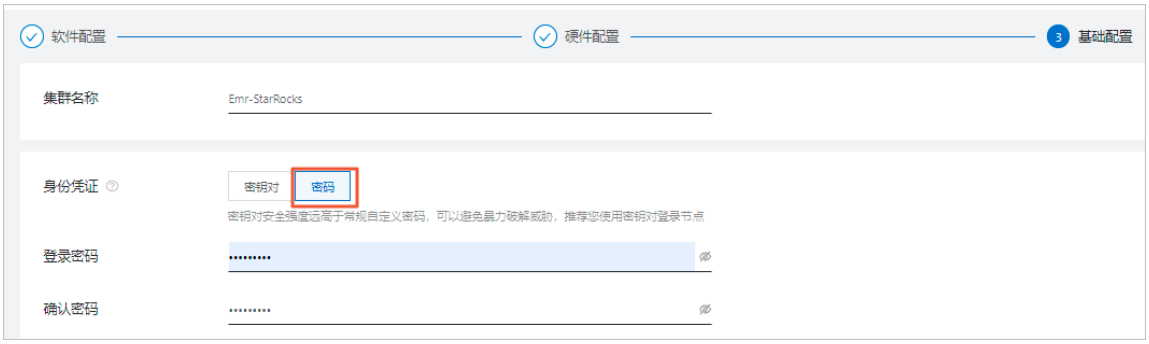
ii. 硬件配置。

配置项	示例	描述
付费类型	按量付费	默认包年包月。当前支持的付费类型如下： ■ **按量付费**：一种后付费模式，即先使用再付费。按量付费是根据实际使用的小时数来支付费用，每小时计费一次，适合短期的测试任务或是灵活的动态任务。 ■ **包年包月**：一种预付费模式，即先付费再使用。 **说明** 建议测试场景下使用**按量付费**，测试正常后再新建一个**包年包月**的生产集群正式使用。
可用区	华东1（杭州） 可用区 I	可用区为在同一地域下的不同物理区域，可用区之间内网互通。通常使用默认的可用区即可。
专有网络	starrocks_test/vpc-bp1f4epmkvncimpgs****	默认选择已有的专有网络。 如需创建新的专有网络，请在专有网络控制台新创建一个，详情请参见创建和管理专有网络。
交换机	vsw_test/vsw-bp1e2f5fhaplp0g6p****	选择在对应VPC下可用区的交换机，如果在这个可用区没有可用的交换机，则需要在专有网络控制台新创建一个，详情请参见创建和管理交换机。
安全组	sg-bp1ddw7sm2risw****/sg-bp1ddw7sm2risw****	默认选择已有的安全组。安全组详情请参见安全组概述。 您也可以单击**新建安全组**，在ECS控制台新建一个安全组，详情请参见创建安全组。 **注意** 禁止使用ECS上创建的企业安全组。
挂载公网	开启	集群是否挂载弹性公网IP地址，默认不开启。 **说明** 创建后如果您需要使用公网IP地址访问，请在ECS上申请开通公网IP地址，详情请参见弹性公网IP中的申请EIP的内容。
实例	使用默认值	您可以根据需要选择实例规格，详情请参见实例规格族。 ■ **系统盘**：根据需要选择ESSD云盘、SSD云盘或者高效云盘。 ■ **系统盘大小**：根据需要调整磁盘容量，推荐至少120 GB。取值范围为60 ~ 500 GB。 ■ **数据盘**：根据需要选择ESSD云盘、SSD云盘或者高效云盘。 ■ **数据盘大小**：根据需要调整磁盘容量，推荐至少80 GB。取值范围为40 ~ 32768 GB。 ■ **实例数量**：默认1台Master，1台Core。

iii. 基础配置。

在**基础信息**区域，配置如下参数。

> 注意　暂不支持高级配置区域的参数，因此请勿设置。



配置项	示例	描述
集群名称	Emr-StarRocks	集群的名字，长度限制为1~64个字符，仅可使用中文、字母、数字、短划线（-）和下划线（_）。
身份凭证	自定义密码	**密钥对**（默认）：使用SSH密钥对登录Linux实例。 关于密钥对的使用详情，请参见SSH密钥对。
		密码：设置Master节点的登录密码，使用密码对登录Linux实例。 密码规则：8~30个字符，且必须同时包含大写字母、小写字母、数字和特殊字符。 特殊字符包括：感叹号（!）、at（@）、井号（#）、美元符号（$）、百分号（%）、乘方（^）、and（&）和星号（*）。
高级设置	根据需求配置	■ **添加用户**：可选配置，添加访问开源大数据软件Web UI的账号。 ■ **ECS应用角色**：当用户的程序在EMR计算节点上运行时，可不填写阿里云AccessKey来访问相关的云服务（例如OSS），EMR会自动申请一个临时AccessKey来授权本次访问。**ECS应用角色**用于控制这个AccessKey的权限。 ■ **引导操作**：可选配置，您可以在集群启动Hadoop前执行您自定义的脚本。 ■ **资源组**：可选配置。

3. 在**确认订单**页面，选中**E-MapReduce服务条款**复选框。
4. 单击**创建**。

创建集群后可以通过刷新页面来查看进度，当集群**状态**显示为**运行中**时，表示集群创建成功。

5.3.2.2. 快速使用StarRocks

5.3.2.2.1. 基础使用

本文通过示例为您介绍，如何快速使用E-MapReduce上的StarRocks集群进行基本的建表和查询操作。

前提条件

已创建StarRocks集群，详情请参见创建集群。

操作步骤

1. 使用SSH方式登录StarRocks集群，详情请参见登录集群。

   主节点的公网IP地址：需要您在新版控制台节点管理页签的emr_master机器组下查看。

2. 执行以下命令，连接StarRocks集群。

```
mysql -h127.0.0.1  -P 9030 -uroot
```

3. 执行以下命令，创建数据库并选择数据库。

```
CREATE DATABASE IF NOT EXISTS load_test;
USE load_test;
```

4. 执行以下命令，创建表。

```
 CREATE TABLE insert_wiki_edit
(
    event_time DATETIME,
    channel VARCHAR(32) DEFAULT '',
    user VARCHAR(128) DEFAULT '',
    is_anonymous TINYINT DEFAULT '0',
    is_minor TINYINT DEFAULT '0',
    is_new TINYINT DEFAULT '0',
    is_robot TINYINT DEFAULT '0',
    is_unpatrolled TINYINT DEFAULT '0',
    delta INT SUM DEFAULT '0',
    added INT SUM DEFAULT '0',
    deleted INT SUM DEFAULT '0'
)
AGGREGATE KEY(event_time, channel, user, is_anonymous, is_minor, is_new, is_robot, is_unpatrolled)
PARTITION BY RANGE(event_time)
(
    PARTITION p06 VALUES LESS THAN ('2015-09-12 06:00:00'),
    PARTITION p12 VALUES LESS THAN ('2015-09-12 12:00:00'),
    PARTITION p18 VALUES LESS THAN ('2015-09-12 18:00:00'),
    PARTITION p24 VALUES LESS THAN ('2015-09-13 00:00:00')
)
DISTRIBUTED BY HASH(user) BUCKETS 10
PROPERTIES("replication_num" = "1");
```

5. 执行以下命令，导入测试数据。

```
INSERT INTO insert_wiki_edit VALUES("2015-09-12 00:00:00","#en.wikipedia","GELongstreet",0,0,0,0,36,36,0),("2015-09-1
2 00:00:00","#ca.wikipedia","PereBot",0,1,0,1,0,17,17,0);
```

6. 执行以下命令，查询数据。

```
select * from insert_wiki_edit;
```

返回信息如下所示。

```
+---------------------+---------------+--------------+--------------+----------+--------+----------+----------------+--
-----+-------+---------+
| event_time          | channel       | user         | is_anonymous | is_minor | is_new | is_robot | is_unpatrolled | d
elta | added | deleted |
+---------------------+---------------+--------------+--------------+----------+--------+----------+----------------+--
-----+-------+---------+
| 2015-09-12 00:00:00 | #en.wikipedia | GELongstreet |            0 |        0 |      0 |        0 |              0 |
36 |    36 |       0 |
| 2015-09-12 00:00:00 | #ca.wikipedia | PereBot      |            0 |        1 |      0 |        1 |              0 |
17 |    17 |       0 |
+---------------------+---------------+--------------+--------------+----------+--------+----------+----------------+--
-----+-------+---------+
2 rows in set (0.16 sec)
```

5.3.2.2.2. 数据仓库场景一：即席查询

本文通过示例为您介绍如何基于StarRocks的视图能力构建实时数仓-即席查询解决方案。

背景信息

StarRocks具体场景的描述，请参见数据仓库：即席查询场景。

前提条件

- 已在新版控制台创建DataFlow集群，具体操作请参见创建集群。
- 已创建StarRocks集群，具体操作请参见创建集群。
- 已创建RDS MySQL，具体操作请参见创建RDS MySQL实例。

> **说明** 本文示例中DataFlow集群为EMR-3.40.0版本、StarRocks集群为EMR-5.5.1版本，MySQL为5.7版本。

使用限制

- DataFlow集群、StarRocks集群和RDS MySQL实例需要在同一个VPC下，并且在同一个可用区下。
- DataFlow集群和StarRocks集群均须开启公网访问。
- DataFlow集群为EMR-4.40.0及以上版本。
- StarRocks集群为EMR-5.5.1及以上版本。
- RDS MySQL为5.7及以上版本。

注意事项

本文档仅供测试使用，生产级别的Flink作业请使用阿里云实时计算Flink版产品进行配置，或者使用YARN或者Kubernetes提交作业。

详情请参见Apache Hadoop YARN和Native Kubernetes。

操作流程

1. 步骤一：创建MySQL源数据表
2. 步骤二：创建StarRocks表
3. 步骤三：执行Flink任务，启动数据流
4. 步骤四：场景演示

步骤一：创建MySQL源数据表

1. 创建测试的数据库和账号，具体操作请参见创建数据库和账号。

   创建完数据库和账号后，需要授权测试账号的读写权限。

   > **说明** 本文示例中创建的数据库名称为flink_cdc，账号为emr_test。

2. 使用创建的测试账号连接MySQL实例，具体操作请参见通过DMS登录RDS MySQL。
3. 执行以下命令，创建数据库。

   ```
   CREATE DATABASE IF NOT EXISTS flink_cdc;
   ```

4. 执行以下命令，创建数据表orders。

```
create table flink_cdc.orders (
   order_id INT NOT NULL AUTO_INCREMENT,
   order_revenue FLOAT NOT NULL,
   order_region VARCHAR(40) NOT NULL,
   customer_id INT NOT NULL,
   PRIMARY KEY ( order_id )
);
```

5. 执行以下命令，创建数据表customers。

```
create table flink_cdc.customers (
   customer_id INT NOT NULL,
   customer_age INT NOT NULL,
   customer_name VARCHAR(40) NOT NULL,
   PRIMARY KEY ( customer_id )
);
```

步骤二：创建StarRocks表

1. 使用SSH方式登录StarRocks集群，详情请参见登录集群。
2. 执行以下，连接StarRocks集群。

```
mysql -h127.0.0.1 -P 9030 -uroot
```

3. 执行以下命令，创建数据库。

```
CREATE DATABASE IF NOT EXISTS `flink_cdc`;
```

4. 执行以下命令，创建数据表customers。

```
CREATE TABLE IF NOT EXISTS `flink_cdc`.`customers` (
  `customer_id` INT NOT NULL  COMMENT "",
  `customer_age` FLOAT NOT NULL  COMMENT "",
  `customer_name` STRING NOT NULL  COMMENT ""
) ENGINE=olap
PRIMARY KEY(`customer_id`)
COMMENT ""
DISTRIBUTED BY HASH(`customer_id`) BUCKETS 1
PROPERTIES (
  "replication_num" = "1"
);
```

5. 执行以下命令，创建数据表orders。

```
CREATE TABLE IF NOT EXISTS `flink_cdc`.`orders` (
  `order_id` INT NOT NULL  COMMENT "",
  `order_revenue` FLOAT NOT NULL  COMMENT "",
  `order_region` STRING NOT NULL  COMMENT "",
  `customer_id` INT NOT NULL  COMMENT ""
) ENGINE=olap
PRIMARY KEY(`order_id`)
COMMENT ""
DISTRIBUTED BY HASH(`order_id`) BUCKETS 1
PROPERTIES (
  "replication_num" = "1"
);
```

6. 执行以下命令，基于ODS表创建DWD视图。

```
CREATE VIEW flink_cdc.dwd_order_customer_valid (
  order_id,
  order_revenue,
  order_region,
  customer_id,
  customer_age,
  customer_name
)
AS
SELECT o.order_id, o.order_revenue, o.order_region, c.customer_id, c.customer_age, c.customer_name
FROM flink_cdc.customers c JOIN flink_cdc.orders o
ON c.customer_id=o.customer_id
WHERE c.customer_id != -1;
```

7. 执行以下命令，基于DWD表创建DWS视图。

```
CREATE VIEW flink_cdc.dws_agg_by_region (
  order_region,
  order_cnt,
  order_total_revenue)
AS
SELECT order_region, count(order_region), sum(order_revenue)
FROM flink_cdc.dwd_order_customer_valid
GROUP BY order_region;
```

步骤三：执行Flink任务，启动数据流

1. 下载Flink CDC connector和Flink StarRocks Connector，并上传到DataFlow集群的*/opt/apps/FLINK/flink-current/lib*目录下。
2. 使用SSH方式登录DataFlow集群，详情请参见登录集群。
3. 执行以下命令，启动集群。

   > **注意** 本文示例仅供测试，如果是生产级别的Flink作业请使用YARN或Kubernetes方式提交，详情请参见Apache Hadoop YARN和Native Kubernetes。

   ```
   /opt/apps/FLINK/flink-current/bin/start-cluster.sh
   ```

4. 添加端口配置。
   i. 执行以下命令，编辑文件*flink-conf.yaml*。

   ```
   vim /etc/taihao-apps/flink-conf/flink-conf.yaml
   ```

   ii. 添加以下内容至文件最后一行。

   ```
   rest.port: 8083
   ```

5. 编写Flink SQL作业，并保存为*demo.sql*。

   执行以下命令，编辑*demo.sql*文件。

   ```
   vim demo.sql
   ```

   文件内容如下所示。

   ```
   CREATE DATABASE IF NOT EXISTS `default_catalog`.`flink_cdc`;
   CREATE TABLE IF NOT EXISTS `default_catalog`.`flink_cdc`.`customers_src` (
     `customer_id` INT NOT NULL,
     `customer_age` FLOAT NOT NULL,
     `customer_name` STRING NOT NULL,
     PRIMARY KEY(`customer_id`)
    NOT ENFORCED
   ) with (
     'connector' = 'mysql-cdc',
     'hostname' = 'rm-2ze8398257383****.mysql.rds.aliyuncs.com',
     'port' = '3306',
     'username' = 'emr_test',
     'password' = 'Yz12****',
     'database-name' = 'flink_cdc',
     'table-name' = 'customers'
   );
   CREATE TABLE IF NOT EXISTS `default_catalog`.`flink_cdc`.`customers_sink` (
     `customer_id` INT NOT NULL,
     `customer_age` FLOAT NOT NULL,
     `customer_name` STRING NOT NULL,
     PRIMARY KEY(`customer_id`)
    NOT ENFORCED
   ) with (
     'load-url' = '10.0.**.**:8030',
     'database-name' = 'flink_cdc',
     'jdbc-url' = 'jdbc:mysql://10.0.**.**:9030',
     'sink.buffer-flush.interval-ms' = '15000',
     'sink.properties.format' = 'json',
     'username' = 'root',
     'table-name' = 'customers',
     'sink.properties.strip_outer_array' = 'true',
     'password' = '',
     'sink.max-retries' = '10',
     'connector' = 'starrocks'
   );
   ```

```
INSERT INTO `default_catalog`.`flink_cdc`.`customers_sink` SELECT * FROM `default_catalog`.`flink_cdc`.`customers_src`;
CREATE TABLE IF NOT EXISTS `default_catalog`.`flink_cdc`.`orders_src` (
  `order_id` INT NOT NULL,
  `order_revenue` FLOAT NOT NULL,
  `order_region` STRING NOT NULL,
  `customer_id` INT NOT NULL,
  PRIMARY KEY(`order_id`)
 NOT ENFORCED
) with (
  'database-name' = 'flink_cdc',
  'table-name' = 'orders',
  'connector' = 'mysql-cdc',
  'hostname' = 'rm-2ze8398257383****.mysql.rds.aliyuncs.com',
  'port' = '3306',
  'username' = 'emr_test',
  'password' = 'Yz12****'
);
CREATE TABLE IF NOT EXISTS `default_catalog`.`flink_cdc`.`orders_sink` (
  `order_id` INT NOT NULL,
  `order_revenue` FLOAT NOT NULL,
  `order_region` STRING NOT NULL,
  `customer_id` INT NOT NULL,
  PRIMARY KEY(`order_id`)
 NOT ENFORCED
) with (
  'sink.properties.strip_outer_array' = 'true',
  'password' = '',
  'sink.max-retries' = '10',
  'connector' = 'starrocks',
  'table-name' = 'orders',
  'jdbc-url' = 'jdbc:mysql://10.0.**.**:9030',
  'sink.buffer-flush.interval-ms' = '15000',
  'sink.properties.format' = 'json',
  'username' = 'root',
  'load-url' = '10.0.**.**:8030',
  'database-name' = 'flink_cdc'
);
INSERT INTO `default_catalog`.`flink_cdc`.`orders_sink` SELECT * FROM `default_catalog`.`flink_cdc`.`orders_src`;
```

涉及参数如下所示：

- 创建数据表customers_src。

参数	描述
connector	固定值为mysql-cdc。
hostname	RDS的内网地址。 您可以在RDS的数据库连接页面，单击内网地址进行复制。例如，rm-2ze8398257383****.mysql.rds.aliyuncs.com。
port	固定值为3306。
username	步骤一：创建MySQL源数据表中创建的账号名。本示例为emr_test。
password	步骤一：创建MySQL源数据表中创建的账号的密码。本示例为Yz12****。
database-name	步骤一：创建MySQL源数据表中创建的数据库名。本示例为flink_cdc。
table-name	步骤一：创建MySQL源数据表中创建的数据表。本示例为customers。

- 创建数据表customers_sink和orders_sink。

参数	描述
load-url	指定FE的IP地址和HTTP端口，格式为 `StarRocks集群的内网IP地址:8030` 。例如，10.0.**.**:8030。
database-name	步骤一：创建MySQL源数据表中创建的数据库名。本示例为flink_cdc。

参数	描述
jdbc-url	用于在StarRocks中执行查询操作。 例如，jdbc:mysql://10.0.**.**:9030。其中，10.0.**.**为StarRocks集群的内网IP地址。
username	StarRocks连接用户名。固定为root。
table-name	本示例固定值为customers。
connector	固定值为starrocks。

6. 执行以下命令，启动Flink任务。

```
/opt/apps/FLINK/flink-current/bin/sql-client.sh -f demo.sql
```

步骤四：场景演示

1. 使用步骤一：创建MySQL源数据表中创建的测试账号连接MySQL实例，具体操作请参见通过DMS登录RDS MySQL。
2. 在RDS数据库窗口执行以下命令，向表orders和customers中插入数据。

```
INSERT INTO flink_cdc.orders(order_id,order_revenue,order_region,customer_id) VALUES(1,10,"beijing",1);
INSERT INTO flink_cdc.orders(order_id,order_revenue,order_region,customer_id) VALUES(2,10,"beijing",1);
INSERT INTO flink_cdc.customers(customer_id,customer_age,customer_name) VALUES(1, 22, "emr_test");
```

3. 使用SSH方式登录StarRocks集群，详情请参见登录集群。
4. 执行以下，连接StarRocks集群。

```
mysql -h127.0.0.1 -P 9030 -uroot
```

5. 执行以下命令，查询ODS层数据。
   i. 执行以下命令，使用数据库。

```
use flink_cdc;
```

   ii. 执行以下命令，查看orders表信息。

```
select * from orders;
```

   返回信息如下所示。

```
+----------+---------------+--------------+-------------+
| order_id | order_revenue | order_region | customer_id |
+----------+---------------+--------------+-------------+
|        1 |            10 | beijing      |           1 |
|        2 |            10 | beijing      |           1 |
+----------+---------------+--------------+-------------+
```

   iii. 执行以下命令，查看customers表信息。

```
select * from customers;
```

   返回信息如下所示。

```
+-------------+--------------+---------------+
| customer_id | customer_age | customer_name |
+-------------+--------------+---------------+
|           1 |           22 | emr_test      |
+-------------+--------------+---------------+
```

6. 执行以下命令，查询DWD层数据。
   i. 执行以下命令，使用数据库。

```
use flink_cdc;
```

ii. 执行以下命令，查看orders表信息。

```
select * from dwd_order_customer_valid;
```

返回信息如下所示。

```
+----------+---------------+--------------+-------------+--------------+---------------+
| order_id | order_revenue | order_region | customer_id | customer_age | customer_name |
+----------+---------------+--------------+-------------+--------------+---------------+
|        1 |            10 | beijing      |           1 |           22 | emr_test      |
|        2 |            10 | beijing      |           1 |           22 | emr_test      |
+----------+---------------+--------------+-------------+--------------+---------------+
2 rows in set (0.00 sec)
```

7. 执行以下命令，查询DWS层数据。

i. 执行以下命令，使用数据库。

```
use flink_cdc;
```

ii. 执行以下命令，查看orders表信息。

```
select * from dws_agg_by_region;
```

返回信息如下所示。

```
+--------------+-----------+---------------------+
| order_region | order_cnt | order_total_revenue |
+--------------+-----------+---------------------+
| beijing      |         2 |                  20 |
+--------------+-----------+---------------------+
1 row in set (0.01 sec)
```

5.3.2.2.3. 数据仓库场景二：分钟级准实时数仓

本文通过示例为您介绍如何基于StarRocks构建分钟级准实时数仓。

背景信息

分钟级准实时数仓场景的详细信息，请参见数据仓库：分钟级准实时场景。

前提条件

- 已在新版控制台创建DataFlow集群，具体操作请参见创建集群。
- 已创建StarRocks集群，具体操作请参见创建集群。
- 已创建RDS MySQL，具体操作请参见创建RDS MySQL实例。
- 已创建EMR Studio集群，并开通了8443、8000和8081端口，具体操作请参见创建EMR Studio集群和添加安全组规则。

> **说明** 本文示例中DataFlow集群为EMR-3.40.0版本、StarRocks集群为EMR-5.6.0版本，MySQL为5.7版本。

使用限制

- DataFlow集群、StarRocks集群和RDS MySQL实例需要在同一个VPC下，并且在同一个可用区下。
- DataFlow集群和StarRocks集群均须开启公网访问。
- RDS MySQL为5.7及以上版本。

操作流程

1. 步骤一：创建MySQL源数据表
2. 步骤二：创建StarRocks表
3. 步骤三：同步RDS中的源数据到StarRocks的ODS表
4. 步骤四：通过任务调度器，编排各数据层的微批同步任务
5. 步骤五：查看数据库和表信息
6. 步骤六：场景演示，查询插入后的数据

步骤一：创建MySQL源数据表

1. 创建测试的数据库和账号，具体操作请参见创建数据库和账号。

   创建完数据库和账号后，需要授权测试账号的读写权限。

> **说明** 本文示例中创建的数据库名称为flink_cdc，账号为emr_test。

2. 使用创建的测试账号连接MySQL实例，具体操作请参见通过DMS登录RDS MySQL。
3. 执行以下命令，创建数据库。

```
CREATE DATABASE IF NOT EXISTS flink_cdc;
```

4. 执行以下命令，创建数据表orders和customers。
   - 创建orders表。

```
CREATE TABLE flink_cdc.orders (
   order_id INT NOT NULL AUTO_INCREMENT,
   order_revenue FLOAT NOT NULL,
   order_region VARCHAR(40) NOT NULL,
   customer_id INT NOT NULL,
   PRIMARY KEY ( order_id )
);
```

   - 创建customers表。

```
CREATE TABLE flink_cdc.customers (
   customer_id INT NOT NULL,
   customer_age INT NOT NULL,
   customer_name VARCHAR(40) NOT NULL,
   PRIMARY KEY ( customer_id )
);
```

步骤二：创建StarRocks表

1. 使用SSH方式登录StarRocks集群，具体操作请参见登录集群。
2. 执行以下，连接StarRocks集群。

```
mysql -h127.0.0.1 -P 9030 -uroot
```

3. 执行以下命令，创建数据库。

```
CREATE DATABASE IF NOT EXISTS `flink_cdc`;
```

4. 执行以下命令，创建数据表customers和orders。
   - 创建customers表

```
CREATE TABLE IF NOT EXISTS `flink_cdc`.`customers` (
  `timestamp` DateTime NOT NULL COMMENT "",
  `customer_id` INT NOT NULL  COMMENT "",
  `customer_age` FLOAT NOT NULL  COMMENT "",
  `customer_name` STRING NOT NULL  COMMENT ""
) ENGINE=olap
PRIMARY KEY(`timestamp`, `customer_id`)
COMMENT ""
DISTRIBUTED BY HASH(`customer_id`) BUCKETS 1
PROPERTIES (
  "replication_num" = "1"
);
```

   - 创建orders表

```
CREATE TABLE IF NOT EXISTS `flink_cdc`.`orders` (
  `timestamp` DateTime NOT NULL COMMENT "",
  `order_id` INT NOT NULL  COMMENT "",
  `order_revenue` FLOAT NOT NULL  COMMENT "",
  `order_region` STRING NOT NULL  COMMENT "",
  `customer_id` INT NOT NULL  COMMENT ""
) ENGINE=olap
PRIMARY KEY(`timestamp`, `order_id`)
COMMENT ""
DISTRIBUTED BY HASH(`order_id`) BUCKETS 1
PROPERTIES (
  "replication_num" = "1"
);
```

5. 执行以下命令，创建DWD表。

```
CREATE TABLE IF NOT EXISTS `flink_cdc`.`dwd_order_customer_valid`(
  `timestamp` DateTime NOT NULL COMMENT "",
  `order_id` INT NOT NULL  COMMENT "",
  `order_revenue` FLOAT NOT NULL  COMMENT "",
  `order_region` STRING NOT NULL  COMMENT "",
  `customer_id` INT NOT NULL  COMMENT "",
  `customer_age` FLOAT NOT NULL  COMMENT "",
  `customer_name` STRING NOT NULL  COMMENT ""
) ENGINE=olap
PRIMARY KEY(`timestamp`, `order_id`)
COMMENT ""
DISTRIBUTED BY HASH(`order_id`) BUCKETS 1
PROPERTIES (
  "replication_num" = "1"
);
```

6. 执行以下命令，创建DWS表。

```
CREATE TABLE IF NOT EXISTS `flink_cdc`.`dws_agg_by_region` (
  `timestamp` DateTime NOT NULL COMMENT "",
  `order_region` STRING NOT NULL  COMMENT "",
  `order_cnt` INT NOT NULL  COMMENT "",
  `order_total_revenue` INT NOT NULL  COMMENT ""
) ENGINE=olap
PRIMARY KEY(`timestamp`, `order_region`)
COMMENT ""
DISTRIBUTED BY HASH(`order_region`) BUCKETS 1
PROPERTIES (
  "replication_num" = "1"
);
```

步骤三：同步RDS中的源数据到StarRocks的ODS表

1. 下载Flink CDC connector和Flink StarRocks Connector，并上传至DataFlow集群的*/opt/apps/FLINK/flink-current/lib*目录下。
2. 拷贝DataFlow集群的*/opt/apps/FLINK/flink-current/opt/connectors/kafka*目录下的JAR包至*/opt/apps/FLINK/flink-current/lib*目录下。
3. 使用SSH方式登录DataFlow集群，具体操作请参见登录集群。
4. 执行以下命令，启动集群。

> 注意　本文示例仅供测试，如果是生产级别的Flink作业请使用YARN或Kubernetes方式提交，详情请参见Apache Hadoop YARN和Native Kubernetes。

```
/opt/apps/FLINK/flink-current/bin/start-cluster.sh
```

5. 编写Flink SQL作业，并保存为*demo.sql*。

   执行以下命令，编辑*demo.sql*文件。

```
vim demo.sql
```

   文件内容如下所示。

```
CREATE DATABASE IF NOT EXISTS `default_catalog`.`flink_cdc`;
-- create source tables
CREATE TABLE IF NOT EXISTS `default_catalog`.`flink_cdc`.`orders_src`(
  `order_id` INT NOT NULL,
  `order_revenue` FLOAT NOT NULL,
  `order_region` STRING NOT NULL,
  `customer_id` INT NOT NULL,
  PRIMARY KEY(`order_id`) NOT ENFORCED
) with (
  'connector' = 'mysql-cdc',
  'hostname' = 'rm-2ze5h9qnki343****.mysql.rds.aliyuncs.com',
  'port' = '3306',
  'username' = 'emr_test',
  'password' = 'Yz12****',
  'database-name' = 'flink_cdc',
  'table-name' = 'orders'
);
CREATE TABLE IF NOT EXISTS `default_catalog`.`flink_cdc`.`customers_src` (
  `customer_id` INT NOT NULL,
  `customer_age` FLOAT NOT NULL,
```

```
  `customer_name` STRING NOT NULL,
  PRIMARY KEY(`customer_id`) NOT ENFORCED
) with (
  'connector' = 'mysql-cdc',
  'hostname' = 'rm-2ze5h9qnki343****.mysql.rds.aliyuncs.com',
  'port' = '3306',
  'username' = 'emr_test',
  'password' = 'Yz12****',
  'database-name' = 'flink_cdc',
  'table-name' = 'customers'
);
CREATE TABLE IF NOT EXISTS `default_catalog`.`flink_cdc`.`orders_sink` (
  `timestamp` TIMESTAMP NOT NULL,
  `order_id` INT NOT NULL,
  `order_revenue` FLOAT NOT NULL,
  `order_region` STRING NOT NULL,
  `customer_id` INT NOT NULL,
  PRIMARY KEY(`timestamp`,`order_id`)
 NOT ENFORCED
) with (
  'connector' = 'starrocks',
  'database-name' = 'flink_cdc',
  'table-name' = 'orders',
  'username' = 'root',
  'password' = '',
  'jdbc-url' = 'jdbc:mysql://192.168.**.**:9030',
  'load-url' = '192.168.**.**:8030',
  'sink.properties.format' = 'json',
  'sink.properties.strip_outer_array' = 'true',
  'sink.buffer-flush.interval-ms' = '15000'
);
CREATE TABLE IF NOT EXISTS `default_catalog`.`flink_cdc`.`customers_sink` (
  `timestamp` TIMESTAMP NOT NULL,
  `customer_id` INT NOT NULL,
  `customer_age` FLOAT NOT NULL,
  `customer_name` STRING NOT NULL,
  PRIMARY KEY(`timestamp`,`customer_id`)
 NOT ENFORCED
) with (
  'connector' = 'starrocks',
  'database-name' = 'flink_cdc',
  'table-name' = 'customers',
  'username' = 'root',
  'password' = '',
  'jdbc-url' = 'jdbc:mysql://192.168.**.**:9030',
  'load-url' = '192.168.**.**:8030',
  'sink.properties.format' = 'json',
  'sink.properties.strip_outer_array' = 'true',
  'sink.buffer-flush.interval-ms' = '15000'
);
BEGIN STATEMENT SET;
INSERT INTO `default_catalog`.`flink_cdc`.`orders_sink`
SELECT
  LOCALTIMESTAMP,
  order_id,
  order_revenue,
  order_region,
  customer_id
FROM `default_catalog`.`flink_cdc`.`orders_src`;
INSERT INTO `default_catalog`.`flink_cdc`.`customers_sink`
SELECT
  LOCALTIMESTAMP,
  customer_id,
  customer_age,
  customer_name
FROM `default_catalog`.`flink_cdc`.`customers_src`;
END;
```

涉及参数如下所示：

- 创建数据表orders_src和customers_src。

参数	描述
connector	固定值为mysql-cdc。
hostname	RDS的内网地址。 您可以在RDS的数据库连接页面，单击内网地址进行复制。例如，rm-2ze5h9qnki343****.mysql.rds.aliyuncs.com。
port	固定值为3306。
username	步骤一：创建MySQL源数据表中创建的账号名。本示例为emr_test。
password	步骤一：创建MySQL源数据表中创建的账号的密码。本示例为Yz12****。
database-name	步骤一：创建MySQL源数据表中创建的数据库名。本示例为flink_cdc。
table-name	步骤一：创建MySQL源数据表中创建的数据表。 ■ orders_src：本示例为orders。 ■ customers_src：本示例为customers。

- 创建数据表orders_sink和customers_sink。

参数	描述
connector	固定值为starrocks。
database-name	步骤一：创建MySQL源数据表中创建的数据库名。本示例为flink_cdc。
table-name	步骤一：创建MySQL源数据表中创建的数据表。 ■ orders_sink：本示例为orders。 ■ customers_sink：本示例为customers。
username	StarRocks连接用户名。固定值为root。
password	不填写。
jdbc-url	用于在StarRocks中执行查询操作。 例如，jdbc:mysql://10.0.**.**:9030。其中，10.0.**.**为StarRocks集群的内网IP地址。
load-url	指定FE的IP地址和HTTP端口，格式为 `StarRocks集群的内网IP地址:8030` 。例如，10.0.**.**:8030。

6. 执行以下命令，启动Flink任务。

```
/opt/apps/FLINK/flink-current/bin/sql-client.sh -f demo.sql
```

步骤四：通过任务调度器，编排各数据层的微批同步任务

将以下两个Job以10分钟为一次间隔，编排成定时任务。

- Job 1

```
-- ODS to DWD
INSERT INTO dwd_order_customer_valid
SELECT
'{start_time}',
o.order_id,
o.order_revenue,
o.order_region,
c.customer_id,
c.customer_age,
c.customer_name
FROM customers c JOIN orders o ON c.customer_id=o.customer_id
WHERE o.timestamp >= '{start_time}' AND o.timestamp < DATE_ADD('{start_time}',INTERVAL '{interval_time}' MINUTE) AND
c.timestamp >= '{start_time}' AND c.timestamp < DATE_ADD('{start_time}',INTERVAL '{interval_time}' MINUTE)
```

- Job 2

```
-- DWD to DWS
INSERT INTO dws_agg_by_region
SELECT
'{start_time}',
order_region,
count(*) AS order_cnt,
sum(order_revenue) AS order_total_revenue
FROM dwd_order_customer_valid
WHERE timestamp >= '{start_time}' AND timestamp < DATE_ADD('{start_time}',INTERVAL '{interval_time}' MINUTE)
GROUP BY timestamp, order_region;
```

本示例使用EMR Studio作为任务调度器，您也可以使用自己的任务编排方案。

1. 为EMR Studio集群添加用户，详情请参见添加用户。
2. 为添加的用户授权。
   i. 使用SSH方式登录EMR Studio集群，具体操作请参见登录集群。
   ii. 执行以下命令，授权添加的用户为AirFlow的Admin Role。

   ```
   source /usr/lib/airflow-current/bin/activate
   airflow users add-role -r Admin -u <user>
   ```

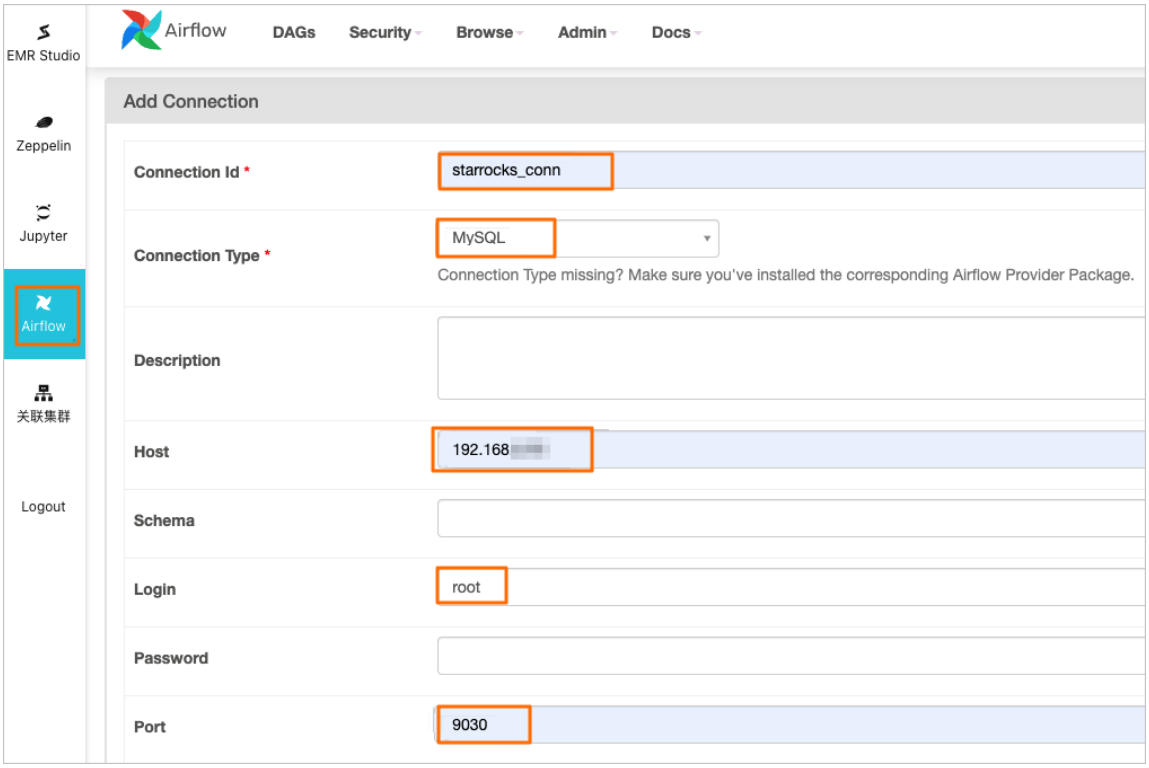
   > **说明** 示例中的 `<user>` 为您上一步骤中添加的用户名称。

3. 进入数据开发控制台。

   在EMR Studio集群的**访问链接与端口**页面，单击**Studio Workspace UI**所在行的链接。

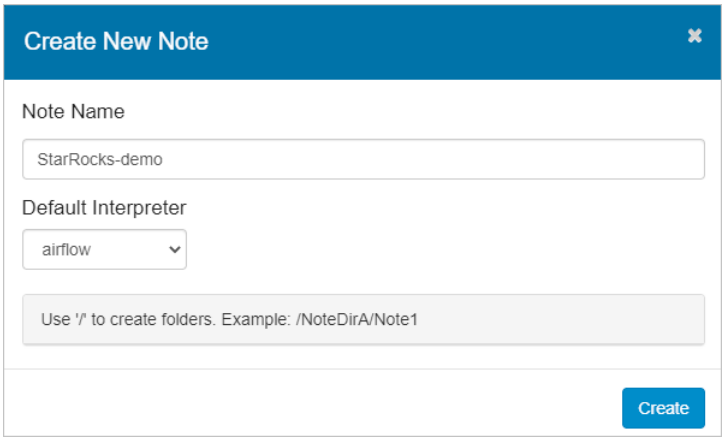
   输入步骤1中添加的用户名和密码，即可正常访问Web UI页面。
4. 创建Airflow的Connection。
   i. 在左侧导航栏中，单击**Airflow**。
   ii. 在**Airflow**页面，选择上方的**Admin > Connections**。
   iii. 单击 图标。

iv. 在Add Connection页面，配置相关参数。



参数	描述
Connection Id	Connection名称，您可以自定义。本示例为starrocks_conn。
Connection Type	选择**MySQL**。
Host	StarRocks集群的内网IP地址。
Login	固定值为root。
Port	固定值为9030。

5. 创建Zeppelin的Note。
   i. 在左侧导航栏中，单击**Zeppelin**。
   ii. 在**Zeppelin**页面，单击**Create New Note**。
   iii. 在**Create New Note**对话框中，输入Note Name，在**Default Interpreter**下拉框中，选择**airflow**。

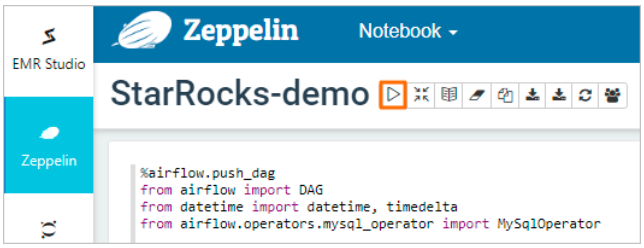


iv. 编写Airflow DAG脚本。

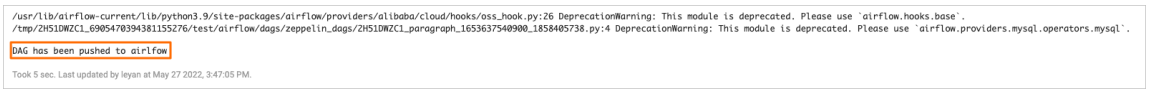
配置starrocks_demo_dag脚本，示例代码如下。

```
%airflow.push_dag
from airflow import DAG
from datetime import datetime, timedelta
from airflow.operators.mysql_operator import MySqlOperator
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime.now(),
    'retries': 0,
    'retry_delay': timedelta(minutes=5),
}
with DAG('starrocks_demo_dag',
         schedule_interval='*/10 * * * *',
         default_args=default_args) as dag:
    execution_time = "{{ ts }}"
    ods_to_dwd_sql = """
    INSERT INTO dwd_order_customer_valid
    SELECT
    '{start_time}',
    o.order_id,
    o.order_revenue,
    o.order_region,
    c.customer_id,
    c.customer_age,
    c.customer_name
    FROM customers c JOIN orders o ON c.customer_id=o.customer_id
    WHERE o.timestamp >= '{start_time}' AND o.timestamp < DATE_ADD('{start_time}',INTERVAL '{interval_time}' MINUTE
) AND
    c.timestamp >= '{start_time}' AND c.timestamp < DATE_ADD('{start_time}',INTERVAL '{interval_time}' MINUTE)
    """.format(start_time=execution_time, interval_time=10)
    dwd_to_dws_sql = """
    INSERT INTO dws_agg_by_region
    SELECT
    '{start_time}',
    order_region,
    count(*) AS order_cnt,
    sum(order_revenue) AS order_total_revenue
    FROM dwd_order_customer_valid
    WHERE timestamp >= '{start_time}' AND timestamp < DATE_ADD('{start_time}',INTERVAL '{interval_time}' MINUTE)
    GROUP BY timestamp, order_region;
    """.format(start_time=execution_time, interval_time=10)
    ods_to_dwd = MySqlOperator(
        task_id='ods_to_dwd',
        sql=ods_to_dwd_sql,
        mysql_conn_id='starrocks_conn',
        autocommit=True
    )
    dwd_to_dws = MySqlOperator(
        task_id='dwd_to_dws',
        sql=dwd_to_dws_sql,
        mysql_conn_id='starrocks_conn',
        autocommit=True
    )
    ods_to_dwd >> dwd_to_dws
```

v. 单击▷图标，运行脚本。

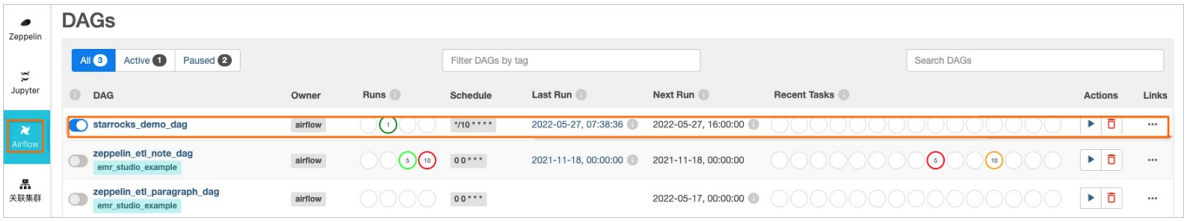


执行成功后，Paragraph输出以下提示信息。



6. 查看DAG状态。

在Airflow页面即可看到starrocks_demo_dag的运行情况。



步骤五：查看数据库和表信息

1. 使用SSH方式登录StarRocks集群，具体操作请参见登录集群。
2. 执行以下，连接StarRocks集群。

```
mysql -h127.0.0.1 -P 9030 -uroot
```

3. 执行以下命令，查询数据库信息。
   i. 执行以下命令，使用数据库。

```
use flink_cdc;
```

   ii. 执行以下命令，查看表信息。

```
show tables;
```

   返回信息如下所示。

```
+--------------------------+
| Tables_in_flink_cdc      |
+--------------------------+
| customers                |
| dwd_order_customer_valid |
| dws_agg_by_region        |
| orders                   |
+--------------------------+
4 rows in set (0.01 sec)
```

步骤六：场景演示，查询插入后的数据

1. 使用步骤一：创建MySQL源数据表中创建的测试账号连接MySQL实例，具体操作请参见通过DMS登录RDS MySQL。
2. 在RDS数据库窗口执行以下命令，向表orders和customers中插入数据。

```
INSERT INTO flink_cdc.orders(order_id,order_revenue,order_region,customer_id) VALUES(1,10,"beijing",1);
INSERT INTO flink_cdc.orders(order_id,order_revenue,order_region,customer_id) VALUES(2,10,"beijing",1);
INSERT INTO flink_cdc.customers(customer_id,customer_age,customer_name) VALUES(1, 22, "emr_test");
```

3. 使用SSH方式登录StarRocks集群，具体操作请参见登录集群。
4. 执行以下命令，连接StarRocks集群。

```
mysql -h127.0.0.1 -P 9030 -uroot
```

5. 执行以下命令，查询ODS层数据。

i. 执行以下命令，使用数据库。

```
use flink_cdc;
```

ii. 执行以下命令，查看orders和customers表信息。

- 查看orders表

```
select * from orders;
```

返回信息如下所示。

```
+----------------------------+----------+--------------+-------------+-------------+
| timestamp                  | order_id | order_revenue | order_region | customer_id |
+----------------------------+----------+--------------+-------------+-------------+
| 2022-05-27 13:39:50.098000 |        1 |            10 | beijing      |           1 |
| 2022-05-27 13:39:50.596000 |        2 |            10 | beijing      |           1 |
+----------------------------+----------+--------------+-------------+-------------+
2 rows in set (0.00 sec)
```

- 查看customers表

```
select * from customers;
```

返回信息如下所示。

```
+----------------------------+-------------+--------------+---------------+
| timestamp                  | customer_id | customer_age | customer_name |
+----------------------------+-------------+--------------+---------------+
| 2022-05-27 13:40:11.005000 |           1 |           22 | emr_test      |
+----------------------------+-------------+--------------+---------------+
1 row in set (0.01 sec)
```

6. 执行以下命令，查询DWD层数据。

i. 执行以下命令，使用数据库。

```
use flink_cdc;
```

ii. 执行以下命令，查看dwd_order_customer_valid表信息。

```
select * from dwd_order_customer_valid;
```

返回信息如下所示。

```
+---------------------+----------+--------------+-------------+-------------+--------------+---------------+
| timestamp           | order_id | order_revenue | order_region | customer_id | customer_age | customer_name |
+---------------------+----------+--------------+-------------+-------------+--------------+---------------+
| 2022-05-27 13:35:00 |        1 |            10 | beijing      |           1 |           22 | emr_test      |
| 2022-05-27 13:35:00 |        2 |            10 | beijing      |           1 |           22 | emr_test      |
+---------------------+----------+--------------+-------------+-------------+--------------+---------------+
2 rows in set (0.01 sec)
```

7. 执行以下命令，查询DWS层数据。

i. 执行以下命令，使用数据库。

```
use flink_cdc;
```

ii. 执行以下命令，查看dws_agg_by_region表信息。

```
select * from dws_agg_by_region;
```

返回信息如下所示。

```
+---------------------+--------------+-----------+---------------------+
| timestamp           | order_region | order_cnt | order_total_revenue |
+---------------------+--------------+-----------+---------------------+
| 2022-05-27 13:35:00 | beijing      |         2 |                  20 |
+---------------------+--------------+-----------+---------------------+
1 row in set (0.00 sec)
```

您也可以执行以下命令，查询部分字段信息。

```
select order_region, sum(order_cnt),sum(order_total_revenue)  from dws_agg_by_region group by order_region;
```

返回信息如下所示。

```
+--------------+------------------+---------------------------+
| order_region | sum(`order_cnt`) | sum(`order_total_revenue`) |
+--------------+------------------+---------------------------+
| beijing      |                2 |                        20 |
+--------------+------------------+---------------------------+
1 row in set (0.05 sec)
```

5.3.2.2.4. 数据仓库场景三：增量数据实时统计

本文通过示例为您介绍如何基于StarRocks构建实时数仓-增量数据实时统计。

背景信息

增量数据实时统计场景的详细信息，请参见数据仓库：增量数据实时统计场景。

前提条件

- 已在新版控制台创建DataFlow集群，具体操作请参见创建集群。
- 已创建StarRocks集群，具体操作请参见创建集群。
- 已创建RDS MySQL，具体操作请参见创建RDS MySQL实例。

> **说明** 本文示例中DataFlow集群为EMR-3.40.0版本、StarRocks集群为EMR-5.6.0版本，MySQL为5.7版本。

使用限制

- DataFlow集群、StarRocks集群和RDS MySQL实例需要在同一个VPC下，并且在同一个可用区下。
- DataFlow集群和StarRocks集群均须开启公网访问。
- RDS MySQL为5.7及以上版本。

操作流程

1. 步骤一：创建MySQL源数据表
2. 步骤二：创建Kafka的Topic
3. 步骤三：创建StarRocks表和导入任务
4. 步骤四：执行Flink任务，启动数据流
5. 步骤五：查看数据库和表信息
6. 步骤六：场景演示，查询插入后的数据

步骤一：创建MySQL源数据表

1. 创建测试的数据库和账号，具体操作请参见创建数据库和账号。
   创建完数据库和账号后，需要授权测试账号的读写权限。

   > **说明** 本文示例中创建的数据库名称为flink_cdc，账号为emr_test。

2. 使用创建的测试账号连接MySQL实例，具体操作请参见通过DMS登录RDS MySQL。
3. 执行以下命令，创建数据库。

```
CREATE DATABASE IF NOT EXISTS flink_cdc;
```

4. 执行以下命令，创建数据表orders和customers。
   - 创建orders表。

```
CREATE TABLE flink_cdc.orders (
   order_id INT NOT NULL AUTO_INCREMENT,
   order_revenue FLOAT NOT NULL,
   order_region VARCHAR(40) NOT NULL,
   customer_id INT NOT NULL,
   PRIMARY KEY ( order_id )
);
```

   - 创建customers表。

```
CREATE TABLE flink_cdc.customers (
   customer_id INT NOT NULL,
   customer_age INT NOT NULL,
   customer_name VARCHAR(40) NOT NULL,
   PRIMARY KEY ( customer_id )
);
```

步骤二：创建Kafka的Topic

1. 使用SSH方式登录DataFlow集群，具体操作请参见登录集群。
2. 执行以下命令， 进入Kafka的*bin*目录。

```
cd /opt/apps/FLINK/flink-current/bin
```

3. 执行以下命令，创建对应的Topic。

```
kafka-topics.sh --create --topic ods_order --replication-factor 1 --partitions 1 --bootstrap-server "192.168.**.**:9092
,192.168.**.**:9092,192.168.**.**:9092"
kafka-topics.sh --create --topic ods_customers --replication-factor 1 --partitions 1 --bootstrap-server "192.168.**.**:
9092,192.168.**.**:9092,192.168.**.**:9092"
kafka-topics.sh --create --topic dwd_order_customer_valid --replication-factor 1 --partitions 1 --bootstrap-server "192
.168.**.**:9092,192.168.**.**:9092,192.168.**.**:9092"
kafka-topics.sh --create --topic dws_agg_by_region --replication-factor 1 --partitions 1 --bootstrap-server "192.168.**
.**:9092,192.168.**.**:9092,192.168.**.**:9092"
```

> **说明** 本文代码示例中的 `192.168.**.**` 为DataFlow集群的内网IP地址，您可以在E-MapReduce控制台的**节点管理**页签查看。

步骤三：创建StarRocks表和导入任务

1. 使用SSH方式登录StarRocks集群，具体操作请参见登录集群。
2. 执行以下，连接StarRocks集群。

```
mysql -h127.0.0.1 -P 9030 -uroot
```

3. 执行以下命令，创建数据库。

```
CREATE DATABASE IF NOT EXISTS `flink_cdc`;
```

4. 执行以下命令，创建数据表customers和orders。
   - 创建customers表

```
CREATE TABLE IF NOT EXISTS `flink_cdc`.`customers` (
  `customer_id` INT NOT NULL  COMMENT "",
  `customer_age` FLOAT NOT NULL  COMMENT "",
  `customer_name` STRING NOT NULL  COMMENT ""
) ENGINE=olap
PRIMARY KEY(`customer_id`)
COMMENT ""
DISTRIBUTED BY HASH(`customer_id`) BUCKETS 1
PROPERTIES (
  "replication_num" = "1"
);
```

   - 创建orders表

```
CREATE TABLE IF NOT EXISTS `flink_cdc`.`orders` (
  `order_id` INT NOT NULL  COMMENT "",
  `order_revenue` FLOAT NOT NULL  COMMENT "",
  `order_region` STRING NOT NULL  COMMENT "",
  `customer_id` INT NOT NULL  COMMENT ""
) ENGINE=olap
PRIMARY KEY(`order_id`)
COMMENT ""
DISTRIBUTED BY HASH(`order_id`) BUCKETS 1
PROPERTIES (
  "replication_num" = "1"
);
```

5. 执行以下命令，创建DWD表。

```
CREATE TABLE IF NOT EXISTS `flink_cdc`.`dwd_order_customer_valid`(
  `order_id` INT NOT NULL  COMMENT "",
  `order_revenue` FLOAT NOT NULL  COMMENT "",
  `order_region` STRING NOT NULL  COMMENT "",
  `customer_id` INT NOT NULL  COMMENT "",
  `customer_age` FLOAT NOT NULL  COMMENT "",
  `customer_name` STRING NOT NULL  COMMENT ""
) ENGINE=olap
PRIMARY KEY(`order_id`)
COMMENT ""
DISTRIBUTED BY HASH(`order_id`) BUCKETS 1
PROPERTIES (
  "replication_num" = "1"
);
```

6. 执行以下命令，创建DWS表。

```
CREATE TABLE IF NOT EXISTS `flink_cdc`.`dws_agg_by_region` (
  `order_region` STRING NOT NULL  COMMENT "",
  `order_cnt` INT NOT NULL  COMMENT "",
  `order_total_revenue` INT NOT NULL  COMMENT ""
) ENGINE=olap
PRIMARY KEY(`order_region`)
COMMENT ""
DISTRIBUTED BY HASH(`order_region`) BUCKETS 1
PROPERTIES (
  "replication_num" = "1"
);
```

7. 执行以下命令，创建Routine Load导入任务，订阅Kafka数据源的数据。

```
CREATE ROUTINE LOAD flink_cdc.routine_load_orders ON orders
COLUMNS (order_id, order_revenue, order_region, customer_id)
PROPERTIES
(
  "format" = "json",
  "jsonpaths" = "[\"$.order_id\",\"$.order_revenue\",\"$.order_region\",\"$.customer_id\"]"
)
FROM KAFKA
(
  "kafka_broker_list" = "192.168.**.**:9092,192.168.**.**:9092,192.168.**.**:9092",
  "kafka_topic" = "ods_order"
);
CREATE ROUTINE LOAD flink_cdc.routine_load_customers ON customers
COLUMNS (customer_id, customer_age, customer_name)
PROPERTIES
(
    "format" = "json",
    "jsonpaths" = "[\"$.customer_id\",\"$.customer_age\",\"$.customer_name\"]"
)
FROM KAFKA
(
  "kafka_broker_list" = "192.168.**.**:9092,192.168.**.**:9092,192.168.**.**:9092",
  "kafka_topic" = "ods_customers"
);
CREATE ROUTINE LOAD flink_cdc.routine_load_dwd_order_customer_valid ON dwd_order_customer_valid
COLUMNS (order_id, order_revenue, order_region, customer_id, customer_age, customer_name)
PROPERTIES
(
    "format" = "json",
    "jsonpaths" = "[\"$.order_id\",\"$.order_revenue\",\"$.order_region\",\"$.customer_id\",\"$.customer_age\",\"$.cust
omer_name\"]"
)
FROM KAFKA
(
  "kafka_broker_list" = "192.168.**.**:9092,192.168.**.**:9092,192.168.**.**:9092",
  "kafka_topic" = "dwd_order_customer_valid"
);
CREATE ROUTINE LOAD flink_cdc.routine_load_dws_agg_by_region ON dws_agg_by_region
COLUMNS (order_region, order_cnt, order_total_revenue)
PROPERTIES
(
    "format" = "json",
    "jsonpaths" = "[\"$.order_region\",\"$.order_cnt\",\"$.order_total_revenue\"]"
)
FROM KAFKA
(
  "kafka_broker_list" = "192.168.**.**:9092,192.168.**.**:9092,192.168.**.**:9092",
  "kafka_topic" = "dws_agg_by_region"
);
```

步骤四：执行Flink任务，启动数据流

1. 下载Flink CDC connector和Flink StarRocks Connector，并上传至DataFlow集群的*/opt/apps/FLINK/flink-current/lib*目录下。
2. 拷贝DataFlow集群的*/opt/apps/FLINK/flink-current/opt/connectors/kafka*目录下的JAR包至*/opt/apps/FLINK/flink-current/lib*目录下。
3. 使用SSH方式登录DataFlow集群，具体操作请参见登录集群。
4. 执行以下命令，启动集群。

   > **注意** 本文示例仅供测试，如果是生产级别的Flink作业请使用YARN或Kubernetes方式提交，详情请参见Apache Hadoop YARN和Native Kubernetes。

   ```
   /opt/apps/FLINK/flink-current/bin/start-cluster.sh
   ```

5. 编写Flink SQL作业，并保存为*demo.sql*。

   执行以下命令，编辑*demo.sql*文件。

   ```
   vim demo.sql
   ```

   文件内容如下所示。

   ```
   CREATE DATABASE IF NOT EXISTS `default_catalog`.`flink_cdc`;
   ```

```
--数据的订单源表。
CREATE TABLE IF NOT EXISTS `default_catalog`.`flink_cdc`.`orders_src`(
  `order_id` INT NOT NULL,
  `order_revenue` FLOAT NOT NULL,
  `order_region` STRING NOT NULL,
  `customer_id` INT NOT NULL,
  PRIMARY KEY(`order_id`) NOT ENFORCED
) with (
  'connector' = 'mysql-cdc',
  'hostname' = 'rm-2ze5h9qnki343****.mysql.rds.aliyuncs.com',
  'port' = '3306',
  'username' = 'emr_test',
  'password' = 'Yz12****',
  'database-name' = 'flink_cdc',
  'table-name' = 'orders'
);
CREATE TABLE IF NOT EXISTS `default_catalog`.`flink_cdc`.`customers_src` (
  `customer_id` INT NOT NULL,
  `customer_age` FLOAT NOT NULL,
  `customer_name` STRING NOT NULL,
  PRIMARY KEY(`customer_id`) NOT ENFORCED
) with (
  'connector' = 'mysql-cdc',
  'hostname' = 'rm-2ze5h9qnki343****.mysql.rds.aliyuncs.com',
  'port' = '3306',
  'username' = 'emr_test',
  'password' = 'Yz12****',
  'database-name' = 'flink_cdc',
  'table-name' = 'customers'
);
-- create ods dwd and dws tables
CREATE TABLE IF NOT EXISTS `default_catalog`.`flink_cdc`.`ods_order_table` (
  `order_id` INT,
  `order_revenue` FLOAT,
  `order_region` VARCHAR(40),
  `customer_id` INT,
  PRIMARY KEY (order_id) NOT ENFORCED
) WITH (
  'connector' = 'upsert-kafka',
  'topic' = 'ods_order',
  'properties.bootstrap.servers' = '192.168.**.**:9092,192.168.**.**:9092,192.168.**.**:9092',
  'key.format' = 'json',
  'value.format' = 'json'
);
CREATE TABLE IF NOT EXISTS `default_catalog`.`flink_cdc`.`ods_customers_table` (
  `customer_id` INT,
  `customer_age` FLOAT,
  `customer_name` STRING,
  PRIMARY KEY (customer_id) NOT ENFORCED
) WITH (
  'connector' = 'upsert-kafka',
  'topic' = 'ods_customers',
  'properties.bootstrap.servers' = '192.168.**.**:9092,192.168.**.**:9092,192.168.**.**:9092',
  'key.format' = 'json',
  'value.format' = 'json'
);
CREATE TABLE IF NOT EXISTS `default_catalog`.`flink_cdc`.`dwd_order_customer_valid` (
  `order_id` INT,
  `order_revenue` FLOAT,
  `order_region` STRING,
  `customer_id` INT,
  `customer_age` FLOAT,
  `customer_name` STRING,
  PRIMARY KEY (order_id) NOT ENFORCED
) WITH (
  'connector' = 'upsert-kafka',
  'topic' = 'dwd_order_customer_valid',
  'properties.bootstrap.servers' = '192.168.**.**:9092,192.168.**.**:9092,192.168.**.**:9092',
  'key.format' = 'json',
  'value.format' = 'json'
);
CREATE TABLE IF NOT EXISTS `default_catalog`.`flink_cdc`.`dws_agg_by_region` (
```

```
  `order_region` VARCHAR(40),
  `order_cnt` BIGINT,
  `order_total_revenue` FLOAT,
  PRIMARY KEY (order_region) NOT ENFORCED
) WITH (
  'connector' = 'upsert-kafka',
  'topic' = 'dws_agg_by_region',
  'properties.bootstrap.servers' = '192.168.**.**:9092,192.168.**.**:9092,192.168.**.**:9092',
  'key.format' = 'json',
  'value.format' = 'json'
);
USE flink_cdc;
BEGIN STATEMENT SET;
INSERT INTO ods_order_table SELECT * FROM orders_src;
INSERT INTO ods_customers_table SELECT * FROM customers_src;
INSERT INTO dwd_order_customer_valid
SELECT
  o.order_id,
  o.order_revenue,
  o.order_region,
  c.customer_id,
  c.customer_age,
  c.customer_name
FROM customers_src c JOIN orders_src o ON c.customer_id=o.customer_id
WHERE c.customer_id <> -1;
INSERT INTO dws_agg_by_region
SELECT
  order_region,
  count(*) as order_cnt,
  sum(order_revenue) as order_total_revenue
FROM dwd_order_customer_valid
GROUP BY order_region;
END;
```

涉及参数如下所示：

- 创建数据表orders_src和customers_src。

参数	描述
connector	固定值为mysql-cdc。
hostname	RDS的内网地址。 您可以在RDS的数据库连接页面，单击内网地址进行复制。例如，rm-2ze5h9qnki343****.mysql.rds.aliyuncs.com。
port	固定值为3306。
username	步骤一：创建MySQL源数据表中创建的账号名。本示例为emr_test。
password	步骤一：创建MySQL源数据表中创建的账号的密码。本示例为Yz12****。
database-name	步骤一：创建MySQL源数据表中创建的数据库名。本示例为flink_cdc。
table-name	步骤一：创建MySQL源数据表中创建的数据表。 ■ orders_src：本示例为orders。 ■ customers_src：本示例为customers。

- 创建数据表ods_order_table、ods_customers_table、dwd_order_customer_valid和dws_agg_by_region。

参数	描述
connector	固定值为upsert-kafka。
topic	步骤二：创建Kafka的Topic中创建的Topic名称。 ■ ods_order_table：本示例为ods_order。 ■ ods_customers_table：本示例为ods_customers。 ■ dwd_order_customer_valid：本示例为dwd_order_customer_valid。 ■ dws_agg_by_region：本示例为dws_agg_by_region。

参数	描述
properties.bootstrap.servers	固定格式为 `192.168.**.**:9092,192.168.**.**:9092,192.168.**.**:9092` 。

6. 执行以下命令，启动Flink任务。

```
/opt/apps/FLINK/flink-current/bin/sql-client.sh -f demo.sql
```

步骤五：查看数据库和表信息

1. 使用SSH方式登录StarRocks集群，具体操作请参见登录集群。
2. 执行以下，连接StarRocks集群。

```
mysql -h127.0.0.1 -P 9030 -uroot
```

3. 执行以下命令，查询数据库信息。
   i. 执行以下命令，使用数据库。

```
use flink_cdc;
```

   ii. 执行以下命令，查看表信息。

```
show tables;
```

   返回信息如下所示。

```
+--------------------------+
| Tables_in_flink_cdc      |
+--------------------------+
| customers                |
| dwd_order_customer_valid |
| dws_agg_by_region        |
| orders                   |
+--------------------------+
4 rows in set (0.01 sec)
```

步骤六：场景演示，查询插入后的数据

1. 使用步骤一：创建MySQL源数据表中创建的测试账号连接MySQL实例，具体操作请参见通过DMS登录RDS MySQL。
2. 在RDS数据库窗口执行以下命令，向表orders和customers中插入数据。

```
INSERT INTO flink_cdc.orders(order_id,order_revenue,order_region,customer_id) VALUES(1,10,"beijing",1);
INSERT INTO flink_cdc.orders(order_id,order_revenue,order_region,customer_id) VALUES(2,10,"beijing",1);
INSERT INTO flink_cdc.customers(customer_id,customer_age,customer_name) VALUES(1, 22, "emr_test");
```

3. 使用SSH方式登录StarRocks集群，具体操作请参见登录集群。
4. 执行以下命令，连接StarRocks集群。

```
mysql -h127.0.0.1 -P 9030 -uroot
```

5. 执行以下命令，查询ODS层数据。
   i. 执行以下命令，使用数据库。

```
use flink_cdc;
```

   ii. 执行以下命令，查看orders表信息。

```
select * from orders;
```

   返回信息如下所示。

```
+----------+---------------+--------------+-------------+
| order_id | order_revenue | order_region | customer_id |
+----------+---------------+--------------+-------------+
|        1 |            10 | beijing      |           1 |
|        2 |            10 | beijing      |           1 |
+----------+---------------+--------------+-------------+
```

iii. 执行以下命令，查看customers表信息。

```
select * from customers;
```

返回信息如下所示。

```
+-------------+--------------+---------------+
| customer_id | customer_age | customer_name |
+-------------+--------------+---------------+
|           1 |           22 | emr_test      |
+-------------+--------------+---------------+
```

6. 执行以下命令，查询DWD层数据。
   i. 执行以下命令，使用数据库。

```
use flink_cdc;
```

   ii. 执行以下命令，查看orders表信息。

```
select * from dwd_order_customer_valid;
```

   返回信息如下所示。

```
+----------+---------------+--------------+-------------+--------------+---------------+
| order_id | order_revenue | order_region | customer_id | customer_age | customer_name |
+----------+---------------+--------------+-------------+--------------+---------------+
|        1 |            10 | beijing      |           1 |           22 | emr_test      |
|        2 |            10 | beijing      |           1 |           22 | emr_test      |
+----------+---------------+--------------+-------------+--------------+---------------+
2 rows in set (0.00 sec)
```

7. 执行以下命令，查询DWS层数据。
   i. 执行以下命令，使用数据库。

```
use flink_cdc;
```

   ii. 执行以下命令，查看orders表信息。

```
select * from dws_agg_by_region;
```

   返回信息如下所示。

```
+--------------+-----------+---------------------+
| order_region | order_cnt | order_total_revenue |
+--------------+-----------+---------------------+
| beijing      |         2 |                  20 |
+--------------+-----------+---------------------+
1 row in set (0.01 sec)
```

5.3.2.2.5. Flink CDC实现MySQL至StarRocks的数据同步

本文以MySQL和EMR-3.39.1版本为例，为您介绍如何使用Flink和StarRocks实现实时数仓中TP（Transaction Processing）和AP（Analytical Processing）数据同步的场景。

前提条件

- 已在新版控制台创建DataFlow集群，详情请参见创建集群。
- 已创建StarRocks集群，详情请参见创建集群。
- 已创建RDS MySQL，详情请参见创建RDS MySQL实例。

> **说明** 本文以MySQL 5.7版本为例介绍。

使用限制

- DataFlow集群、StarRocks集群和RDS MySQL实例需要在同一个VPC下。
- DataFlow集群和StarRocks集群均须开启公网访问。
- DataFlow集群和StarRocks集群均为EMR-3.39.1版本。
- RDS MySQL须为5.7及以上版本。

注意事项

- 如果RDS的表有修改（`ALTER TABLE`），则MySQL修改后的Schema变更需要在StarRocks手动同步。如果RDS的表有新建，则MySQL新加的表需要重新运行StarRocks Migrate Tool以进行数据同步。
- 本文档仅供测试使用，生产级别的Flink作业请使用阿里云VVP产品进行配置，或者使用YARN或者Kubernetes提交作业。

  详情请参见Apache Hadoop YARN和Native Kubernetes。

步骤一：准备测试数据

1. 创建测试的数据库和账号，详情请参见创建数据库和账号。

   创建完数据库和账号后，需要授权测试账号的读写权限。

   > **说明** 本文创建的数据库名称为test_cdc。

2. 使用创建的测试账号连接MySQL实例，详情请参见通过DMS登录RDS MySQL。
3. 执行以下命令，创建数据表。

```
CREATE TABLE test_cdc.`t_user` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  `age` tinyint(4) DEFAULT NULL,
  `create_time` datetime DEFAULT NULL,
  `update_time` datetime DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

步骤二：配置同步工具和启动Flink任务

1. 下载Flink CDC connector和Flink StarRocks Connector，并上传到DataFlow集群的*/opt/apps/FLINK/flink-current/lib*目录下。

   > **说明** 下载过程需要一定时间，请耐心等待。

2. 使用SSH方式登录DataFlow集群，详情请参见登录集群。
3. 执行以下命令，启动集群。

   > **注意** 本文示例仅供测试，如果是生产级别的Flink作业请使用YARN或Kubernetes方式提交，详情请参见Apache Hadoop YARN和Native Kubernetes。

```
/opt/apps/FLINK/flink-current/bin/start-cluster.sh
```

4. 下载并修改配置文件。
   i. 下载StarRocks Migrate Tool，并上传到DataFlow集群的*root*目录下。
   ii. 执行以下命令，解压缩*smt.tar.gz*文件。

```
tar -zxvf smt.tar.gz && cd smt
```

iii. 执行以下命令，编辑*config_prod.conf*文件。

```
vim conf/config_prod.conf
```

请根据实际信息修改各参数值，各参数描述如下表所示。

参数	描述
host	RDS的内网地址。 您可以在RDS的数据库连接页面，单击内网地址进行复制。例如，rm-bp1nu0c46fn9k****.mysql.rds.aliyuncs.com。
port	固定值3306。
user	RDS上创建的账号。 填写步骤一：准备测试数据中账号的用户名。
password	RDS上创建账号的密码。 填写步骤一：准备测试数据中账号的密码。
be_num	StarRocks集群BE（Backend）个数，如果是最小集群，则直接设置为1。
database	正则表达式用于匹配RDS数据库的名称，表示需要同步到StarRocks的数据库。例如，`^test.*$`。
table	正则表达式用于匹配RDS表的名称，表示需要同步到StarRocks的表。例如，`^.*$`。
flink.starrocks.jdbc-url	用于在StarRocks中执行查询操作。 例如，jdbc:mysql://172.17.**.**:9030。其中，`172.17.**.**`为StarRocks集群的内网IP地址。
flink.starrocks.load-url	指定FE的IP地址和HTTP端口，格式为`StarRocks集群的内网IP地址:8030`。 例如，172.17.**.**:8030。
flink.starrocks.username	StarRocks连接用户名。固定为root。
flink.starrocks.password	StarRocks连接密码。 **说明** 默认值为空，可以不填写密码。

配置文件详细信息请参见示例。

5. 执行以下命令，将所有建表语句都生成在*result*目录下。

```
./starrocks-migrate-tool
```

您可以通过`ls result`命令，查看*result*下的目录。

返回信息如下所示。

```
flink-create.1.sql  flink-create.all.sql  starrocks-create.1.sql  starrocks-create.all.sql  starrocks-external-create.1.sql  starrocks-external-create.all.sql
```

说明

- 本文示例的配置文件仅定义了table-rule.1等规则，*.1.sql*格式文件对应的就是table-rule.1的建表语句。如果您的配置文件有table-rule.2等规则，则*.all.sql*的文件为所有规则的集合。
- `external-create`后缀的文件，为对应数据源的外表。如果对于部分场景小的维表您不想同步，则可以直接通过外表查询，使用该文件可以生成对应的外表。本文示例未使用。

6. 执行以下命令，创建StarRocks表。

```
mysql -h<Header节点的内网IP地址> -P9030 -uroot -p < result/starrocks-create.1.sql
```

说明 如果修改*config_prod.conf*文件时，没有设置StarRocks连接密码，则直接按回车键。

7. 执行以下命令，启动Flink任务。

```
/opt/apps/FLINK/flink-current/bin/sql-client.sh -f result/flink-create.1.sql
```

步骤三：查看数据库和表信息

1. 使用SSH方式登录StarRocks集群，详情请参见登录集群。
2. 执行以下命令，查看数据库信息。

```
show databases;
```

返回信息如下所示。

```
+--------------------+
| Database           |
+--------------------+
| _statistics_       |
| information_schema |
| test_cdc           |
+--------------------+
3 rows in set (0.00 sec)
```

3. 执行以下命令，查看表信息。

```
 use test_cdc;
```

步骤四：场景演示

查询插入后的数据

1. 在RDS数据库窗口执行以下命令，插入数据。

```
INSERT INTO test_cdc.t_user(`name`,age,create_time,update_time) VALUES("aliyun.com.0",30,NOW(),NOW());
INSERT INTO test_cdc.t_user(`name`,age,create_time,update_time) VALUES("aliyun.com.1",31,NOW(),NOW());
INSERT INTO test_cdc.t_user(`name`,age,create_time,update_time) VALUES("aliyun.com.2",32,NOW(),NOW());
```

2. 在StarRocks连接窗口执行以下命令，查看表数据。

```
select * from t_user;
```

返回信息如下，表示数据已成功插入。

```
+------+--------------+------+---------------------+---------------------+
| id   | name         | age  | create_time         | update_time         |
+------+--------------+------+---------------------+---------------------+
| 4    | aliyun.com.0 |   30 | 2022-03-10 13:22:41 | 2022-03-10 13:22:41 |
| 5    | aliyun.com.1 |   31 | 2022-03-10 13:22:41 | 2022-03-10 13:22:41 |
| 6    | aliyun.com.2 |   32 | 2022-03-10 13:22:42 | 2022-03-10 13:22:42 |
+------+--------------+------+---------------------+---------------------+
3 rows in set (0.00 sec)
```

同步数据更新

1. 在RDS数据库窗口执行以下命令，更新指定数据。

```
UPDATE test_cdc.t_user SET age=35 where name="aliyun.com.0";
```

2. 在StarRocks连接窗口执行以下命令，查看表数据。

```
select * from t_user where name = "aliyun.com.0";
```

返回信息如下，表示数据已同步更新。

```
+------+--------------+------+---------------------+---------------------+
| id   | name         | age  | create_time         | update_time         |
+------+--------------+------+---------------------+---------------------+
| 4    | aliyun.com.0 |   35 | 2022-03-10 13:22:41 | 2022-03-10 13:22:41 |
+------+--------------+------+---------------------+---------------------+
1 row in set (0.01 sec)
```

同步数据删除

1. 在RDS数据库窗口执行以下命令，删除指定数据。

```
DELETE FROM test_cdc.t_user where 1=1;
```

2. 在StarRocks连接窗口执行以下命令，查看表数据。

```
select * from t_user;
```

返回信息如下，表示数据已同步删除。

```
Empty set (0.01 sec)
```

示例

配置文件示例如下。

```
[db]
host = rm-bp1nu0c46fn9k****.mysql.rds.aliyuncs.com
port = 3306
user = ***
password = ***
# currently available types: `mysql`, `pgsql`, `oracle`, `hive`, `clickhouse`
type = mysql
# # only takes effect on `type == hive`.
# # Available values: kerberos, none, nosasl, kerberos_http, none_http, zk, ldap
# authentication = kerberos
[other]
# number of backends in StarRocks
be_num = 1
# `decimal_v3` is supported since StarRocks-1.8.1
use_decimal_v3 = false
# directory to save the converted DDL SQL
output_dir = ./result
# !!!`database` `table` `schema` are case sensitive in `oracle`!!!
[table-rule.1]
# pattern to match databases for setting properties
# !!! database should be a `whole instance(or pdb) name` but not a regex when it comes with an `oracle db` !!!
database = ^test.*$
# pattern to match tables for setting properties
table = ^.*$
# `schema` only takes effect on `postgresql` and `oracle` and `sqlserver`
schema = ^.*$
############################################
### starrocks table configurations
############################################
# # set a column as the partition_key
# partition_key = p_key
# # override the auto-generated partitions
# partitions = START ("2021-01-02") END ("2021-01-04") EVERY (INTERVAL 1 day)
# # only take effect on tables without primary keys or unique indexes
# duplicate_keys=k1,k2
# # override the auto-generated distributed keys
# distributed_by=k1,k2
# # override the auto-generated distributed buckets
# bucket_num=32
# # properties.xxxxx: properties used to create tables
# properties.in_memory = false
############################################
### flink sink configurations
### DO NOT set `connector`, `table-name`, `database-name`, they are auto-generated
############################################
flink.starrocks.jdbc-url=jdbc:mysql://172.17.**.**:9030
flink.starrocks.load-url=172.17.**.**:8030
flink.starrocks.username=root
flink.starrocks.password=
flink.starrocks.sink.max-retries=10
flink.starrocks.sink.buffer-flush.interval-ms=15000
flink.starrocks.sink.properties.format=json
flink.starrocks.sink.properties.strip_outer_array=true
# # used to set the server-id for mysql-cdc jobs instead of using a random server-id
# flink.cdc.server-id = 5000
############################################
### flink-cdc plugin configuration for `postgresql`
############################################
# # for `9.*` decoderbufs, wal2json, wal2json_rds, wal2json_streaming, wal2json_rds_streaming
# # refer to https://ververica.github.io/flink-cdc-connectors/master/content/connectors/postgres-cdc.html
# # and https://debezium.io/documentation/reference/postgres-plugins.html
# flink.cdc.decoding.plugin.name = decoderbufs
```

5.3.2.2.6. 数据湖分析场景一：查询阿里云OSS上的数据

本文通过创建Iceberg表为例，介绍如何使用StarRocks的数据湖分析能力查询阿里云OSS上的数据。

背景信息

StarRocks具体场景的描述，请参见数据湖分析。StarRocks更具体的操作说明，请参见基础使用。

前提条件

- 已在OSS上创建存储空间，具体操作请参见创建存储空间。
- 已创建EMR-5.4.1及后续的版本的Hadoop集群，具体操作请参见创建集群。
- 已创建EMR-5.6.0及后续的版本的StarRocks集群，具体操作请参见创建集群。

> **说明** 本文示例中的Hadoop集群为EMR-5.4.1版本，StarRocks集群为EMR-5.6.0版本。

使用限制

创建的Hadoop集群和StarRocks集群需要在同一个VPC下，并且在同一个可用区下。

注意事项

目前不支持查询阿里云OSS-HDFS服务（JindoFS服务）上的数据。OSS-HDFS服务的详细信息请参见OSS-HDFS服务概述，具体操作请参见OSS-HDFS服务快速入门。

操作步骤

1. 添加OSS相关的配置。
   i. 进入StarRocks服务页面。
      a. 登录EMR on ECS控制台。
      b. 在顶部菜单栏处，根据实际情况选择地域和资源组。
      c. 在**集群管理**页面，单击StarRocks集群操作列的**集群服务**。
      d. 单击StarRocks服务区域的**配置**。
   ii. 修改配置项。
      a. 在StarRocks的**服务配置**区域，单击**core-site.xml**页签。
      b. 修改以下参数的参数值。

参数	描述
fs.oss.accessKeyId	OSS的AccessKey ID。
fs.oss.accessKeySecret	OSS的AccessKey Secret。
fs.oss.endpoint	OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。

      c. 单击**保存**。
      d. 在**确认修改配置**对话框中， 输入执行原因，单击**确定**。
   iii. 配置Accesskey。
      a. 在StarRocks的**服务配置**区域，单击**be.conf**页签。
      b. 单击**自定义配置**。
      c. 在**新增配置项**对话框中，新增以下配置项。

参数	描述
bject_storage_access_key_id	OSS的AccessKey ID。
object_storage_secret_access_key	OSS的AccessKey Secret。
object_storage_endpoint	OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。

      d. 单击**确定**。
      e. 在**确认修改配置**对话框中， 输入执行原因，单击**确定**。
   iv. 重启StarRocks服务，具体操作请参见重启服务。
2. 准备数据。
   i. 使用SSH方式登录Hadoop集群，具体操作请参见登录集群。
   ii. 执行以下命令，启动Spark SQL。

```
spark-sql --packages org.apache.iceberg:iceberg-spark3-runtime:0.12.1 --conf spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions --conf spark.sql.catalog.spark_catalog=org.apache.iceberg.spark.SparkSessionCatalog --conf spark.sql.catalog.spark_catalog.type=hive --conf spark.executor.instances=20
```

iii. 执行以下命令，创建数据库。

```
CREATE DATABASE iceberg;
```

iv. 执行以下命令，创建测试表。

```
CREATE TABLE iceberg_table_oss (id bigint, data string) USING iceberg TBLPROPERTIES('write.parquet.compression-code
c'='snappy') location 'oss://<yourBucketName>/iceberg/iceberg_table';
```

v. 执行以下命令，插入数据。

```
INSERT INTO iceberg_table_oss VALUES (1, 'a'), (2, 'b'), (3, 'c');
```

3. 创建外表。

i. 使用SSH方式登录StarRocks集群，具体操作请参见登录集群。

ii. 执行以下命令，连接StarRocks集群。

```
mysql -h127.0.0.1 -P 9030 -uroot
```

iii. 执行以下命令，创建Iceberg资源。

```
CREATE EXTERNAL RESOURCE "iceberg0"
PROPERTIES (
  "type" = "iceberg",
  "starrocks.catalog-type"="HIVE",
  "iceberg.catalog.hive.metastore.uris"="thrift://192.168.**.**:9083"
);
```

参数	描述
RESOURCE	创建的Iceberg的资源名。您可以自定义，本示例为iceberg0。
type	固定值为iceberg。
iceberg.catalog.hive.metastore.uris	Hadoop集群Thrift Server的内网地址。 格式为 `thrift://<Hadoop集群的内网IP地址>:9083` 。 您可以在Hadoop集群的**节点管理**页面，获取任意节点的内网IP地址。

iv. 执行以下命令，创建测试数据库。

```
CREATE DATABASE iceberg_db;
```

v. 执行以下命令，使用创建的测试数据库。

```
USE iceberg_db;
```

vi. 执行以下命令，创建外表。

```
CREATE EXTERNAL TABLE `iceberg_tbl_oss` (
  `id` bigint NULL,
  `data` varchar(200) NULL
) ENGINE=ICEBERG
PROPERTIES (
  "resource" = "iceberg0",
  "database" = "iceberg",
  "table" = "iceberg_table_oss"
);
```

参数	描述
resource	已创建的资源名称。 本文示例为iceberg0。
database	Iceberg数据库名称。 在步骤2中指定的数据库的名称，本文示例为iceberg。如果未指定，则为default。
table	Iceberg数据表名称。 在步骤2中创建的表名。本文示例为iceberg_table_oss。

4. 查询数据。

   在StarRocks集群中执行以下命令，查询表数据。

   ```
   select * from iceberg_tbl_oss;
   ```

   返回信息如下所示。

   ```
   +------+------+
   | id   | data |
   +------+------+
   |    2 | b    |
   |    1 | a    |
   |    3 | c    |
   +------+------+
   ```

5.3.2.2.7. 数据湖分析场景二：查询Iceberg数据源

本文通过示例为您介绍，如何使用E-MapReduce上的StarRocks集群进行Iceberg数据源查询。

背景信息

StarRocks具体场景的描述，请参见数据湖分析。StarRocks更具体的操作说明，请参见基础使用。

前提条件

- 已创建EMR-5.4.1及后续的版本的Hadoop集群，具体操作请参见创建集群。
- 已创建EMR-5.6.0及后续的版本的StarRocks集群，具体操作请参见创建集群。

> **说明** 本文示例中的Hadoop集群为EMR-5.4.1版本，StarRocks集群为EMR-5.6.0版本。

使用限制

- Hadoop集群须为EMR-5.4.1及后续的版本。
- StarRocks集群须为EMR-5.6.0及后续的版本。
- 创建的Hadoop集群和StarRocks集群需要在同一个VPC下，并且在同一个可用区下。

操作步骤

1. 准备数据。
   i. 使用SSH方式登录Hadoop集群，具体操作请参见登录集群。
   ii. 执行以下命令，启动Spark SQL。

   ```
   spark-sql --packages org.apache.iceberg:iceberg-spark3-runtime:0.12.1 --conf spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions --conf spark.sql.catalog.spark_catalog=org.apache.iceberg.spark.SparkSessionCatalog --conf spark.sql.catalog.spark_catalog.type=hive --conf spark.executor.instances=20
   ```

   iii. 执行以下命令，创建数据库。

   ```
   CREATE DATABASE iceberg;
   ```

   iv. 执行以下命令，创建测试表。

   ```
   CREATE TABLE iceberg_table (id bigint, data string) USING iceberg TBLPROPERTIES('write.parquet.compression-codec'='snappy') location 'hdfs://192.168.**.**:9000/user/hive/warehouse/iceberg.db/iceberg_table';
   ```

   代码中的location填写的格式为 `'hdfs://{Hadoop集群的内网地址}:9090//user/hive/warehouse/iceberg.db/{表名}'` 。此处的location使用的是内网IP地址而不是host，主要是因为Hadoop集群默认使用host作为location创建Hive表，而StarRocks集群没有对应的host，所以为了避免配置host使用了内网IP地址。

   > **说明** 如果您没有使用内网IP地址，则可能会遇到unknown host问题，此时需要您给每个BE和FE都配置下host信息，即在*/etc*目录下的*hosts*文件中增加一行信息 `<Hadoop集群的内网IP地址> emr-header-1-****` ，例如 `192.168.**.** emr-header-1.cluster-26****` 。

   v. 执行以下命令，插入数据。

   ```
   INSERT INTO iceberg_table VALUES (1, 'a'), (2, 'b'), (3, 'c');
   ```

2. 创建外表。
   i. 使用SSH方式登录StarRocks集群，具体操作请参见登录集群。

ii. 执行以下命令，连接StarRocks集群。

```
mysql -h127.0.0.1 -P 9030 -uroot
```

iii. 执行以下命令，创建Iceberg资源。

```
CREATE EXTERNAL RESOURCE "iceberg0"
PROPERTIES (
  "type" = "iceberg",
  "starrocks.catalog-type"="HIVE",
  "iceberg.catalog.hive.metastore.uris"="thrift://192.168.**.**:9083"
);
```

参数	描述
type	固定值为iceberg。
iceberg.catalog.hive.metastore.uris	Hadoop集群Thrift Server的内网地址。 格式为 `thrift://<Hadoop集群的内网IP地址>:9083` 。 您可以在Hadoop集群的**节点管理**页面，获取任意节点的内网IP地址。

iv. 执行以下命令，创建测试数据库。

```
CREATE DATABASE iceberg_db;
```

v. 执行以下命令，使用创建的测试数据库。

```
USE iceberg_db;
```

vi. 执行以下命令，创建外表。

```
CREATE EXTERNAL TABLE `iceberg_tbl` (
  `id` bigint NULL,
  `data` varchar(200) NULL
) ENGINE=ICEBERG
PROPERTIES (
  "resource" = "iceberg0",
  "database" = "iceberg",
  "table" = "iceberg_table"
);
```

参数	描述
resource	已创建的资源名称。 本文示例为iceberg0。
database	Iceberg数据库名称。 在步骤1中指定的数据库的名称，本文示例为iceberg。如果未指定，则为default。
table	Iceberg数据表名称。 在步骤1中创建的表名。本文示例为iceberg_table。

3. 查询数据。

在StarRocks集群中执行以下命令，查询表数据。

```
select * from iceberg_tbl;
```

返回信息如下所示。

```
+------+------+
| id   | data |
+------+------+
|    2 | b    |
|    1 | a    |
|    3 | c    |
+------+------+
3 rows in set (0.07 sec)
```

5.3.2.2.8. 数据湖分析场景三：查询Hive数据源

本文通过示例为您介绍，如何使用E-MapReduce上的StarRocks集群进行Hive数据源查询。

背景信息

StarRocks具体场景的描述，请参见数据湖分析。

前提条件

- 已创建EMR-3.40.0及后续的版本的Hadoop集群，具体操作请参见创建集群。
- 已创建EMR-5.6.0及后续的版本的StarRocks集群，具体操作请参见创建集群。

> **说明** 本文示例中的Hadoop集群为EMR-3.40.0版本，StarRocks集群为EMR-5.6.0版本。

使用限制

创建的Hadoop集群和StarRocks集群需要在同一个VPC下，并且在同一个可用区下。

操作步骤

1. 准备数据。
   i. 使用SSH方式登录Hadoop集群，具体操作请参见登录集群。
   ii. 执行以下命令，进入Hive命令行。

   ```
   hive
   ```

   iii. 执行以下命令，创建测试表。

   ```
   create table if not exists t (id bigint, value string) location 'hdfs://192.168.**.**:9000/user/hive/warehouse/t';
   ```

   代码中的location填写的格式为 `hdfs://{Hadoop集群的内网地址}:9090//user/hive/warehouse/{表名}` 。此处的location使用的是内网IP地址而不是host，主要是因为Hadoop集群默认使用host作为location创建Hive表，而StarRocks集群没有对应的host，所以为了避免配置host使用了内网IP地址。

   > **说明** 如果您没有使用内网IP地址，则可能会遇到unknown host问题，此时需要您给每个BE和FE都配置下host信息，即在*/etc*目录下的*hosts*文件中增加一行信息 `<Hadoop集群的内网IP地址> emr-header-1-****` ，例如 `192.168.**.** emr-header-1.cluster-26****` 。

   iv. 执行以下命令，插入数据。

   ```
   insert into table t select 1, 'value-1';
   ```

2. 创建外表。
   i. 使用SSH方式登录StarRocks集群，具体操作请参见登录集群。
   ii. 执行以下，连接StarRocks集群。

   ```
   mysql -h127.0.0.1 -P 9030 -uroot
   ```

   iii. 执行以下命令，创建Hive资源。

   ```
   CREATE EXTERNAL RESOURCE "hive0"
   PROPERTIES (
     "type" = "hive",
     "hive.metastore.uris" = "thrift://192.168.**.**:9083"
   );
   ```

参数	描述
type	Hive外表。固定值为hive。
hive.metastore.uris	Hadoop集群Thrift Server的内网地址。 格式为 `thrift://<Hadoop集群的内网IP地址>:9083` 。 您可以在Hadoop集群的**节点管理**页面，获取任意节点的内网IP地址。

   iv. 执行以下命令，创建测试数据库。

   ```
   CREATE DATABASE hive_test;
   ```

v. 执行以下命令，使用创建的测试数据库。

```
USE hive_test;
```

vi. 执行以下命令，创建外表。

```
CREATE EXTERNAL TABLE `test` (
  `id` bigint NULL,
  `value` string NULL
) ENGINE=HIVE
PROPERTIES (
  "resource" = "hive0",
  "database" = "default",
  "table" = "t"
);
```

参数	描述
resource	已创建的资源名称。 本文示例为hive0。
database	Hive数据库名称。 如果建表时未指定数据库，则为default。本文在准备数据阶段未指定数据库，所以为default。
table	Hive表名称。 在准备数据阶段创建的表名。本文示例为t。

3. 查询数据。

在StarRocks集群中，执行以下命令查询数据。

```
select * from test;
```

返回信息如下所示。

```
+------+---------+
| id   | value   |
+------+---------+
|    1 | value-1 |
+------+---------+
1 row in set (0.04 sec)
```

常见问题

Q：为什么查询不到新增的数据？

A：如果您在写数据之前就建立了外表并执行过查询，可能会出现查不到新增数据的情况，其原因是当前版本的StarRocks查询外表时会产生缓存，每隔一段时间会自动刷新。

您可以执行以下命令，手动刷新。

```
refresh external table test;
```

5.3.3. 数据导入

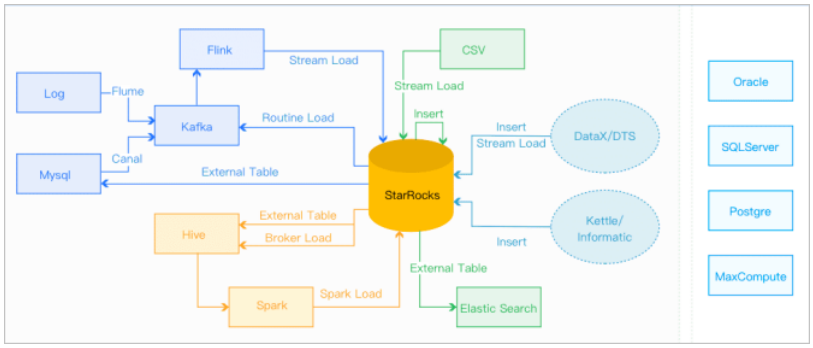
5.3.3.1. 导入概述

为了更好地满足各种不同的业务场景，StarRocks支持多种数据模型，StarRocks中存储的数据需要按照特定的模型进行组织。本文为您介绍数据导入的基本概念、原理、系统配置、不同导入方式的适用场景，以及一些最佳实践案例和常见问题。

背景信息

数据导入功能是将原始数据按照相应的模型进行清洗转换并加载到StarRocks中，方便查询使用。StarRocks提供了多种导入方式，您可以根据数据量大小或导入频率等要求选择最适合自己业务需求的导入方式。

StarRocks导入方式与各数据源关系图如下。



您可以根据不同的数据来源选择不同的导入方式：

- 离线数据导入：如果数据源是Hive或HDFS，推荐使用Broker Load。如果数据表很多导入比较麻烦可以使用Hive外表，性能会比Broker load导入效果差，但是可以避免数据搬迁。如果单表的数据量特别大，或者需要做为全局数据字典来精确去重可以考虑使用Spark Load。
- 实时数据导入：日志数据和业务数据库的Binlog同步到Kafka后，优先推荐通过Routine Load导入StarRocks。如果导入过程中有复杂的多表关联和ETL预处理可以使用Flink（Flink Connector）处理以后，再通过Stream Load写入StarRocks。
- 程序写入StarRocks：推荐使用Stream Load，可以参见Stream Load中Java或Python的Demo。
- 文本文件导入：推荐使用Stream Load。
- MySQL数据导入：推荐使用MySQL外表，通过 `insert into new_table select * from external_table` 的方式导入。
- StarRocks内部导入：推荐使用Insert Into方式导入，跟外部调度器配合实现简单的ETL处理。

注意事项

向StarRocks导入数据时，通常会采用程序对接的方式。以下是导入数据时的一些注意事项：

- 选择合适的导入方式：根据数据量大小、导入频次或数据源所在位置选择导入方式。

  例如，如果原始数据存放在HDFS上，则使用Broker load导入。
- 确定导入方式的协议：如果选择了Broker Load导入方式，则外部系统需要能使用MySQL协议定期提交和查看导入作业。
- 确定导入方式的类型：导入方式分为同步或异步。如果是异步导入方式，外部系统在提交创建导入后，必须调用查看导入命令，根据查看导入命令的结果来判断导入是否成功。
- 制定Label生成策略：Label生成策略需满足对每一批次数据唯一且固定的原则。
- 保证Exactly-Once：外部系统需要保证数据导入的At-Least-Once，StarRocks的Label机制可以保证数据导入的At-Most-Once，即可整体上保证数据导入的Exactly-Once。

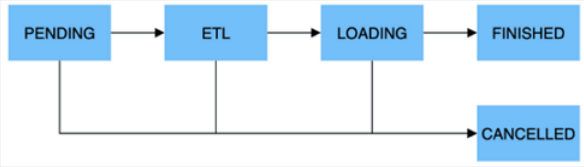
基本概念

名词	描述
导入作业	读取用户提交的源数据并进行清洗转换后，将数据导入到StarRocks系统中。导入完成后，数据即可被用户查询到。
Label	用于标识一个导入作业，所有导入作业都有一个Label。 Label可由用户指定或系统自动生成。Label在一个数据库内是唯一的，一个Label仅可用于一个成功的导入作业。当一个Label对应的导入作业成功后，不可再重复使用该Label提交导入作业。如果某Label对应的导入作业失败，则该Label可以被再使用。该机制可以保证Label对应的数据最多被导入一次，即At-Most-Once语义。
原子性	StarRocks中所有导入方式都提供原子性保证，即同一个导入作业内的所有有效数据要么全部生效，要么全部不生效，不会出现仅导入部分数据的情况。此处的有效数据不包括由于类型转换错误等数据质量问题而被过滤的数据，数据质量问题可以参见常见问题。
MySQL和HTTP协议	StarRocks提供MySQL协议和HTTP协议两种访问协议接口来提交作业。
Broker Load	Broker导入，即通过部署的Broker程序读取外部数据源（例如HDFS）中的数据，并导入到StarRocks。Broker进程利用自身的计算资源对数据进行预处理导入。
Spark Load	Spark导入，即通过外部资源（例如Spark）对数据进行预处理生成中间文件，StarRocks读取中间文件导入。Spark Load是一种异步的导入方式，您需要通过MySQL协议创建导入，并通过查看导入命令检查导入结果。
FE	Frontend，StarRocks系统的元数据和调度节点。在导入流程中主要负责导入执行计划的生成和导入任务的调度工作。
BE	Backend，StarRocks系统的计算和存储节点。在导入流程中主要负责数据的ETL和存储。

名词	描述
Tablet	StarRocks表的逻辑分片，一个表按照分区、分桶规则可以划分为多个分片，详情请参见数据分布。

基本原理

导入执行流程如下图所示。



一个导入作业主要分为以下五个阶段。

阶段	描述
PENDING	非必须。该阶段是指用户提交导入作业后，等待FE调度执行。 Broker Load和Spark Load包括该步骤。
ETL	非必须。该阶段执行数据的预处理，包括清洗、分区、排序和聚合等。 Spark Load包括该步骤，他使用外部计算资源Spark完成ETL。
LOADING	该阶段先对数据进行清洗和转换，然后将数据发送给BE处理。当数据全部导入后，进入等待生效过程，此时导入作业依旧是LOADING状态。
FINISHED	在导入作业涉及的所有数据均生效后，作业的状态变成FINISHED，FINISHED后导入的数据均可查询。FINISHED是导入作业的最终状态。
CANCELLED	在导入作业状态变为FINISHED之前，作业随时可能被取消并进入CANCELLED状态，例如，您手动取消或导入出现错误等。CANCELLED也是导入作业的一种最终状态。

数据导入格式如表。

类型	描述
整型类	TINYINT、SMALLINT、INT、BIGINT、LARGEINT。例如：1，1000，1234。
浮点类	FLOAT、DOUBLE、DECIMAL。例如：1.1，0.23，.356。
日期类	DATE、DATETIME。例如：2017-10-03，2017-06-13 12:34:03。
字符串类	CHAR、VARCHAR。例如：I am a student，a。

导入方式

为适配不同的数据导入需求，StarRocks系统提供了5种不同的导入方式，以支持不同的数据源（例如HDFS、Kafka和本地文件等），或者按不同的方式导入数据，StarRocks目前导入数据的方式分为同步导入和异步导入两种。

所有导入方式都支持CSV数据格式。其中Broker Load还支持Parquet和ORC数据格式。

导入方式	描述	导入类型
Broker Load	通过Broker进程访问并读取外部数据源，然后采用MySQL协议向StarRocks创建导入作业。提交的作业将异步执行，您可以通过 `SHOW LOAD` 命令查看导入结果。 Broker Load适用于源数据在Broker进程可访问的存储系统（例如HDFS）中，数据量为几十GB到上百GB，详细信息请参见Broker Load。	异步导入
Spark Load	通过外部的Spark资源实现对导入数据的预处理，提高StarRocks大数据量的导入性能并且节省StarRocks集群的计算资源。Spark Load是一种异步导入方式，需要通过MySQL协议创建导入作业，并通过 `SHOW LOAD` 查看导入结果。 Spark Load适用于初次迁移大数据量（可达到TB级别）到StarRocks的场景，且源数据在Spark可访问的存储系统（例如HDFS）中，详细信息请参见Spark Load。	异步导入

导入方式	描述	导入类型
Stream Load	是一种同步执行的导入方式。您可以通过HTTP协议发送请求将本地文件或数据流导入到StarRocks中，并等待系统返回导入的结果状态，从而判断导入是否成功。 Stream Load适用于导入本地文件，或通过程序导入数据流中的数据，详细信息请参见Stream Load。	同步导入
Routine Load	Routine Load（例行导入）提供了一种自动从指定数据源进行数据导入的功能。您可以通过MySQL协议提交例行导入作业，生成一个常驻线程，不间断的从数据源（例如Kafka）中读取数据并导入到StarRocks中，详细信息请参见Routine Load。	异步导入
Insert Into	类似MySQL中的Insert语句，StarRocks提供 `INSERT INTO tbl SELECT ...;` 的方式从StarRocks的表中读取数据并导入到另一张表，或者通过 `INSERT INTO tbl VALUES(...);` 插入单条数据，详细信息请参见Insert Into。	同步导入

> **注意** 如果是外部程序接入StarRocks的导入功能，需要先判断使用导入方式是哪类，然后再确定接入逻辑。

- 同步导入

  同步导入方式即用户创建导入任务，StarRocks同步执行，执行完成后返回导入结果。用户可以通过该结果判断导入是否成功。

  操作步骤：

  i. 用户（外部系统）创建导入任务。
  ii. StarRocks返回导入结果。
  iii. 用户（外部系统）判断导入结果。如果导入结果为失败，则可以再次创建导入任务。

- 异步导入

  异步导入方式即用户创建导入任务后，StarRocks直接返回创建成功。创建成功不代表数据已经导入成功。导入任务会被异步执行，用户在创建成功后，需要通过轮询的方式发送查看命令查看导入作业的状态。如果创建失败，则可以根据失败信息，判断是否需要再次创建。

  操作步骤：

  i. 用户（外部系统）创建导入任务。
  ii. StarRocks返回创建任务的结果。
  iii. 用户（外部系统）判断创建任务的结果，如果成功则进入步骤4；如果失败则可以回到步骤1，重新尝试创建导入任务。
  iv. 用户（外部系统）轮询查看任务状态，直至状态变为FINISHED或CANCELLED。

场景	描述
HDFS导入	如果HDFS导入源数据存储在HDFS中，当数据量为几十GB到上百GB时，则可以采用Broker Load方法向StarRocks导入数据。此时要求部署的Broker进程可以访问HDFS数据源。导入数据的作业异步执行，您可以通过 `SHOW LOAD` 命令查看导入结果。 如果源数据存储在HDSF中，当数据量达到TB级别时，则可以采用Spark Load方法向StarRocks导入数据。此时要求部署的Spark进程可以访问HDFS数据源。导入数据的作业异步执行，您可以通过 `SHOW LOAD` 命令查看导入结果。 对于其他外部数据源，只要Broker或Spark进程能读取对应数据源，也可以采用Broker Load或Spark Load方法导入数据。
本地文件导入	数据存储在本地文件中，数据量小于10 GB，可以采用Stream Load方法将数据快速导入StarRocks系统。采用HTTP协议创建导入作业，作业同步执行，您可以通过HTTP请求的返回值判断导入是否成功。
Kafka导入	数据来自于Kafka等流式数据源，需要向StarRocks系统导入实时数据时，可以采用Routine Load方法。您通过MySQL协议创建例行导入作业，StarRocks持续不断地从Kafka中读取并导入数据。
Insert Into导入	手工测试及临时数据处理时可以使用 `Insert Into` 方法向StarRocks表中写入数据。 其中， `INSERT INTO tbl SELECT ...;` 语句是从StarRocks的表中读取数据并导入到另一张表， `INSERT INTO tbl VALUES(...);` 语句是向指定表里插入单条数据。

导入方式介绍

导入类型

适用场景

内存限制

您可以通过设置参数来限制单个导入作业的内存使用，以防止导入占用过多的内存而导致系统OOM。不同导入方式限制内存的方式略有不同，详情可以参见各个导入方式的文档。

一个导入作业通常会分布在多个BE上执行，内存参数限制的是一个导入作业在单个BE上的内存使用，而不是在整个集群的内存使用。同时，每个BE会设置可用于导入作业的内存总上限，详情请参见通用系统配置。配置限制了所有在该BE上运行的导入任务的总体内存使用上限。

较小的内存限制可能会影响导入效率，因为导入流程可能会因为内存达到上限而频繁的将内存中的数据写回磁盘。而过大的内存限制可能导致当导入并发较高时系统OOM。所以需要根据需求合理地设置内存参数。

通用系统配置

以下配置属于FE的系统配置，可以通过FE的配置文件*fe.conf*来修改。

参数	描述
max_load_timeout_second	导入超时时间的最大、最小取值范围，均以秒为单位。默认的最大超时时间为3天，最小超时时间为1秒。您自定义的导入超时时间不可超过该范围。该参数通用于所有类型的导入任务。
min_load_timeout_second	
desired_max_waiting_jobs	等待队列可以容纳的最多导入任务数目，默认值为100。 例如，FE中处于PENDING状态（即等待执行）的导入任务数目达到该值，则新的导入请求会被拒绝。此配置仅对异步执行的导入有效，如果处于等待状态的异步导入任务数达到限额，则后续创建导入的请求会被拒绝。
max_running_txn_num_per_db	每个数据库中正在运行的导入任务的最大个数（不区分导入类型、统一计数），默认值为100。 当数据库中正在运行的导入任务超过最大值时，后续的导入任务不会被执行。如果是同步作业，则作业会被拒绝；如果是异步作业，则作业会在队列中等待。
label_keep_max_second	导入任务记录的保留时间。 已经完成的（FINISHED或CANCELLED）导入任务记录会在StarRocks系统中保留一段时间，时间长短则由此参数决定。参数默认值为3天。该参数通用于所有类型的导入任务。

以下配置属于BE的系统配置，可以通过BE的配置文件*be.conf*来修改。

参数	描述
push_write_mbytes_per_sec	BE上单个Tablet的写入速度限制。默认值是10，即10MB/s。 根据Schema以及系统的不同，通常BE对单个Tablet的最大写入速度大约在10~30MB/s之间。您可以适当调整该参数来控制导入速度。
write_buffer_size	导入数据在BE上会先写入到一个内存块，当该内存块达到阈值后才会写回磁盘。默认值为100 MB。 过小的阈值可能导致BE上存在大量的小文件。您可以适当提高该阈值减少文件数量。但过大的阈值可能导致RPC超时，详细请参见参数tablet_writer_rpc_timeout_sec。
tablet_writer_rpc_timeout_sec	导入过程中，发送一个Batch（1024行）的RPC超时时间。默认为600秒。 因为该RPC可能涉及多个分片内存块的写盘操作，所以可能会因为写盘导致RPC超时，可以适当调整超时时间来减少超时错误（例如send batch fail）。同时，如果调大参数write_buffer_size，则tablet_writer_rpc_timeout_sec参数也需要适当调大。
streaming_load_rpc_max_alive_time_sec	在导入过程中，StarRocks会为每个Tablet开启一个Writer，用于接收数据并写入。该参数指定了Writer的等待超时时间。默认为600秒。 如果在参数指定时间内Writer没有收到任何数据，则Writer会被自动销毁。当系统处理速度较慢时，Writer可能长时间接收不到下一批数据，导致导入报错 `TabletWriter add batch with unknown id` 。此时可适当调大该参数。
load_process_max_memory_limit_percent	分别为最大内存和最大内存百分比，限制了单个BE上可用于导入任务的内存上限。系统会在两个参数中取较小者，作为最终的BE导入任务内存使用上限。 • load_process_max_memory_limit_percent：表示对BE总内存限制的百分比。默认为80。总内存限制mem_limit默认为80%，表示对物理内存的百分比。即假设物理内存为M，则默认导入内存限制为M * 80% * 80%。 • load_process_max_memory_limit_bytes：默认为100 GB。
load_process_max_memory_limit_bytes	

FE配置

BE配置

常见问题

- Q：报错提示 `Label Already Exists` 。

A：同一个数据库内已经有一个相同Label的导入作业导入成功或者正在执行。需要检查不同导入方式之间是否有Label冲突，或者是否有任务重复提交了。

由于StarRocks系统中导入的Label不区分导入方式，所以存在其他导入方式使用了相同Label的问题。重复Label排查方法如下。

通过 `SHOW LOAD WHERE LABEL = "xxx"` 命令查看，其中xxx为待检查的Label字符串，查看是否已经存在具有相同Label的FINISHED状态的导入任务。

- Q：数据质量问题报错 `ETL_QUALITY_UNSATISFIED; msg:quality not good enough to cancel` 。

  A：可以通过SHOW LOAD中URL查看错误数据。常见的错误类型如下：

  - `convert csv string to INT failed.`

    导入文件某列的字符串转化对应类型的时候出错。例如，将abc转化为数字时失败。
  - `the length of input is too long than schema.`

    导入文件某列长度不正确，例如定长字符串超过建表时设置的长度、INT类型的字段超过4个字节。
  - `actual column number is less than schema column number.`

    导入文件某一行按照指定的分隔符切分后列数小于指定的列数，可能是分隔符不正确。
  - `actual column number is more than schema column number.`

    导入文件某一行按照指定的分隔符切分后列数大于指定的列数。
  - `the frac part length longer than schema scale.`

    导入文件某Decimal列的小数部分超过指定的长度。
  - `the int part length longer than schema precision.`

    导入文件某Decimal列的整数部分超过指定的长度。
  - `the length of decimal value is overflow.`

    导入文件某Decimal列的长度超过指定的长度。
  - `there is no corresponding partition for this key.`

    导入文件某行的分区列的值不在分区范围内。

5.3.3.2. Broker Load

在Broker Load模式下，通过部署的Broker程序，StarRocks可读取对应数据源（例如，Apache HDFS，阿里云OSS）上的数据，利用自身的计算资源对数据进行预处理和导入。本文为您介绍Broker Load导入的使用示例以及常见问题。

背景信息

Broker Load是一种异步的导入方式。您需要通过MySQL协议创建导入，并通过查看导入命令检查导入结果。StarRocks支持从外部存储系统导入数据，支持CSV、ORCFile和Parquet等文件格式，建议单次导入数据量在几十GB到上百GB级别。

Broker Load导入

阿里云EMR StarRocks集群在创建时已经自动搭建并启动Broker服务，Broker服务位于每个Core节点上。使用以下SQL命令可以查看Broker实例。

```
SHOW PROC "/brokers"\G
```

返回信息如下所示。

```
*************************** 1. row ***************************
          Name: broker
            IP: 10.0.1.151
          Port: 8000
         Alive: true
 LastStartTime: 2022-04-13 11:38:46
LastUpdateTime: 2022-04-13 15:26:44
        ErrMsg:
*************************** 2. row ***************************
          Name: broker
            IP: 10.0.1.154
          Port: 8000
         Alive: true
 LastStartTime: 2022-04-13 11:38:46
LastUpdateTime: 2022-04-13 15:26:44
        ErrMsg:
*************************** 3. row ***************************
          Name: broker
            IP: 10.0.1.153
          Port: 8000
         Alive: true
 LastStartTime: 2022-04-13 11:38:46
LastUpdateTime: 2022-04-13 15:26:44
        ErrMsg:
*************************** 4. row ***************************
          Name: broker
            IP: 10.0.1.152
          Port: 8000
         Alive: true
 LastStartTime: 2022-04-13 11:38:46
LastUpdateTime: 2022-04-13 15:26:44
        ErrMsg:
4 rows in set (0.00 sec)
```

- 语法

```
LOAD LABEL db_name.label_name
    (data_desc, ...)
WITH BROKER broker_name broker_properties
    [PROPERTIES (key1=value1, ... )]
```

- 参数描述

  执行 `HELP BROKER LOAD` 命令，可以查看创建导入作业的详细语法。

  - Lable

    导入任务的标识。每个导入任务都有一个唯一的Label。Label是您在导入命令中自定义的或系统自动生成的名称。通过该Label，您可以查看对应导入任务的执行情况，并且Label可以用来防止导入相同的数据。当导入任务状态为FINISHED时，对应的Label就不能再次使用了。当Label对应的导入任务状态为CANCELLED时，可以再次使用该Label提交导入作业。

- 数据描述类date_desc

  数据描述类参数，主要指的是语句中data_desc部分的参数。每组data_desc表述了本次导入涉及到的数据源地址、ETL函数，目标表及分区等信息。

  Broker Load支持一次导入任务涉及多张表，每个Broker Load导入任务可通过多个data_desc声明多张表来实现多表导入。每个单独的data_desc可以指定属于该表的数据源地址，可以用多个file_path来指定导入同一个表的多个文件。Broker Load保证了单次导入的多张表之间原子性成功或失败。date_desc常见参数如下所示。

```
data_desc:
    DATA INFILE ('file_path', ...)
    [NEGATIVE]
    INTO TABLE tbl_name
    [PARTITION (p1, p2)]
    [COLUMNS TERMINATED BY column_separator ]
    [FORMAT AS file_type]
    [(col1, ...)]
    [COLUMNS FROM PATH AS (colx, ...)]
    [SET (k1=f1(xx), k2=f2(xx))]
    [WHERE predicate]
```

  相关参数描述如下表所示。

参数	描述
file_path	文件路径可以指定到文件，也可以用星号（*）通配符指定某个目录下的所有文件。中间的目录也可以使用通配符匹配。 可以使用的通配符有? * [] {} ^，使用规则请参见FileSystem。 例如， 通过*hdfs://hdfs_host:hdfs_port/user/data/tablename//*，可以匹配*tablename*下所有分区内的所有文件。通过 *hdfs://hdfs_host:hdfs_port/user/data/tablename/dt=202104/*，可以匹配*tablename*下4月分区的所有文件。
negative	设置数据取反导入。 该功能适用的场景是当数据表中聚合列的类型均为SUM类型时，如果希望撤销某一批导入的数据，可以通过negative参数导入同一批数据，StarRocks会自动为这批数据在聚合列上数据取反，以达到消除同一批数据的功能。
partition	指定待导入表的Partition信息。 如果待导入数据不属于指定的Partition，则不会被导入。同时，不指定Partition中的数据会被认为是“错误数据”。对于不想导入，也不想记录为“错误数据”的数据，可以使用where predicate来过滤。
column_separator	COLUMNS TERMINATED BY column_separator，用于指定导入文件中的列分隔符，默认为\t。 如果是不可见字符，则需要加\x作为前缀，使用十六进制来表示分隔符。例如，Hive文件的分隔符为\x01，则列分隔符为\\x01。
file_type	FORMAT AS file_type，用于指定导入文件的类型。例如，parquet、orc、csv，默认值为csv。 parquet类型也可以通过文件后缀名.*parquet*或者.*parq*判断。

COLUMNS FROM PATH AS	提取文件路径中的分区字段。
	例如，导入文件为*/path/col_name=col_value/dt=20210101/file1*，其中*col_name/dt*为表中的列，则将col_value、20210101分别导入到col_name和dt对应的列的代码示例如下。
	`(col1, col2)`
	`COLUMNS FROM PATH AS (col_name, dt)`

set column mapping	SET (k1=f1(xx), k2=f2(xx))，data_desc中的SET语句负责设置列函数变换。 列函数变换支持所有查询的等值表达式变换。如果原始数据的列和表中的列不一一对应，则需要使用该属性。
where predicate	WHERE predicate，data_desc中的WHERE语句负责过滤已经完成transform的数据。 被过滤的数据不会进入容忍率的统计中。如果多个data_desc中声明了关于同一张表的多个条件，则会以AND语义合并这些条件。

- 导入作业参数

  导入作业参数是指Broker Load创建导入语句中属于broker_properties部分的参数。导入作业参数是作用于整个导入作业的。

  ```
  broker_properties:
      (key2=value2, ...)
  ```

  部分参数描述如下表所示。

参数	描述
timeout	导入作业的超时时间（以秒为单位）。 您可以在opt_properties中自行设置每个导入的超时时间。导入任务在设定的时限内未完成则会被系统取消，变为CANCELLED。Broker Load的默认导入超时时间为4小时。 **注意** 通常情况下，不需要您手动设置导入任务的超时时间。当在默认超时时间内无法完成导入时，可以手动设置任务的超时时间。 推荐超时时间的计算方式为：**超时时间** > （**（总文件大小**(MB) * **待导入的表及相关**Roll up**表的个数**）/ (30 * **导入并发数**）) 公式中的30为目前BE导入的平均速度，表示30 MB/s。例如，如果待导入数据文件为1 GB，待导入表包含2个Rollup表，当前的导入并发数为3，则timeout的最小值为 (1 * 1024 * 3) / (10 * 3) = 102 秒。 由于每个StarRocks集群的机器环境不同且集群并发的查询任务也不同，所以StarRocks集群的最慢导入速度需要您根据历史的导入任务速度进行推测。
max_filter_ratio	导入任务的最大容忍率，默认为0容忍，取值范围是0~1。当导入的错误率超过该值，则导入失败。如果您希望忽略错误的行，可以设置该参数值大于0，来保证导入可以成功。 计算公式为：`max_filter_ratio = (dpp.abnorm.ALL / (dpp.abnorm.ALL + dpp.norm.ALL ))` 其中，`dpp.abnorm.ALL` 表示数据质量不合格的行数，例如类型不匹配、列数不匹配和长度不匹配等。`dpp.abnorm.ALL` 指的是导入过程中正确数据的条数，可以通过 `SHOW LOAD` 命令查询导入任务的正确数据量。 **原始文件的行数** = dpp.abnorm.ALL + dpp.norm.ALL
load_mem_limit	导入内存限制。默认值为0，表示不限制。
strict_mode	Broker Load导入可以开启Strict Mode模式。开启方式为 `properties ("strict_mode" = "true")`。 默认关闭。 Strict Mode模式是对于导入过程中的列类型转换进行严格过滤。严格过滤的策略为，对于列类型转换，如果Strict Mode为true，则错误的数据将被过滤掉。错误数据是指原始数据并不为空值，在参与列类型转换后结果为空值的数据。但以下场景除外： ■ 对于导入的某列由函数变换生成时，Strict Mode对其不产生影响。 ■ 对于导入的某列类型包含范围限制的，如果原始数据能正常通过类型转换，但无法通过范围限制的，Strict Mode对其也不产生影响。例如，如果类型是decimal(1,0)，原始数据为10，则属于可以通过类型转换但不在列声明的范围内，Strict Mode对其不产生影响。

- 创建阿里云OSS导入任务示例

  注意 在阿里云EMR StarRocks上使用broker作为Broker名称即可。

```
LOAD LABEL tpch.lineitem
(
    DATA INFILE("oss://bucket/tpc_h/sf1/lineitem.tbl")
    INTO TABLE `lineitem`
    COLUMNS TERMINATED BY '|'
    (l_orderkey, l_partkey, l_suppkey, l_linenumber, l_quantity, l_extendedprice, l_discount, l_tax, l_returnflag, l_line
status, l_shipdate, l_commitdate, l_receiptdate, l_shipinstruct, l_shipmode, l_comment)
)
WITH BROKER broker
(
    "fs.oss.accessKeyId" = "xxx",
    "fs.oss.accessKeySecret" = "xxx",
    "fs.oss.endpoint" = "oss-cn-beijing-internal.aliyuncs.com"
);
```

Broker Load导入是异步的，您可以在 `SHOW LOAD` 命令中指定Label来查询对应导入作业的执行状态。具体语法可执行 `HELP SHOW LOAD` 命令查看。

> **注意** `SHOW LOAD` 命令只能查看异步导入方式的LOAD任务。同步方式的LOAD任务，例如Stream Load任务，目前无法使用 `SHOW LOAD` 命令查看。

查看导入任务状态示例如下。

```
show load where label = 'label1'\G
*************************** 1. row ***************************
         JobId: 76391
         Label: label1
         State: FINISHED
      Progress: ETL:N/A; LOAD:100%
          Type: BROKER
       EtlInfo: unselected.rows=4; dpp.abnorm.ALL=15; dpp.norm.ALL=28133376
      TaskInfo: cluster:N/A; timeout(s):10800; max_filter_ratio:5.0E-5
      ErrorMsg: N/A
    CreateTime: 2019-07-27 11:46:42
  EtlStartTime: 2019-07-27 11:46:44
 EtlFinishTime: 2019-07-27 11:46:44
 LoadStartTime: 2019-07-27 11:46:44
LoadFinishTime: 2019-07-27 11:50:16
           URL: http://192.168.**.**:8040/api/_load_error_log?file=__shard_4/error_log_insert_stmt_4bb00753932c491a-a6da6e2
725415317_4bb00753932c491a_a6da6e2725415317
    JobDetails: {"Unfinished backends":{"9c3441027ff948a0-8287923329a2b6a7":[10002]},"ScannedRows":2390016,"TaskNumber":1,"
All backends":{"9c3441027ff948a0-8287923329a2b6a7":[10002]},"FileNumber":1,"FileSize":1073741824}
```

返回参数的描述如下表所示。

参数	描述
JobId	导入任务的唯一ID，每个导入任务的JobId都不同，由系统自动生成。与Label不同的是，JobId永远不会相同，而Label则可以在导入任务失败后被复用。
Label	导入任务的标识。
State	导入任务当前所处的阶段。 • PENDING：表示当前导入任务正在等待被执行。 • LOADING：表示正在执行中。 • CANCELLED：表示导入失败。 • FINISHED：表示导入成功。
Progress	导入任务的进度描述。分为ETL和LOAD两种进度，分别对应导入流程的ETL和LOADING两个阶段。目前Broker Load只有LOADING阶段，所以ETL固定显示为N/A，而LOAD的进度范围为0~100%。 LOAD的进度的计算公式为 `LOAD进度 = 当前完成导入的表个数 / 本次导入任务设计的总表个数 * 100%` 。 如果所有导入表均完成导入，此时LOAD的进度为99%， 导入进入到最后生效阶段，待整个导入任务完成后，LOAD的进度才会改为100%。 > **注意** 导入进度并不是线性的，所以如果一段时间内进度没有变化，并不代表导入任务没有执行。
Type	导入任务的类型。Broker Load的Type取值是BROKER。
EtlInfo	主要显示导入的数据量指标unselected.rows，dpp.norm.ALL和dpp.abnorm.ALL。 您可以根据unselected.rows的参数值判断where条件过滤了多少行，根据dpp.norm.ALL和dpp.abnorm.ALL两个指标可以验证当前导入任务的错误率是否超过max-filter-ratio。三个指标之和就是原始数据量的总行数。
TaskInfo	主要显示当前导入任务参数，即创建Broker Load导入任务时您指定的参数，包括cluster，timeout和max-filter-ratio。

参数	描述
ErrorMsg	如果导入任务状态为CANCELLED，则显示失败的原因，包括type和msg两部分。如果导入任务成功则显示N/A。type的取值意义如下： • USER-CANCEL：取消的任务。 • ETL-RUN-FAIL：在ETL阶段失败的导入任务。 • ETL-QUALITY-UNSATISFIED：数据质量不合格，即错误数据率超过了max-filter-ratio。 • LOAD-RUN-FAIL：在LOADING阶段失败的导入任务。 • TIMEOUT：没在超时时间内完成的导入任务。 • UNKNOWN：未知的导入错误。

CreateTime
EtlStartTime
EtlFinishTime
LoadStartTime
LoadFinishTime
	分别表示导入创建的时间、ETL阶段开始的时间、ETL阶段完成的时间、LOADING阶段开始的时间和整个导入任务完成的时间。
	• 由于Broker Load导入没有ETL阶段，所以EtlStartTime、EtlFinishTime和LoadStartTime被设置为同一个值。
	• 如果导入任务长时间停留在CreateTime，而LoadStartTime为N/A，则说明目前导入任务堆积严重，您可以减少导入提交的频率。
	`LoadFinishTime - CreateTime = 整个导入任务所消耗时间`
	`LoadFinishTime - LoadStartTime = 整个Broker load导入任务执行时间 = 整个导入任务所消耗时间 - 导入任务等待的时间`

URL	导入任务的错误数据样例，访问URL地址即可获取本次导入的错误数据样例。当本次导入不存在错误数据时，URL字段为N/A。
JobDetails	显示作业的详细运行状态。包括导入文件的个数、总大小（字节）、子任务个数、已处理的原始行数，运行子任务的BE节点ID，以及未完成的BE节点ID。 `{"Unfinished backends":{"9c3441027ff948a0-8287923329a2b6a7":[10002]},"ScannedRows":2390016,"TaskNumber":1,"All backends":{"9c3441027ff948a0-8287923329a2b6a7":[10002]},"FileNumber":1,"FileSize":1073741824}` 其中已处理的原始行数，每5秒更新一次。该行数仅用于展示当前的进度，不代表最终实际的处理行数。实际处理行数以EtlInfo中显示的数据为准。

当Broker Load作业状态不为CANCELLED或FINISHED时，可以手动取消。取消时需要指定待取消导入任务的Label。可执行`HELP CANCEL LOAD`命令查看取消导入命令的语法。

```
CANCEL LOAD
[FROM db_name]
WHERE [LABEL = "load_label" | LABEL like "label_pattern"];
```

HDFS导入

- HDFS导入语法示例

```
LOAD LABEL db1.label1
(
    DATA INFILE("hdfs://emr-header-1.cluster-xxx:9000/user/hive/test.db/ml/file1")
    INTO TABLE tbl1
    COLUMNS TERMINATED BY ","
    (tmp_c1, tmp_c2)
    SET
    (
        id=tmp_c2,
        name=tmp_c1
    ),
    DATA INFILE("hdfs://emr-header-1.cluster-xxx:9000/user/hive/test.db/ml/file2")
    INTO TABLE tbl2
    COLUMNS TERMINATED BY ","
    (col1, col2)
    where col1 > 1
)
WITH BROKER 'broker1'
(
    "username" = "hdfs_username",
    "password" = "hdfs_password"
)
PROPERTIES
(
    "timeout" = "3600"
);
```

- HDFS认证

  社区版本的HDFS支持简单认证和Kerberos认证两种认证方式。

  - 简单认证（Simple）：用户的身份由与HDFS建立链接的客户端操作系统决定。

    涉及参数如下表。

参数	描述
hadoop.security.authentication	认证方式。默认值为simple。
username	HDFS的用户名。
password	HDFS的密码。

  - Kerberos认证：客户端的身份由用户自己的Kerberos证书决定。

    涉及参数如下表。

参数	描述
hadoop.security.authentication	认证方式。默认值为kerberos。
kerberos_principal	指定Kerberos的Principal。
kerberos_keytab	指定Kerberos的keytab文件路径。该文件必须为Broker进程所在服务器上的文件。
kerberos_keytab_content	指定Kerberos中keytab文件内容经过Base64编码之后的内容。 注意 该参数和kerberos_keytab参数只需配置一个。

- HDFS HA配置

  通过配置NameNode HA，可以在NameNode切换时，自动识别到新的NameNode。配置以下参数用于访问以HA模式部署的HDFS集群。

参数	描述
dfs.nameservices	指定HDFS服务的名称，您可以自定义。 例如，设置dfs.nameservices为my_ha。
dfs.ha.namenodes.xxx	自定义NameNode的名称，多个名称时以逗号（,）分隔。其中xxx为dfs.nameservices中自定义的名称。 例如，设置dfs.ha.namenodes.my_ha为my_nn。

参数	描述
dfs.namenode.rpc-address.xxx.nn	指定NameNode的RPC地址信息。其中nn表示dfs.ha.namenodes.xxx中配置的NameNode的名称。 例如，设置dfs.namenode.rpc-address.my_ha.my_nn参数值的格式为host:port。
dfs.client.failover.proxy.provider	指定Client连接NameNode的Provider，默认值为org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider。

示例如下。

```
(
    "dfs.nameservices" = "my-ha",
    "dfs.ha.namenodes.my-ha" = "my-namenode1,my-namenode2",
    "dfs.namenode.rpc-address.my-ha.my-namenode1" = "nn1-host:rpc_port",
    "dfs.namenode.rpc-address.my-ha.my-namenode2" = "nn2-host:rpc_port",
    "dfs.client.failover.proxy.provider" = "org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider"
)
```

HA模式可以与简单认证、Kerberos认证两种认证方式组合，进行集群访问。例如，通过简单认证方式访问HA HDFS。

```
(
    "username"="user",
    "password"="passwd",
    "dfs.nameservices" = "my-ha",
    "dfs.ha.namenodes.my-ha" = "my_namenode1,my_namenode2",
    "dfs.namenode.rpc-address.my-ha.my-namenode1" = "nn1-host:rpc_port",
    "dfs.namenode.rpc-address.my-ha.my-namenode2" = "nn2-host:rpc_port",
    "dfs.client.failover.proxy.provider" = "org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider"
)
```

HDFS集群的配置可以写入*hdfs-site.xml*文件中，您使用Broker进程读取HDFS集群的信息时，只需要填写集群的文件路径名和认证信息即可。

查看Broker实例

创建导入任务

查看导入任务状态

取消导入任务

导入示例

1. 创建测试表，下面是tpch的lineitem。

```
CREATE TABLE lineitem (
  l_orderkey bigint,
  l_partkey bigint,
  l_suppkey bigint,
  l_linenumber int,
  l_quantity double,
  l_extendedprice double,
  l_discount double,
  l_tax double,
  l_returnflag string,
  l_linestatus string,
  l_shipdate date,
  l_commitdate date,
  l_receiptdate date,
  l_shipinstruct string,
  l_shipmode string,
  l_comment string
)
ENGINE=OLAP
DUPLICATE KEY(l_orderkey)
DISTRIBUTED BY HASH(l_orderkey) BUCKETS 96
PROPERTIES(
  "replication_num" = "1"
);
```

2. 创建导入任务。

```
LOAD LABEL tpch.lineitem
(
    DATA INFILE("oss://xxx/tpc_h/sf1/lineitem.tbl")
    INTO TABLE `lineitem`
    COLUMNS TERMINATED BY '|'
    (l_orderkey, l_partkey, l_suppkey, l_linenumber, l_quantity, l_extendedprice, l_discount, l_tax, l_returnflag, l_li
nestatus, l_shipdate, l_commitdate, l_receiptdate, l_shipinstruct, l_shipmode, l_comment)
)
WITH BROKER broker
(
    "fs.oss.accessKeyId" = "xxx",
    "fs.oss.accessKeySecret" = "xxx",
    "fs.oss.endpoint" = "xxx"
);
```

3. 查看导入任务状态。

```
show load where label = 'lineitem'\G;
*************************** 1. row ***************************
         JobId: 12826
         Label: lineitem
         State: FINISHED
      Progress: ETL:100%; LOAD:100%
          Type: BROKER
       EtlInfo: unselected.rows=0; dpp.abnorm.ALL=0; dpp.norm.ALL=6001215
      TaskInfo: cluster:N/A; timeout(s):14400; max_filter_ratio:0.0
      ErrorMsg: NULL
    CreateTime: 2022-04-13 15:07:53
  EtlStartTime: 2022-04-13 15:07:56
 EtlFinishTime: 2022-04-13 15:07:56
 LoadStartTime: 2022-04-13 15:07:56
LoadFinishTime: 2022-04-13 15:08:06
           URL: NULL
    JobDetails: {"Unfinished backends":{"97f1acd1-6e70-4699-9199-b1722020931a":[]},"ScannedRows":6001215,"TaskNumber":1
,"All backends":{"97f1acd1-6e70-4699-9199-b1722020931a":[10002,10003,10004,10005]},"FileNumber":1,"FileSize":753862072}
2 rows in set (0.00 sec)
```

4. 导入成功后进行查询操作。
   - 查询表lineitem中的行数。

     ```
     select count(*) from lineitem;
     ```

     返回信息如下所示。

     ```
     +----------+
     | count(*) |
     +----------+
     |  6001215 |
     +----------+
     1 row in set (0.03 sec)
     ```

   - 查询表lineitem中的前2行信息。

     ```
     select * from lineitem limit 2;
     ```

     返回信息如下所示。

```
+------------+-----------+-----------+--------------+------------+-----------------+------------+-------+------------
--+--------------+-------------+--------------+---------------+------------+--------------------------
-------------------+
| l_orderkey | l_partkey | l_suppkey | l_linenumber | l_quantity | l_extendedprice | l_discount | l_tax | l_returnfla
g | l_linestatus | l_shipdate | l_commitdate | l_receiptdate | l_shipinstruct | l_shipmode | l_comment
|
+------------+-----------+-----------+--------------+------------+-----------------+------------+-------+------------
--+--------------+-------------+--------------+---------------+------------+--------------------------
-------------------+
|         69 |    115209 |      7721 |            1 |         48 |         58761.6 |       0.01 |  0.07 | A
| F            | 1994-08-17 | 1994-08-11   | 1994-09-08    | NONE           | TRUCK      | regular epitaphs. carefull
y even ideas hag |
|         69 |    104180 |      9201 |            2 |         32 |        37893.76 |       0.08 |  0.06 | A
| F            | 1994-08-24 | 1994-08-17   | 1994-08-31    | NONE           | REG AIR    | s sleep carefully bold,
|
+------------+-----------+-----------+--------------+------------+-----------------+------------+-------+------------
--+--------------+-------------+--------------+---------------+------------+--------------------------
-------------------+
2 rows in set (0.01 sec)
```

导入任务并发度

一个作业可以拆成一个或者多个任务，任务之间并行执行。拆分由LOAD语句中的DataDescription来决定。例如：

- 多个DataDescription对应导入多个不同的表，每个会拆成一个任务。
- 多个DataDescription对应导入同一个表的不同分区，每个也会拆成一个任务。

每个任务还会拆分成一个或者多个实例，然后将这些实例平均分配到BE上并行执行。实例的拆分由以下FE配置决定：

- min_bytes_per_broker_scanner：单个实例处理的最小数据量，默认值为64 MB。
- max_broker_concurrency：单个任务最大并发实例数，默认值为100。
- load_parallel_instance_num：单个BE上并发实例数，默认值为1个。

实例总数的计算公式为 `实例的总数 = min（导入文件总大小/单个实例处理的最小数据量，单个任务最大并发实例数，单个BE上并发实例数 * BE数）` 。

通常情况下，一个作业只有一个DataDescription，只会拆分成一个任务。任务会拆成与BE数相等的实例，然后分配到所有BE上并行执行。

常见问题

- Q：数据质量问题报错 `ETL_QUALITY_UNSATISFIED; msg:quality not good enough to cancel` 。

  A：处理方法请参见常见问题。

- Q：导入报错 `failed to send batch 或 TabletWriter add batch with unknown id` 。

  A：处理方法请参见通用系统配置中的BE配置，适当修改参数query_timeout和streaming_load_rpc_max_alive_time_sec。

- Q：导入报错 `LOAD-RUN-FAIL; msg:OrcScannerAdapter::init_include_columns. col name = xxx not found` 。

  A：如果是Parquet或者ORC格式的数据，需要保持文件头的列名与StarRocks表中的列名一致。示例如下。

  ```
  (tmp_c1,tmp_c2)
  SET
  (
     id=tmp_c2,
     name=tmp_c1
  )
  ```

  表示将Parquet或ORC文件中以（tmp_c1, tmp_c2）为列名的列，映射到StarRocks表中的（id, name）列。如果没有设置 `SET` ，则以column中的列作为映射。

  > 注意 如果使用某些Hive版本直接生成的ORC文件，ORC文件中的表头并非Hive meta数据，而是（_col0, _col1, _col2, ...），可能导致Invalid Column Name错误，则需要使用 `SET` 进行映射。

- Q：如何配置Hadoop ViewFS （federation）？

  A：需要将ViewFs相关的配置文件*core-site.xml*和*hdfs-site.xml*拷贝到*broker/conf*目录中。如果有自定义的FileSystem，则需要将相关的JAR拷贝到*broker/lib*目录中。

- Q：访问开启Kerberos认证的集群时，报错 `Can't get Kerberos realm` 。

  A：首先检查是不是所有的Broker所在机器都配置了*/etc/krb5.conf*文件。如果都配置了仍然报错，需要在Broker的启动脚本中的JAVA_OPTS变量最后，加上 `-Djava.security.krb5.conf:/etc/krb5.conf` 。

- Q：长时间等待，该如何处理？

  A：您可以在BE的*log/be.INFO*中，搜索error，了解一下具体原因。

5.3.3.3. Spark Load

Spark Load通过外部的Spark资源实现对导入数据的预处理，提高StarRocks大数据量的导入性能并且节省StarRocks集群的计算资源。Spark Load主要用于初次迁移、大数据量导入StarRocks的场景（数据量可到TB级别）。本文为您介绍Spark Load导入的基本概念、基本原理、使用示例、最佳实践以及常见问题。

背景信息

Spark Load是一种异步导入方式，您需要通过MySQL协议创建Spark类型导入任务，并可以通过 `SHOW LOAD` 命令查看导入结果。

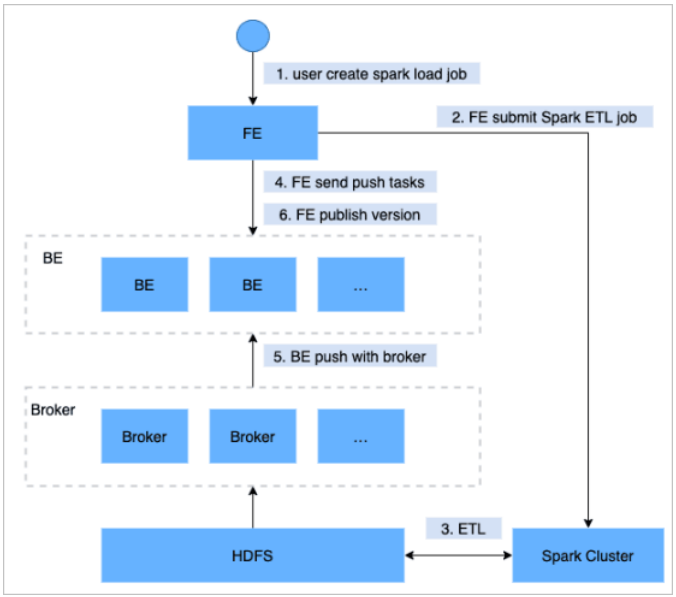
基本概念

- Spark ETL：在导入流程中主要负责数据的ETL工作，包括全局字典构建（BITMAP类型）、分区、排序和聚合等。
- Broker：是一个独立的无状态进程。封装了文件系统接口，提供StarRocks读取远端存储系统中文件的能力。
- 全局字典：保存了数据从原始值到编码值映射的数据结构，原始值可以是任意数据类型，而编码后的值为整型。全局字典主要应用于精确去重预计算的场景。

基本原理

用户通过MySQL客户端提交Spark类型导入任务，FE记录元数据并返回用户提交成功。

Spark Load的主要流程如下图所示。



Spark Load任务的执行主要分为以下几个阶段：

1. 向FE提交Spark Load任务。
2. FE调度提交ETL任务到Spark集群执行。
3. Spark集群执行ETL完成对导入数据的预处理，包括全局字典构建（BITMAP类型）、分区、排序和聚合等。
4. ETL任务完成后，FE获取预处理过的每个分片的数据路径，并调度相关的BE执行Push任务。
5. BE通过Broker读取数据，转化为StarRocks存储格式。
6. FE调度生效版本，完成导入任务。

全局字典

目前StarRocks中BITMAP列是使用类库Roaringbitmap实现的，而Roaringbitmap的输入数据类型只能是整型，因此如果要在导入流程中实现对于BITMAP列的预计算，则需要将输入数据的类型转换成整型。在StarRocks现有的导入流程中，全局字典的数据结构是基于Hive表实现的，保存了原始值到编码值的映射。

1. 读取上游数据源的数据，生成一张Hive临时表，记为hive-table。
2. 从hive-table中抽取待去重字段的去重值，生成一张新的Hive表，记为distinct-value-table。
3. 新建一张全局字典表，记为dict-table。一列为原始值，一列为编码后的值。
4. 将distinct-value-table与dict-table进行LEFT JOIN，计算出新增的去重值集合，然后对这个集合使用窗口函数进行编码，此时去重列原始值就多了一列编码后的值，最后将这两列的数据写回dict-table。
5. 将dict-table与hive-table进行JOIN，完成hive-table中原始值替换成整型编码值的工作。
6. hive-table会被下一步数据预处理的流程所读取，经过计算后导入到StarRocks中。

适用场景

构建流程

数据预处理

数据预处理的基本流程如下：

1. 从数据源读取数据，上游数据源可以是HDFS文件，也可以是Hive表。
2. 对读取到的数据完成字段映射、表达式计算，并根据分区信息生成分桶字段bucket-id。
3. 根据StarRocks表的Rollup元数据生成RollupTree。
4. 遍历RollupTree，进行分层的聚合操作，下一个层级的Rollup可以由上一个层的Rollup计算得来。
5. 每次完成聚合计算后，会根据bucket-id对数据进行分桶然后写入HDFS中。
6. 后续Broker会拉取HDFS中的文件然后导入StarRocks BE节点中。

基本操作

Spark作为一种外部计算资源在StarRocks中用来完成ETL工作，未来可能还有其他的外部资源会加入到StarRocks中使用。例如，Spark或GPU用于查询，HDFS或S3用于外部存储，MapReduce用于ETL等，因此引入Resource Management来管理StarRocks使用的这些外部资源。

提交Spark导入任务之前，需要配置执行ETL任务的Spark集群。操作语法如下所示。

```
-- create spark resource
CREATE EXTERNAL RESOURCE resource_name
PROPERTIES
(
 type = spark,
 spark_conf_key = spark_conf_value,
 working_dir = path,
 broker = broker_name,
 broker.property_key = property_value
);
-- drop spark resource
DROP RESOURCE resource_name;
-- show resources
SHOW RESOURCES
SHOW PROC "/resources";
-- privileges
GRANT USAGE_PRIV ON RESOURCE resource_name TO user_identityGRANT USAGE_PRIV ON RESOURCE resource_name TO ROLE role_name;
REVOKE USAGE_PRIV ON RESOURCE resource_name FROM user_identityREVOKE USAGE_PRIV ON RESOURCE resource_name FROM ROLE role_na
me;
```

- 创建资源

  resource-name为StarRocks中配置的Spark资源的名字。

  PROPERTIES是Spark资源的相关参数，参数描述如下表所示，更多参数描述请参见Spark Configuration。

参数		描述
type		资源类型。必填参数，目前仅支持Spark。
Spark相关参数	spark.master	必填参数，目前支持yarn。
	spark.submit.deployMode	Spark程序的部署模式。必填参数，支持cluster和client两种。
	spark.hadoop.fs.defaultFS	Master为YARN时必填。
	spark.hadoop.yarn.resourcemanager.address	单点Resource Manager地址。
	spark.hadoop.yarn.resourcemanager.ha.enabled	Resource Manager启用HA。默认值为true。
	spark.hadoop.yarn.resourcemanager.ha.rm-ids	Resource Manager逻辑ID列表。
	spark.hadoop.yarn.resourcemanager.hostname.rm-id	对于每个rm-id，指定Resource Manager对应的主机名。 **说明** HA Resource Manager只需配置spark.hadoop.yarn.resourcemanager.hostname.rm-id或spark.hadoop.yarn.resourcemanager.address.rm-id中的任意一个。

参数		描述
	spark.hadoop.yarn.resourcemanager.address.rm-id	对于每个rm-id，指定host:port以供客户端提交作业。 **说明** HA Resource Manager只需配置spark.hadoop.yarn.resourcemanager.hostname.rm-id或spark.hadoop.yarn.resourcemanager.address.rm-id中的任意一个。
working_dir		ETL使用的目录。 **说明** Spark作为ETL资源使用时必填。例如，*hdfs://host:port/tmp/starrocks*。
broker		Broker名字。 **说明** Spark作为ETL资源使用时必填。需要使用 `ALTER SYSTEM ADD BROKER` 命令提前完成配置。
broker.property_key		Broker读取ETL生成中间文件时需要指定的认证信息等。

创建资源示例如下所示。

```
-- yarn cluster模式
CREATE EXTERNAL RESOURCE "spark0"
PROPERTIES
(
    "type" = "spark",
    "spark.master" = "yarn",
    "spark.submit.deployMode" = "cluster",
    "spark.jars" = "xxx.jar,yyy.jar",
    "spark.files" = "/tmp/aaa,/tmp/bbb",
    "spark.executor.memory" = "1g",
    "spark.yarn.queue" = "queue0",
    "spark.hadoop.yarn.resourcemanager.address" = "resourcemanager_host:8032",
    "spark.hadoop.fs.defaultFS" = "hdfs://namenode_host:9000",
    "working_dir" = "hdfs://namenode_host:9000/tmp/starrocks",
    "broker" = "broker0",
    "broker.username" = "user0",
    "broker.password" = "password0"
);
-- yarn HA cluster模式
CREATE EXTERNAL RESOURCE "spark1"
PROPERTIES
(
    "type" = "spark",
    "spark.master" = "yarn",
    "spark.submit.deployMode" = "cluster",
    "spark.hadoop.yarn.resourcemanager.ha.enabled" = "true",
    "spark.hadoop.yarn.resourcemanager.ha.rm-ids" = "rm1,rm2",
    "spark.hadoop.yarn.resourcemanager.hostname.rm1" = "host1",
    "spark.hadoop.yarn.resourcemanager.hostname.rm2" = "host2",
    "spark.hadoop.fs.defaultFS" = "hdfs://namenode_host:9000",
    "working_dir" = "hdfs://namenode_host:9000/tmp/starrocks",
    "broker" = "broker1"
);
-- HDFS HA cluster模式
CREATE EXTERNAL RESOURCE "spark2"
PROPERTIES
(
    "type" = "spark",
    "spark.master" = "yarn",
    "spark.hadoop.yarn.resourcemanager.address" = "resourcemanager_host:8032",
    "spark.hadoop.fs.defaultFS" = "hdfs://myha",
    "spark.hadoop.dfs.nameservices" = "myha",
    "spark.hadoop.dfs.ha.namenodes.myha" = "mynamenode1,mynamenode2",
    "spark.hadoop.dfs.namenode.rpc-address.myha.mynamenode1" = "nn1_host:rpc_port",
    "spark.hadoop.dfs.namenode.rpc-address.myha.mynamenode2" = "nn2_host:rpc_port",
    "spark.hadoop.dfs.client.failover.proxy.provider" = "org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProx
yProvider",
    "working_dir" = "hdfs://myha/tmp/starrocks",
    "broker" = "broker2",
    "broker.dfs.nameservices" = "myha",
    "broker.dfs.ha.namenodes.myha" = "mynamenode1,mynamenode2",
    "broker.dfs.namenode.rpc-address.myha.mynamenode1" = "nn1_host:rpc_port",
    "broker.dfs.namenode.rpc-address.myha.mynamenode2" = "nn2_host:rpc_port",
    "broker.dfs.client.failover.proxy.provider" = "org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvi
der"
);
```

- 查看资源

  普通账户只能看到自己有USAGE-PRIV使用权限的资源。root和admin账户可以看到所有的资源。

- 资源权限

  资源权限通过GRANT REVOKE来管理，目前仅支持USAGE-PRIV使用权限。您可以将USAGE-PRIV权限赋予某个用户或者某个角色，角色的使用与之前一致。示例如下。

```
-- 授予spark0资源的使用权限给用户user0
GRANT USAGE_PRIV ON RESOURCE "spark0" TO "user0"@"%";
-- 授予spark0资源的使用权限给角色role0
GRANT USAGE_PRIV ON RESOURCE "spark0" TO ROLE "role0";
-- 授予所有资源的使用权限给用户user0
GRANT USAGE_PRIV ON RESOURCE * TO "user0"@"%";
-- 授予所有资源的使用权限给角色role0
GRANT USAGE_PRIV ON RESOURCE * TO ROLE "role0";
-- 撤销用户user0的spark0资源使用权限
REVOKE USAGE_PRIV ON RESOURCE "spark0" FROM "user0"@"%";
```

FE底层通过执行spark-submit命令提交Spark任务，因此需要为FE配置Spark客户端，建议使用官方2.4.5或以上版本的Spark 2.x，Spark下载地址下载完成后，请按照以下步骤完成配置：

1. 配置SPARK-HOME环境变量

   将Spark客户端放在FE同一台机器上的目录下，并在FE的配置文件中配置spark_home_default_dir指向此目录，此配置项的值默认为FE根目录下的*lib/spark2x*路径，此配置项不可为空。

2. 配置Spark依赖包

   将Spark客户端下的*jars*文件夹内所有JAR包归档打包成一个ZIP文件，并在FE的配置文件中配置spark_resource_path指向此ZIP文件。如果此配置项为空，则FE会尝试寻找FE根目录下的*lib/spark2x/jars/spark-2x.zip*文件，如果没有找到则会报文件不存在的错误。

   当提交Spark Load任务时，会将归档好的依赖文件上传至远端仓库，默认仓库路径挂在*working_dir/{cluster_id}*目录下，并以 `--spark-repository--{resource-name}` 命名，表示集群内的一个Resource对应一个远端仓库，远端仓库目录结构参考如下。

   ```
   ---spark-repository--spark0/
      |---archive-1.0.0/
      |   |---lib-990325d2c0d1d5e45bf675e54e44fb16-spark-dpp-1.0.0-jar-with-dependencies.jar
      |   |---lib-7670c29daf535efe3c9b923f778f61fc-spark-2x.zip
      |---archive-1.1.0/
      |   |---lib-64d5696f99c379af2bee28c1c84271d5-spark-dpp-1.1.0-jar-with-dependencies.jar
      |   |---lib-1bbb74bb6b264a270bc7fca3e964160f-spark-2x.zip
      |---archive-1.2.0/
      |   |-...
   ```

除了Spark依赖（默认以spark-2x.zip命名），FE还会上传DPP的依赖包至远端仓库。如果此次Spark Load提交的所有依赖文件都已存在远端仓库，则不需要再上传依赖，节省下了每次重复上传大量文件的时间。

FE底层通过YARN命令获取正在运行的Application的状态，以及终止Application，因此需要为FE配置YARN客户端，建议使用官方2.5.2或以上版本的Hadoop 2.x。Hadoop下载地址，下载完成后，请按照以下步骤完成配置：

1. 配置YARN可执行文件路径

   将下载好的YARN客户端放在FE同一台机器的目录下，并在FE配置文件中配置yarn_client_path参数，指向YARN的二进制可执行文件，默认为FE根目录下的*lib/yarn-client/hadoop/bin/yarn*路径。

2. 配置生成YARN所需的配置文件的路径（可选）

   当FE通过YARN客户端获取Application的状态，或者终止Application时，默认会在FE根目录下的*lib/yarn-config*路径下生成执行yarn命令所需的配置文件，此路径可以通过在FE配置文件配置*yarn_config_dir*参数修改，目前生成的配置文件包括*core-site.xml*和*yarn-site.xml*。

- 创建语法

```
LOAD LABEL load_label
    (data_desc, ...)
WITH RESOURCE resource_name
[resource_properties]
[PROPERTIES (key1=value1, ... )]
* load_label:
    db_name.label_name
* data_desc:
    DATA INFILE ('file_path', ...)
    [NEGATIVE]
    INTO TABLE tbl_name
    [PARTITION (p1, p2)]
    [COLUMNS TERMINATED BY separator ]
    [(col1, ...)]
    [COLUMNS FROM PATH AS (col2, ...)]
    [SET (k1=f1(xx), k2=f2(xx))]
    [WHERE predicate]
    DATA FROM TABLE hive_external_tbl
    [NEGATIVE]
    INTO TABLE tbl_name
    [PARTITION (p1, p2)]
    [SET (k1=f1(xx), k2=f2(xx))]
    [WHERE predicate]
* resource_properties:
 (key2=value2, ...)
```

创建导入的详细语法可以执行 `HELP SPARK LOAD` 命令查看帮助。Spark Load的创建导入语法中参数意义如下：

- Label

  导入任务的标识。每个导入任务，都有一个在单DataBase内部唯一的Label。具体规则与Broker Load一致。

- 数据描述类参数

  目前支持的数据源有CSV和Hive table。其他规则与Broker Load一致。

- 导入作业参数

  导入作业参数主要指的是Spark Load创建导入语句中的属于opt_properties部分的参数。导入作业参数是作用于整个导入作业的。规则与Broker Load一致。

- Spark资源参数

  Spark资源需要提前配置到StarRocks系统中并且赋予用户USAGE-PRIV权限后才能使用Spark Load。当您有临时性的需求，例如增加任务使用的资源而修改Spark configs时，可以设置以下参数，设置仅对本次任务生效，并不影响StarRocks集群中已有的配置。

  ```
  WITH RESOURCE 'spark0'
  (
     "spark.driver.memory" = "1g",
     "spark.executor.memory" = "3g"
  )
  ```

- 数据源为Hive表时的导入

  如果期望在导入流程中将Hive表作为数据源，则需要先新建一张类型为Hive的外部表，然后提交导入命令时指定外部表的表名即可。

- 导入流程构建全局字典

  适用于StarRocks表聚合列的数据类型为BITMAP类型。在Load命令中指定需要构建全局字典的字段即可，格式为 `StarRocks字段名称=bitmap_dict（hive表字段名称）` 。

  > **注意** 目前只有在上游数据源为Hive表时才支持全局字典的构建。

- 示例：

- 上游数据源为HDFS文件时创建导入任务的情况

```
LOAD LABEL load_label
    (data_desc, ...)
WITH RESOURCE resource_name
[resource_properties]
[PROPERTIES (key1=value1, ... )]
* load_label:
    db_name.label_name
* data_desc:
    DATA INFILE ('file_path', ...)
    [NEGATIVE]
    INTO TABLE tbl_name
    [PARTITION (p1, p2)]
    [COLUMNS TERMINATED BY separator ]
    [(col1, ...)]
    [COLUMNS FROM PATH AS (col2, ...)]
    [SET (k1=f1(xx), k2=f2(xx))]
    [WHERE predicate]
    DATA FROM TABLE hive_external_tbl
    [NEGATIVE]
    INTO TABLE tbl_name
    [PARTITION (p1, p2)]
    [SET (k1=f1(xx), k2=f2(xx))]
    [WHERE predicate]
* resource_properties:
 (key2=value2, ...)
```

- 上游数据源是Hive表时创建导入任务的情况
  a. 新建Hive资源。

```
CREATE EXTERNAL RESOURCE hive0
properties
(
    "type" = "hive",
    "hive.metastore.uris" = "thrift://emr-header-1.cluster-xxx:9083"
);
```

  b. 新建Hive外部表。

```
CREATE EXTERNAL TABLE hive_t1
(
    k1 INT,
    K2 SMALLINT,
    k3 varchar(50),
    uuid varchar(100)
)
ENGINE=hive
properties
(
    "resource" = "hive0",
    "database" = "tmp",
    "table" = "t1"
);
```

  c. 提交load命令，要求导入的StarRocks表中的列必须在Hive外部表中存在。

```
LOAD LABEL db1.label1
(
    DATA FROM TABLE hive_t1
    INTO TABLE tbl1
    SET
    (
        uuid=bitmap_dict(uuid)
    )
)
WITH RESOURCE 'spark0'
(
    "spark.executor.memory" = "2g",
    "spark.shuffle.compress" = "true"
)
PROPERTIES
(
    "timeout" = "3600"
);
```

Spark Load和Broker Load都是异步导入方式。您必须将创建导入的Label记录下来，并且在 `SHOW LOAD` 命令中使用此Label来查看导入结果。查看导入的命令在所有导入方式中是通用的，具体语法可执行 `HELP SHOW LOAD` 命令查看。

执行以下命令，查看导入任务。

```
show load order by createtime desc limit 1\G
```

返回信息如下。

```
*************************** 1. row ***************************
  JobId: 76391
  Label: label1
  State: FINISHED
 Progress: ETL:100%; LOAD:100%
  Type: SPARK
 EtlInfo: unselected.rows=4; dpp.abnorm.ALL=15; dpp.norm.ALL=28133376
 TaskInfo: cluster:cluster0; timeout(s):10800; max_filter_ratio:5.0E-5
 ErrorMsg: N/A
 CreateTime: 2019-07-27 11:46:42
 EtlStartTime: 2019-07-27 11:46:44
 EtlFinishTime: 2019-07-27 11:49:44
 LoadStartTime: 2019-07-27 11:49:44
LoadFinishTime: 2019-07-27 11:50:16
   URL: http://1.1.1.1:8089/proxy/application_1586619723848_0035/
 JobDetails: {"ScannedRows":28133395,"TaskNumber":1,"FileNumber":1,"FileSize":200000}
```

返回结果中涉及到的参数如下表所示。

参数	描述
State	导入任务当前所处的阶段。 任务提交之后状态为PENDING，提交Spark ETL之后状态变为ETL，ETL完成之后FE调度BE执行push操作，状态变为LOADING，push完成并且版本生效后状态变为FINISHED。 导入任务的最终阶段有CANCELLED和FINISHED两个，当Load Job处于这两个阶段时导入完成。其中CANCELLED为导入失败，FINISHED为导入成功。
Progress	导入任务的进度描述。包括ETL和LOAD两种进度，对应了导入流程的ETL和LOADING两个阶段。 LOAD的进度范围为0~100%。 `LOAD进度 = 当前已完成所有replica导入的tablet个数 / 本次导入任务的总tablet个数* 100%` **说明** • 如果所有导入表均完成导入，此时LOAD的进度为99%，导入进入到最后生效阶段，整个导入完成后，LOAD的进度才会变为100%。 • 因为导入进度并不是线性的，所以如果一段时间内进度没有变化，并不代表导入没有在执行。
Type	导入任务的类型。Spark Load为SPARK。
CreateTime	导入任务的创建时间。
EtlStartTime	ETL阶段开始的时间。
EtlFinishTime	ETL阶段完成的时间。
LoadStartTime	LOADING阶段开始的时间。
LoadFinishTime	整个导入任务完成的时间。
JobDetails	显示作业的详细运行状态，包括导入文件的个数、总大小（字节）、子任务个数、已处理的原始行数等。示例如下。 `{"ScannedRows":139264,"TaskNumber":1,"FileNumber":1,"FileSize":940754064}`
URL	可以复制输入到浏览器，跳转至相应Application的Web页面。

其余返回结果集中参数含义可以参见Broker Load，详情请参见Broker Load。

Spark任务提交过程中产生的详细日志，日志默认保存在FE根目录下*log/spark_launcher_log*路径下，并以*spark-launcher-{load-job-id}-{label}.log*格式命名，日志会在此目录下保存一段时间，当FE元数据中的导入信息被清理时，相应的日志也会被清理，默认保存时间为3天。

当Spark Load作业状态不为CANCELLED或FINISHED时，您可以手动取消。取消时需要指定待取消导入任务的Label。取消导入命令语法可以执行`HELP CANCEL LOAD`命令查看。

配置ETL集群

配置Spark客户端

配置YARN客户端

创建导入任务

查看导入任务

查看Spark Launcher提交日志

取消导入任务

相关系统配置

以下配置属于Spark Load的系统级别配置，也就是作用于所有Spark Load导入任务的配置，主要通过修改*fe.conf*来调整配置值。

参数	描述
enable-spark-load	开启Spark Load和创建Resource功能。 默认值为false，表示关闭此功能。
spark-load-default-timeout-second	任务默认超时时间。 默认值为259200秒（3天）。
spark-home-default-dir	Spark客户端路径。 默认值为*fe/lib/spark2x*。
spark-launcher-log-dir	打包好的Spark依赖文件路径。 默认值为空。
spark-launcher-log-dir	Spark客户端的提交日志存放的目录。 默认值为*fe/log/spark-launcher-log*。
yarn-client-path	YARN二进制可执行文件路径。 默认值为*fe/lib/yarn-client/hadoop/bin/yarn*。
yarn-config-dir	YARN配置文件生成路径。 默认值为*fe/lib/yarn-config*。

最佳实践

使用Spark Load最适合的场景是原始数据在文件系统（HDFS）中，数据量在几十GB到TB级别。小数据量还是建议使用Stream Load或者Broker Load。

完整Spark Load导入示例，请参见03_sparkLoad2StarRocks.md。

常见问题

- Q：：报错 `When running with master 'yarn' either HADOOP-CONF-DIR or YARN-CONF-DIR must be set in the environment` 。

  A：使用Spark Load时没有在Spark客户端的*spark-env.sh*中配置HADOOP-CONF-DIR环境变量。

- Q：提交Spark Job时用到spark-submit命令，报错 `Cannot run program "xxx/bin/spark-submit": error=2, No such file or directory` 。

  A：使用Spark Load时spark_home_default_dir配置项没有指定，或者指定了错误的Spark客户端根目录。

- Q：报错 `File xxx/jars/spark-2x.zip does not exist` 。

  A：使用Spark Load时spark-resource-path配置项没有指向打包好的ZIP文件，可以检查指向文件路径和文件名词是否一致。

- Q：报错 `yarn client does not exist in path: xxx/yarn-client/hadoop/bin/yarn` 。

  A：使用Spark Load时yarn-client-path配置项没有指定YARN的可执行文件。

- Q：报错 `Cannot execute hadoop-yarn/bin/../libexec/yarn-config.sh` 。

  A：使用CDH的Hadoop时，需要配置HADOOP_LIBEXEC_DIR环境变量，由于*hadoop-yarn*和*hadoop*目录不同，默认*libexec*目录会找*hadoop-yarn/bin/../libexec*，而*libexec*在*hadoop*目录下。 `yarn application status` 命令获取Spark任务状态报错导致导入作业失败。

5.3.3.4. Stream Load

StarRocks支持从本地直接导入数据，支持CSV文件格式，数据量在10 GB以下。本文为您介绍Stream Load导入的基本原理、使用示例、最佳实践以及常见问题。

背景信息

Stream Load是一种同步的导入方式，通过发送HTTP请求将本地文件或数据流导入到StarRocks中。Stream Load同步执行导入并返回导入结果。您可以直接通过请求的返回值判断导入是否成功。

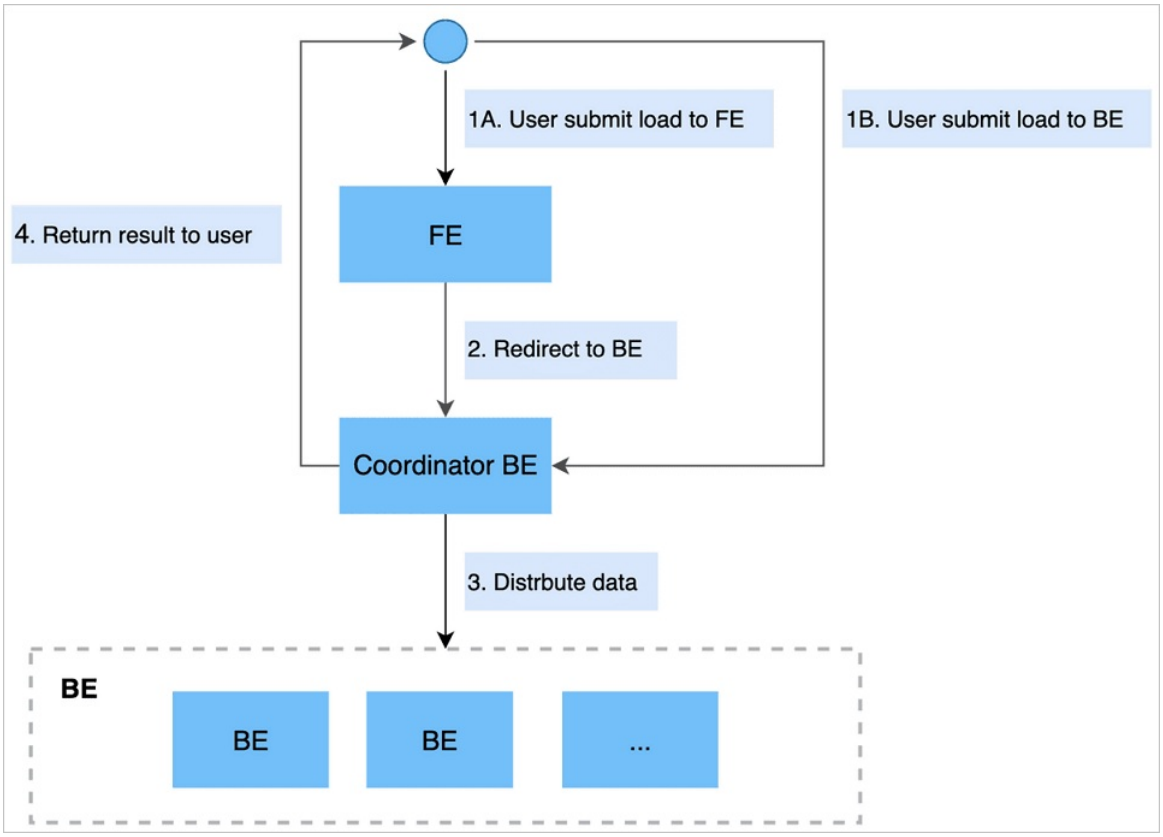
基本概念

Coordinator：协调节点。负责接收数据并分发数据到其他数据节点，导入完成后返回结果。

基本原理

Stream Load通过HTTP协议提交导入命令。如果提交到FE节点，则FE节点会通过HTTP Redirect指令将请求转发给某一个BE节点，您也可以直接提交导入命令给某一指定BE节点。该BE节点作为Coordinator节点，将数据按表Schema划分并分发数据到相关的BE节点。导入的最终结果由Coordinator节点返回给用户。

Stream Load的主要流程如下图所示。



导入示例

Stream Load通过HTTP协议提交和传输数据。本示例通过`curl`命令展示如何提交导入任务。您也可以通过其他HTTP Client进行操作。

- 语法

```
curl --location-trusted -u user:passwd [-H ""...] -T data.file -XPUT \
    http://fe_host:http_port/api/{db}/{table}/_stream_load
```

> 说明
> - 当前支持HTTP chunked与非chunked两种上传方式，对于非chunked方式，必须要有Content-Length来标示上传的内容长度，保证数据的完整性。
> - 建议设置Expect Header字段内容为100-continue，可以在某些出错场景下避免不必要的数据传输。

Header中支持的属性见下表的导入任务参数描述，格式为`-H "key1:value1"`。如果同时有多个任务参数，需要用多个-H来指示，类似于`-H "key1:value1" -H "key2:value2"……`。

创建导入任务的详细语法可以通过`HELP STREAM LOAD`命令查看。Stream Load中所有与导入任务相关的参数均设置在Header中。相关参数描述如下表所示。

参数		描述
签名参数	user:passwd	Stream Load创建导入任务使用的是HTTP协议，已通过Basic access authentication进行签名。StarRocks系统会根据签名来验证用户身份和导入权限。
导入任务参数	label	导入任务的标签，相同标签的数据无法多次导入。 您可以通过指定Label的方式来避免一份数据重复导入的问题。当前StarRocks系统会保留最近30分钟内成功完成的任务的Label。
	column_separator	用于指定导入文件中的列分隔符，默认为\t。 如果是不可见字符，则需要加\x作为前缀，使用十六进制来表示分隔符。例如，Hive文件的分隔符\x01，需要指定为 `-H "column_separator:\x01"` 。
	row_delimiter	指定导入文件中的行分隔符，默认为\n。 **注意** curl命令无法传递\n，换行符手动指定为\n时，shell会先传递反斜线（\），然后传递n而不是直接传递换行符\n。 Bash支持另一种转义字符串语法，传递\n和\t时，使用美元符号和全角单引号（$'）启动字符串并以半角单引号（'）结束字符串。例如， `-H $'row_delimiter:\n'` 。
	columns	用于指定导入文件中的列和Table中列的对应关系。 如果源文件中的列正好对应表中的内容，则无需指定该参数。如果源文件与表Schema不对应，则需要该参数来配置数据转换规则。列有两种形式，一种是直接对应于导入文件中的字段，可以直接使用字段名表示，一种需要通过计算得出。 ○ 示例1：表中有3列 `c1, c2, c3` ，源文件中的3列依次对应的是 `c3,c2,c1` ，则需要指定 `-H "columns: c3, c2, c1"` 。 ○ 示例2：表中有3列 `c1, c2, c3` ，源文件中前3列与表中的列一一对应，但是还有多余1列，则需要指定 `-H "columns: c1, c2, c3, temp"` ，最后1列随意指定名称用于占位即可。 ○ 示例3：表中有3列 `year, month, day` ，源文件中只有一个时间列，为2018-06-01 01:02:03格式，则可以指定 `-H "columns: col, year = year(col), month=month(col), day=day(col)"` 完成导入。
	where	用于抽取部分数据。用户如需将不需要的数据过滤掉，那么可以通过设定这个选项来达到。 例如，只导入k1列等于20180601的数据，则可以在导入时指定 `-H "where: k1 = 20180601"` 。
	max_filter_ratio	最大容忍可过滤（例如，因为数据不规范等原因而过滤）的数据比例。默认零容忍。 **说明** 此处数据不规范的数据不包括通过WHERE条件过滤的数据。
	partitions	用于指定该导入所涉及的Partition。 如果您能够确定数据对应的Partition，则推荐指定该项。不满足指定分区的数据将被过滤掉。例如，指定导入到p1和p2分区，可以指定 `-H "partitions: p1, p2"` 。
	timeout	指定导入的超时时间。默认是600秒。 设置范围为1~259200，单位为秒。
	strict_mode	指定此次导入是否开启严格模式，默认为开启。 关闭方式为 `-H "strict_mode: false"` 。
	timezone	指定本次导入所使用的时区。默认为东八区。 该参数会影响所有导入涉及和时区有关的函数结果。
	exec_mem_limit	导入内存限制。默认值为2 GB。

- 示例

```
curl --location-trusted -u root -T date -H "label:123" \
    http://abc.com:8030/api/test/date/_stream_load
```

- 返回结果

导入任务完成后，Stream Load会以JSON格式返回导入任务的相关内容，返回结果示例如下。

```
{
    "TxnId": 1003,
    "Label": "b6f3bc78-0d2c-45d9-9e4c-faa0a0149bee",
    "Status": "Success",
    "ExistingJobStatus": "FINISHED", // optional
    "Message": "OK",
    "NumberTotalRows": 1000000,
    "NumberLoadedRows": 1000000,
    "NumberFilteredRows": 1,
    "NumberUnselectedRows": 0,
    "LoadBytes": 40888898,
    "LoadTimeMs": 2144,
    "ErrorURL": "[http://192.168.1.1:8042/api/_load_error_log?file=__shard_0/error_log_insert_stmt_db18266d4d9b4ee5-abb00ddd64bdf005_db18266d4d9b4ee5_abb00ddd64bdf005](http://192.168.1.1:8042/api/_load_error_log?file=__shard_0/error_log_insert_stmt_db18266d4d9b4ee5-abb00ddd64bdf005_db18266d4d9b4ee5_abb00ddd64bdf005)"
}
```

参数	描述
TxnId	导入的事务ID。用户可不感知。
Status	导入任务最后的状态。
Success	表示任务导入成功，数据已经可见。
Publish Timeout	导入作业已经成功Commit，但是由于某种原因数据并不能立即可见。您可以认为导入任务已经成功无需重试导入。
Label Already Exists	表示该Label已经被其他作业占用，可能是导入成功，也可能是正在导入中。如果此次导入失败，您可以指定Label重试此次作业。
Message	导入状态的详细说明。导入失败时会返回具体的失败原因。
NumberTotalRows	从数据流中读取到的总行数。
NumberLoadedRows	导入任务的数据行数，仅在导入状态为Success时有效。
NumberFilteredRows	导入任务过滤掉的行数，即数据质量不合格的行。
NumberUnselectedRows	通过Where条件被过滤掉的行数。
LoadBytes	导入任务的源文件数据量大小。
LoadTimeMs	导入任务所用的时间，单位为ms。
ErrorURL	被过滤数据的具体内容，仅保留前1000条数据。如果导入任务失败，可以直接用以下方式获取被过滤的数据并进行分析，以调整导入任务。 `wget http://192.168.1.1:8042/api/_load_error_log?file=__shard_0/error_log_insert_stmt_db18266d4d9b4ee5-abb00ddd64bdf005_db18266d4d9b4ee5_abb00ddd64bdf005`

Stream Load无法手动取消，Stream Load在超时或者导入错误后会被系统自动取消。

创建导入任务

取消导入任务

最佳实践

Stream Load的最佳使用场景是原始文件在内存中或者存储在本地磁盘中。由于Stream Load是一种同步的导入方式，所以当您希望用同步方式获取导入结果时，也可以使用该导入方式。

由于Stream Load是由BE发起的导入并分发数据，建议的导入数据量在1 GB到10 GB之间。系统默认的最大Stream Load导入数据量为10 GB，所以导入超过10 GB的文件需要修改BE的配置项streaming_load_max_mb。例如，待导入文件大小为15 GB，则可以修改BE的配置项streaming_load_max_mb大于15 GB即可。

Stream Load的默认超时为300秒，按照StarRocks目前最大的导入限速来看，导入超过3 GB大小的文件就需要修改导入任务默认的超时时间了。例如，导入一个10 GB的文件，timeout应该设置为1000s。

`导入任务超时时间 = 导入数据量 / 10M/s`，具体的平均导入速度需要您根据自己的集群情况计算。

数据情况：数据在客户端本地磁盘路径*/home/store-sales*中，导入的数据量约为15 GB，希望导入到数据库bj-sales的表store-sales中。

集群情况：Stream Load的并发数不受集群大小影响。

示例如下：

1. 因为导入文件大小超过默认的最大导入大小10 GB，所以需要修改BE的配置文件*BE.conf*。

   例如，修改参数streaming_load_max_mb，将最大导入大小调整为16000。

2. 计算大概的导入时间是否超过默认timeout值，导入时间为 `15000 / 10 = 1500s` ，如果超过了默认的timeout时间，则需要修改FE的配置*FE.conf*，修改参数stream_load_default_timeout_second，将导入时间调整为1500。

3. 创建导入任务。

   ```
   curl --location-trusted -u user:password -T /home/store_sales \
       -H "label:abc" [http://abc.com:8000/api/bj_sales/store_sales/_stream_load](http://abc.com:8000/api/bj_sales/store_sales/_stream_load)
   ```

- Java开发Stream Load，详情请参见stream_load。
- Spark集成Stream Load，详情请参见01_sparkStreaming2StarRocks。

应用场景

数据量

完整示例

代码集成示例

常见问题

- Q：数据质量问题报错 `ETL_QUALITY_UNSATISFIED; msg:quality not good enough to cancel` 。

  A：处理方法请参见常见问题。

- Q：报错 `Label Already Exists` 。

  A：处理方法请参见常见问题。

  由于Stream Load是采用HTTP协议提交创建导入任务，通常各个语言的HTTP Client均会自带请求重试逻辑。StarRocks系统在接受到第一个请求后，已经开始操作Stream Load，但是由于没有及时向Client端返回结果，Client端会发生再次重试创建请求的情况。此时StarRocks系统由于已经在操作第一个请求，所以第二个请求会遇到Label Already Exists的情况。

  排查该问题的可能方法：使用Label搜索FE Master的日志，看是否存在同一个Label出现了两次的情况。如果存在则表示Client端重复提交了该请求。

  建议您根据当前请求的数据量，计算出大致的导入耗时，并根据导入超时时间调大Client端的请求超时时间，避免请求被Client端多次提交。

5.3.3.5. Routine Load

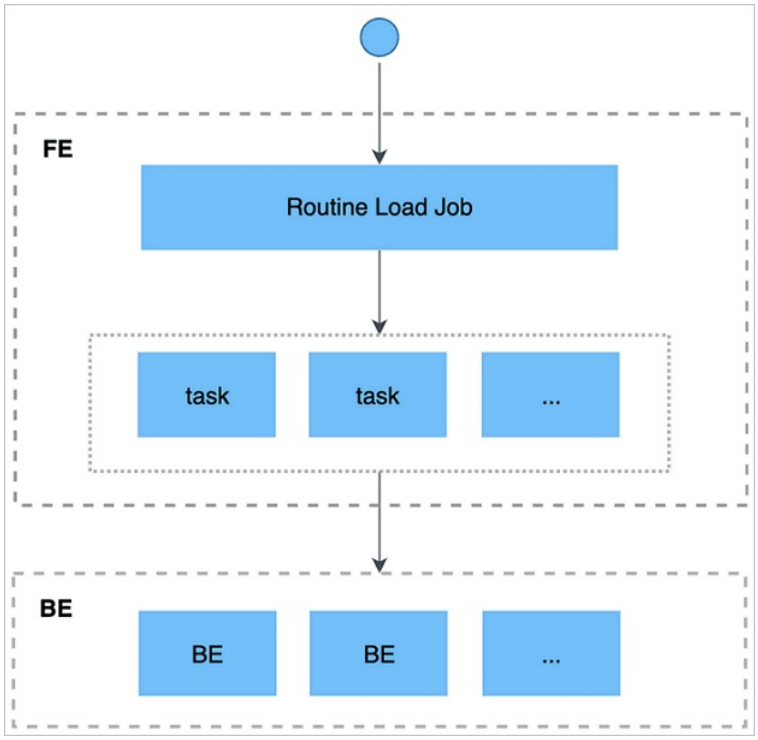
Routine Load是一种例行导入方式，StarRocks通过该方式支持从Kafka持续不断的导入数据，并且支持通过SQL控制导入任务的暂停、重启和停止。本文为您介绍Routine Load导入的基本原理、导入示例以及常见问题。

基本概念

- RoutineLoadJob：提交的一个例行导入任务。
- JobScheduler：例行导入任务调度器，用于调度和拆分一个RoutineLoadJob为多个Task。
- Task：RoutineLoadJob被JobScheduler根据规则拆分的子任务。
- TaskScheduler：任务调度器，用于调度Task的执行。

基本原理

Routine Load的导入流程如下图。



导入流程如下：

1. 用户通过支持MySQL协议的客户端向FE提交一个Kafka导入任务。
2. FE将一个导入任务拆分成若干个Task，每个Task负责导入指定的一部分数据。
3. 每个Task被分配到指定的BE上执行。在BE上，一个Task被视为一个普通的导入任务，通过Stream Load的导入机制进行导入。
4. BE导入完成后，向FE汇报。
5. FE根据汇报结果，继续生成后续新的Task，或者对失败的Task进行重试。
6. FE会不断的产生新的Task，来完成数据不间断的导入。

导入示例

- 支持访问无认证或使用SSL方式认证的Kafka集群。
- 支持的消息格式如下：
  - CSV文本格式，每一个message为一行，且行尾不包含换行符。
  - JSON文本格式。
- 不支持Array类型。
- 仅支持Kafka 0.10.0.0及以上版本。
- 语法

```
CREATE ROUTINE LOAD [database.][job_name] ON [table_name]
    [COLUMNS TERMINATED BY "column_separator" ,]
    [COLUMNS (col1, col2, ...) ,]
    [WHERE where_condition ,]
    [PARTITION (part1, part2, ...)]
    [PROPERTIES ("key" = "value", ...)]
    FROM [DATA_SOURCE]
    [(data_source_properties1 = 'value1',
    data_source_properties2 = 'value2',
    ...)]
```

相关参数描述如下表所示。

参数	是否必填	描述
job_name	是	导入作业的名称，前缀可以携带导入数据库名称，常见命名方式为时间戳+表名。一个DataBase内，任务名称不可重复。

参数	是否必填	描述
table_name	是	导入的目标表的名称。
COLUMN TERMINATED子句	否	指定源数据文件中的列分隔符，分隔符默认为\t。
COLUMN子句	否	指定源数据中列和表中列的映射关系。 • 映射列：例如，目标表有三列col1、col2和col3，源数据有4列，其中第1、2、4列分别对应col2、col1和col3，则书写为 `COLUMNS (col2, col1, temp, col3)` ，其中temp列为不存在的一列，用于跳过源数据中的第三列。 • 衍生列：除了直接读取源数据的列内容之外，StarRocks还提供对数据列的加工操作。例如，目标表后加入了第四列col4，其结果由col1 + col2产生，则可以书写为 `COLUMNS (col2, col1, temp, col3, col4 = col1 + col2)` 。
WHERE子句	否	指定过滤条件，可以过滤掉不需要的行。过滤条件可以指定映射列或衍生列。 例如，只导入k1大于100并且k2等于1000的行，则书写为 `WHERE k1 > 100 and k2 = 1000` 。
PARTITION子句	否	指定导入目标表的Partition。如果不指定，则会自动导入到对应的Partition中。
PROPERTIES子句	否	指定导入作业的通用参数。
desired_concurrent_number	否	导入并发度，指定一个导入作业最多会被分成多少个子任务执行。必须大于0，默认值为3。
max_batch_interval	否	每个子任务的最大执行时间。范围为5~60，单位是秒。默认值为10。 1.15版本后，该参数表示子任务的调度时间，即任务多久执行一次。任务的消费数据时间为*fe.conf*中的routine_load_task_consume_second，默认为3s。任务的执行超时时间为*fe.conf*中的routine_load_task_timeout_second，默认为15s。
max_batch_rows	否	每个子任务最多读取的行数。必须大于等于200000。默认值为200000。 1.15版本后，该参数只用于定义错误检测窗口范围，窗口的范围是10 * max-batch-rows。
max_batch_size	否	每个子任务最多读取的字节数。单位为字节，范围是100 MB到1 GB。默认值为100 MB。 1.15版本后，废弃该参数，任务消费数据的时间为*fe.conf*中的routine_load_task_consume_second，默认为3s。
max_error_number	否	采样窗口内，允许的最大错误行数。必须大于等于0。默认是0，即不允许有错误行。 **注意** 被WHERE条件过滤掉的行不算错误行。
strict_mode	否	是否开启严格模式，默认为开启。 如果开启后，非空原始数据的列类型变换为NULL，则会被过滤。关闭方式为设置该参数为false。
timezone	否	指定导入作业所使用的时区。 默认为使用Session的timezone参数。该参数会影响所有导入涉及的和时区有关的函数结果。
DATA_SOURCE	是	指定数据源，请使用KAFKA。
data_source_properties	否	指定数据源相关的信息。包括以下参数： • kafka_broker_list：Kafka的Broker连接信息，格式为 `ip:host` 。多个Broker之间以逗号（,）分隔。 • kafka_topic：指定待订阅的Kafka的Topic。 **说明** 如果指定数据源相关的信息，则kafka_broker_list和kafka_topic必填。 • kafka_partitions和kafka_offsets：指定需要订阅的Kafka Partition，以及对应的每个Partition的起始offset。 • property：Kafka相关的属性，功能等同于Kafka Shell中 `"--property"` 参数。创建导入任务更详细的语法可以通过执行 `HELP ROUTINE LOAD;` 命令查看。

说明 创建导入任务更详细的语法可以通过执行 `HELP ROUTINE LOAD;` 命令查看。

- 示例：从一个本地Kafka集群导入数据。

```
CREATE ROUTINE LOAD routine_load_wikipedia ON routine_wiki_edit
COLUMNS TERMINATED BY ",",
COLUMNS (event_time, channel, user, is_anonymous, is_minor, is_new, is_robot, is_unpatrolled, delta, added, deleted)
PROPERTIES
(
    "desired_concurrent_number"="1",
    "max_error_number"="1000"
)
FROM KAFKA
(
    "kafka_broker_list"= "localhost:9092",
    "kafka_topic" = "starrocks-load"
);
```

- 显示database下，所有的例行导入作业（包括已停止或取消的作业）。结果为一行或多行。

```
USE [database];
SHOW ALL ROUTINE LOAD;
```

- 显示database下，名称为job_name的当前正在运行的例行导入作业。

```
SHOW ROUTINE LOAD FOR [database].[job_name];
```

> 注意 StarRocks只能查看当前正在运行中的任务，已结束和未开始的任务无法查看。

查看任务状态的具体命令和示例，都可以通过`HELP SHOW ROUTINE LOAD`命令查看。查看任务运行状态（包括子任务）的具体命令和示例，可以通过`HELP SHOW ROUTINE LOAD TASK`命令查看。

执行`SHOW ALL ROUTINE LOAD`命令，可以查看当前正在运行的所有Routine Load任务，返回如下类似信息。

```
*************************** 1. row ***************************
                  Id: 14093
                Name: routine_load_wikipedia
          CreateTime: 2020-05-16 16:00:48
           PauseTime: N/A
             EndTime: N/A
              DbName: default_cluster:load_test
           TableName: routine_wiki_edit
               State: RUNNING
      DataSourceType: KAFKA
      CurrentTaskNum: 1
       JobProperties: {"partitions":"*","columnToColumnExpr":"event_time,channel,user,is_anonymous,is_minor,is_new,is_robot
,is_unpatrolled,delta,added,deleted","maxBatchIntervalS":"10","whereExpr":"*","maxBatchSizeBytes":"104857600","columnSepara
tor":"','","maxErrorNum":"1000","currentTaskConcurrentNum":"1","maxBatchRows":"200000"}
DataSourceProperties: {"topic":"starrocks-load","currentKafkaPartitions":"0","brokerList":"localhost:9092"}
    CustomProperties: {}
           Statistic: {"receivedBytes":150821770,"errorRows":122,"committedTaskNum":12,"loadedRows":2399878,"loadRowsRate":
199000,"abortedTaskNum":1,"totalRows":2400000,"unselectedRows":0,"receivedBytesRate":12523000,"taskExecuteTimeMs":12043}
            Progress: {"0":"13634667"}
ReasonOfStateChanged:
        ErrorLogUrls: http://172.26.**.**:9122/api/_load_error_log?file=__shard_53/error_log_insert_stmt_47e8a1d107ed4932-8
f1ddf7b01ad2fee_47e8a1d107ed4932_8f1ddf7b01ad2fee, http://172.26.**.**:9122/api/_load_error_log?file=__shard_54/error_log_i
nsert_stmt_e0c0c6b040c044fd-a162b16f6bad53e6_e0c0c6b040c044fd_a162b16f6bad53e6, http://172.26.**.**:9122/api/_load_error_lo
g?file=__shard_55/error_log_insert_stmt_ce4c95f0c72440ef-a442bb300bd743c8_ce4c95f0c72440ef_a442bb300bd743c8
            OtherMsg:
1 row in set (0.00 sec)
```

本示例创建名为routine_load_wikipedia的导入任务，相关参数描述如下表。

参数	描述
State	导入任务状态。RUNNING表示该导入任务处于持续运行中。
Statistic	进度信息，记录了从创建任务开始后的导入信息。
receivedBytes	接收到的数据大小，单位是Byte。
errorRows	导入错误行数。

参数	描述
committedTaskNum	FE提交的Task数。
loadedRows	已导入的行数。
loadRowsRate	导入数据速率，单位是行每秒（row/s）。
abortedTaskNum	BE失败的Task数。
totalRows	接收的总行数。
unselectedRows	被WHERE条件过滤的行数。
receivedBytesRate	接收数据速率，单位是Bytes/s。
taskExecuteTimeMs	导入耗时，单位是ms。
ErrorLogUrls	错误信息日志，可以通过URL看到导入过程中的错误信息。

使用PAUSE语句后，此时导入任务进入PAUSED状态，数据暂停导入，但任务未终止，可以通过RESUME语句重启任务。

例如，暂停名称为job_name的例行导入任务。

```
PAUSE ROUTINE LOAD FOR [job_name];
```

可以通过 `HELP PAUSE ROUTINE LOAD` 命令查看帮助和示例。

暂停导入任务后，任务的State变更为PAUSED，Statistic和Progress中的导入信息停止更新。此时，任务并未终止，通过 `SHOW ROUTINE LOAD` 语句可以看到已经暂停的导入任务。

使用RESUME语句后，任务会短暂的进入NEED_SCHEDULE状态，表示任务正在重新调度，一段时间后会重新恢复至RUNING状态，继续导入数据。

例如，恢复名称为job_name的例行导入任务。

```
RESUME ROUTINE LOAD FOR [job_name];
```

可以通过 `HELP RESUME ROUTINE LOAD` 命令查看帮助和示例。

执行 `SHOW ROUTINE LOAD` 命令，查看任务状态。返回信息如下。

```
*************************** 1. row ***************************
                  Id: 14093
                Name: routine_load_wikipedia
          CreateTime: 2020-05-16 16:00:48
           PauseTime: N/A
             EndTime: N/A
              DbName: default_cluster:load_test
           TableName: routine_wiki_edit
               State: NEED_SCHEDULE
      DataSourceType: KAFKA
      CurrentTaskNum: 0
       JobProperties: {"partitions":"*","columnToColumnExpr":"event_time,channel,user,is_anonymous,is_minor,is_new,is_robot
,is_unpatrolled,delta,added,deleted","maxBatchIntervalS":"10","whereExpr":"*","maxBatchSizeBytes":"104857600","columnSepara
tor":"','","maxErrorNum":"1000","currentTaskConcurrentNum":"1","maxBatchRows":"200000"}
DataSourceProperties: {"topic":"starrocks-load","currentKafkaPartitions":"0","brokerList":"localhost:9092"}
    CustomProperties: {}
           Statistic: {"receivedBytes":162767220,"errorRows":132,"committedTaskNum":13,"loadedRows":2589972,"loadRowsRate":
115000,"abortedTaskNum":7,"totalRows":2590104,"unselectedRows":0,"receivedBytesRate":7279000,"taskExecuteTimeMs":22359}
            Progress: {"0":"13824771"}
ReasonOfStateChanged:
        ErrorLogUrls: http://172.26.108.172:9122/api/_load_error_log?file=__shard_54/error_log_insert_stmt_e0c0c6b040c044fd
-a162b16f6bad53e6_e0c0c6b040c044fd_a162b16f6bad53e6, http://172.26.108.172:9122/api/_load_error_log?file=__shard_55/error_l
og_insert_stmt_ce4c95f0c72440ef-a442bb300bd743c8_ce4c95f0c72440ef_a442bb300bd743c8, http://172.26.108.172:9122/api/_load_er
ror_log?file=__shard_56/error_log_insert_stmt_8753041cd5fb42d0-b5150367a5175391_8753041cd5fb42d0_b5150367a5175391
           OtherMsg:
1 row in set (0.00 sec)
```

再一次执行 `SHOW ROUTINE LOAD` 命令，查看任务状态。返回信息如下。

```
*************************** 1. row ***************************
                  Id: 14093
                Name: routine_load_wikipedia
          CreateTime: 2020-05-16 16:00:48
           PauseTime: N/A
             EndTime: N/A
              DbName: default_cluster:load_test
           TableName: routine_wiki_edit
               State: RUNNING
      DataSourceType: KAFKA
      CurrentTaskNum: 1
       JobProperties: {"partitions":"*","columnToColumnExpr":"event_time,channel,user,is_anonymous,is_minor,is_new,is_robot
,is_unpatrolled,delta,added,deleted","maxBatchIntervalS":"10","whereExpr":"*","maxBatchSizeBytes":"104857600","columnSepara
tor":"'",'"',"maxErrorNum":"1000","currentTaskConcurrentNum":"1","maxBatchRows":"200000"}
DataSourceProperties: {"topic":"starrocks-load","currentKafkaPartitions":"0","brokerList":"localhost:9092"}
    CustomProperties: {}
           Statistic: {"receivedBytes":175337712,"errorRows":142,"committedTaskNum":14,"loadedRows":2789962,"loadRowsRate":
118000,"abortedTaskNum":7,"totalRows":2790104,"unselectedRows":0,"receivedBytesRate":7422000,"taskExecuteTimeMs":23623}
            Progress: {"0":"14024771"}
ReasonOfStateChanged:
        ErrorLogUrls: http://172.26.**.**:9122/api/_load_error_log?file=__shard_55/error_log_insert_stmt_ce4c95f0c72440ef-a
442bb300bd743c8_ce4c95f0c72440ef_a442bb300bd743c8, http://172.26.**.**:9122/api/_load_error_log?file=__shard_56/error_log_i
nsert_stmt_8753041cd5fb42d0-b5150367a5175391_8753041cd5fb42d0_b5150367a5175391, http://172.26.**.**:9122/api/_load_error_lo
g?file=__shard_57/error_log_insert_stmt_31304c87bb82431a-9f2baf7d5fd7f252_31304c87bb82431a_9f2baf7d5fd7f252
           OtherMsg:
1 row in set (0.00 sec)
ERROR: No query specified
```

> **说明** 第一次查询任务时，State变为NEED_SCHEDULE，表示任务正在重新调度。第二次查询任务时，State变为RUNING，同时Statistic和Progress中的导入信息开始更新，继续导入数据。

使用STOP语句让导入任务进入STOP状态，数据停止导入，任务终止，无法恢复数据导入。

例如，停止名称为job_name的例行导入任务。

```
STOP ROUTINE LOAD FOR [job_name];
```

您可以通过 `HELP STOP ROUTINE LOAD` 命令查看帮助和示例。

执行 `SHOW ROUTINE LOAD` 命令，查看任务状态。返回信息如下。

```
*************************** 1. row ***************************
                  Id: 14093
                Name: routine_load_wikipedia
          CreateTime: 2020-05-16 16:00:48
           PauseTime: N/A
             EndTime: 2020-05-16 16:08:25
              DbName: default_cluster:load_test
           TableName: routine_wiki_edit
               State: STOPPED
      DataSourceType: KAFKA
      CurrentTaskNum: 0
       JobProperties: {"partitions":"*","columnToColumnExpr":"event_time,channel,user,is_anonymous,is_minor,is_new,is_robot
,is_unpatrolled,delta,added,deleted","maxBatchIntervalS":"10","whereExpr":"*","maxBatchSizeBytes":"104857600","columnSepara
tor":"'",'"',"maxErrorNum":"1000","currentTaskConcurrentNum":"1","maxBatchRows":"200000"}
DataSourceProperties: {"topic":"starrocks-load","currentKafkaPartitions":"0","brokerList":"localhost:9092"}
    CustomProperties: {}
           Statistic: {"receivedBytes":325534440,"errorRows":264,"committedTaskNum":26,"loadedRows":5179944,"loadRowsRate":
109000,"abortedTaskNum":18,"totalRows":5180208,"unselectedRows":0,"receivedBytesRate":6900000,"taskExecuteTimeMs":47173}
            Progress: {"0":"16414875"}
ReasonOfStateChanged:
        ErrorLogUrls: http://172.26.**.**:9122/api/_load_error_log?file=__shard_67/error_log_insert_stmt_79e9504cafee4fbd-b
3981a65fb158cde_79e9504cafee4fbd_b3981a65fb158cde, http://172.26.**.**:9122/api/_load_error_log?file=__shard_68/error_log_i
nsert_stmt_b6981319ce56421b-bf4486c2cd371353_b6981319ce56421b_bf4486c2cd371353, http://172.26.**.**:9122/api/_load_error_lo
g?file=__shard_69/error_log_insert_stmt_1121400c1f6f4aed-866c381eb49c966e_1121400c1f6f4aed_866c381eb49c966e
           OtherMsg:
```

停止导入任务后，任务的State变更为STOPPED，Statistic和Progress中的导入信息再也不会更新。此时，通过 `SHOW ROUTINE LOAD` 语句无法看到已经停止的导入任务。

环境要求

创建导入任务

查看任务状态

暂停导入任务

恢复导入任务

停止导入任务

常见问题

Q：导入任务被PAUSE，报错 `Broker: Offset out of range` 。

A：通过 `SHOW ROUTINE LOAD` 命令查看最新的offset，通过Kafka客户端查看该offset有没有数据。如果没有数据，则可能原因如下：

- 导入时指定了未来的offset。
- 还没来得及导入，Kafka已经将该offset的数据清理。您需要根据StarRocks的导入速度设置合理的log清理参数log.retention.hours和log.retention.bytes等。

5.3.3.6. Insert Into

StarRocks中INSERT INTO语句的使用方式和MySQL等数据库中INSERT INTO语句的使用方式类似， 但在StarRocks中，所有的数据写入都是一个独立的导入作业 ，所以StarRocks中将INSERT INTO作为一种导入方式介绍。本文为您介绍Insert Into导入的使用场景、相关配置以及导入示例。

适用场景

- INSERT INTO导入会同步返回导入流程的运行结果。
- 如果仅导入几条测试数据，验证一下StarRocks系统的功能，则可以使用INSERT INTO VALUES语句。
- 如果将已经在StarRocks表中的数据进行ETL转换并导入到一个新的StarRocks表中，则可以使用INSERT INTO SELECT语句。
- 您可以创建一种外部表。例如，MySQL外部表映射一张MySQL系统中的表。然后通过INSERT INTO SELECT语句将外部表中的数据导入到StarRocks表中存储。

注意事项

- 当执行INSERT INTO语句时，对于不符合目标表格式的数据（例如，字符串超长），默认的行为是过滤。对于数据不能被过滤的业务场景，可以通过设置会话变量enable_insert_strict为true来确保当有数据被过滤的时候，INSERT INTO语句不会成功执行。
- 因为StarRocks的INSERT INTO复用导入数据的逻辑，所以每一次INSERT INTO语句都会产生一个新的数据版本。因为频繁小批量导入操作会产生过多的数据版本，而过多的小版本会影响查询的性能，所以不建议频繁的使用INSERT INTO语句导入数据或作为生产环境的日常例行导入任务。如果有流式导入或者小批量导入任务的需求，则可以使用Stream Load或者Routine Load的方式进行导入。

基本操作

```
INSERT INTO table_name
[ PARTITION (p1, ...) ]
[ WITH LABEL label]
[ (column [, ...]) ]
[ [ hint [, ...] ] ]
{ VALUES ( { expression | DEFAULT } [, ...] ) [, ...] | query }
```

参数描述如下表所示。

参数	描述
table_name	导入数据的目的表的名称。填写形式为db_name.table_name。
partitions	指定待导入的分区，必须是table_name表中存在的分区，多个分区名称用逗号（,）分隔。如果指定目标分区，则只会导入符合目标分区的数据。如果没有指定，则默认值为table_name表中所有的分区。
label	为insert作业指定一个Label，Label是该INSERT INTO导入作业的标识。每个导入作业，都有一个在单DataBase内部唯一的Label。 **注意** 建议指定Label，而不是由系统自动分配。如果由系统自动分配，在INSERT INTO语句执行过程中，因网络错误导致连接断开等，则无法得知INSERT INTO是否成功。如果指定Label，则可以再次通过Label查看任务结果。

参数	描述
column_name	指定的目的列，必须是table_name表中存在的列。 导入表的目标列，可以以任意的顺序存在。如果没有指定目标列，则默认值是table_name表的所有列。如果导入表中的某个列不在目标列中，则这个列需要有默认值，否则INSERT INTO会失败。如果查询语句的结果列类型与目标列的类型不一致，则会调用隐式类型转化，如果不能进行转化，则INSERT INTO语句会报语法解析错误。
expression	需要赋值给某个列的对应表达式。
default	让对应列使用默认值。
query	一个普通查询，查询的结果会写入到目标中。查询语句支持任意StarRocks支持的SQL查询语句。
values	通过VALUES语句插入一条或者多条数据。 注意 VALUES方式仅适用于导入几条数据作为Demo的情况，完全不适用于任何测试和生产环境。StarRocks系统本身也不适合单条数据导入的场景。建议使用INSERT INTO SELECT的方式进行批量导入。

INSERT INTO本身就是一个SQL命令，其返回结果如下所示：

- 执行成功
  - 示例1

    执行 `insert into tbl1 select * from empty_tbl;` 导入语句。返回结果如下。

    ```
    Query OK, 0 rows affected (0.02 sec)
    ```

  - 示例2

    执行 `insert into tbl1 select * from tbl2;` 导入语句。返回结果如下。

    ```
    Query OK, 4 rows affected (0.38 sec)
    {'label':'insert_8510c568-9eda-4173-9e36-6adc7d35291c', 'status':'visible', 'txnId':'4005'}
    ```

  - 示例3

    执行 `insert into tbl1 with label my_label1 select * from tbl2;` 导入语句。返回结果如下。

    ```
    Query OK, 4 rows affected (0.38 sec)
    {'label':'my_label1', 'status':'visible', 'txnId':'4005'}
    ```

  - 示例4

    执行 `insert into tbl1 select * from tbl2;` 导入语句。返回结果如下。

    ```
    Query OK, 2 rows affected, 2 warnings (0.31 sec)
    {'label':'insert_f0747f0e-7a35-46e2-affa-13a235f4020d', 'status':'visible', 'txnId':'4005'}
    ```

    或者返回如下结果。

    ```
    Query OK, 2 rows affected, 2 warnings (0.31 sec)
    {'label':'insert_f0747f0e-7a35-46e2-affa-13a235f4020d', 'status':'committed', 'txnId':'4005'}
    ```

    返回结果中涉及的参数描述如下表所示。

参数	描述
rows affected	表示总共有多少行数据被导入。
warnings	表示被过滤的行数。
status	导入数据是否可见。参数值如下： ○ visible：表示可见。 ○ committed：表示不可见。
txnId	insert对应的导入事务的ID。
err	显示一些非预期错误。当需要查看被过滤的行时，您可以使用如下语句。返回结果中的URL可以用于查询错误的数据。

- 执行失败

  执行失败表示没有成功导入任何数据。

  例如，执行 `insert into tbl1 select * from tbl2 where k1 = "a";` 导入语句。返回结果如下。

  ```
  ERROR 1064 (HY000): all partitions have no load data. url: [http://10.74.167.16:8042/api/_load_error_log?file=__shard_2/error_log_insert_stmt_ba8bb9e158e4879-ae8de8507c0bf8a2_ba8bb9e158e4879_ae8de8507c0bf8a2](http://10.74.**.**:8042/api/_load_error_log?file=__shard_2/error_log_insert_stmt_ba8bb9e158e4879-ae8de8507c0bf8a2_ba8bb9e158e4879_ae8de8507c0bf8a2)
  ```

  其中，返回信息中的 `ERROR 1064 (HY000): all partitions have no load data` 显示失败原因，后面的URL可以用于查询错误的数据。

语法

导入结果介绍

相关配置

timeout：导入任务的超时时间，单位是秒。导入任务在设定的timeout时间内未完成则会被系统取消，变成CANCELLED。目前INSERT INTO并不支持自定义导入的timeout时间，所有INSERT INTO导入的超时时间是统一的，默认的timeout时间为1小时。如果导入任务无法在规定时间内完成，则需要调整FE的insert_load_default_timeout_second参数。

参数	描述
enable_insert_strict	INSERT INTO导入本身不能控制导入可容忍的错误率。您只能通过Session参数enable_insert_strict来控制导入可容忍的错误率。 • true（默认值）：表示如果有一条数据错误，则导入失败。 • false：表示至少有一条数据被正确导入，则返回成功。如果有失败数据，则还会返回一个Label。 可通过 `SET enable_insert_strict = false` 来设置。
query_timeout	INSERT INTO本身也是一个SQL命令，因此INSERT INTO语句也受到Session量query_timeout的限制。可以通过 `SET query_timeout = xxx` 来增加超时时间，单位是秒。

FE配置

Session变量

导入示例

1. 执行以下命令，创建数据库。

   ```
   CREATE DATABASE IF NOT EXISTS load_test;
   ```

2. 执行以下命令，指定数据库。

   ```
   USE load_test;
   ```

3. 执行以下命令，创建数据表。

   ```
   CREATE TABLE insert_wiki_edit
   (
       event_time DATETIME,
       channel VARCHAR(32) DEFAULT '',
       user VARCHAR(128) DEFAULT '',
       is_anonymous TINYINT DEFAULT '0',
       is_minor TINYINT DEFAULT '0',
       is_new TINYINT DEFAULT '0',
       is_robot TINYINT DEFAULT '0',
       is_unpatrolled TINYINT DEFAULT '0',
       delta INT SUM DEFAULT '0',
       added INT SUM DEFAULT '0',
       deleted INT SUM DEFAULT '0'
   )
   AGGREGATE KEY(event_time, channel, user, is_anonymous, is_minor, is_new, is_robot, is_unpatrolled)
   PARTITION BY RANGE(event_time)
   (
       PARTITION p06 VALUES LESS THAN ('2015-09-12 06:00:00'),
       PARTITION p12 VALUES LESS THAN ('2015-09-12 12:00:00'),
       PARTITION p18 VALUES LESS THAN ('2015-09-12 18:00:00'),
       PARTITION p24 VALUES LESS THAN ('2015-09-13 00:00:00')
   )
   DISTRIBUTED BY HASH(user) BUCKETS 10
   PROPERTIES("replication_num" = "1");
   ```

执行以下命令，通过VALUES语句导入数据。

```
INSERT INTO insert_wiki_edit VALUES("2015-09-12 00:00:00","#en.wikipedia","GELongstreet",0,0,0,0,0,36,36,0),("2015-09-12 00
:00:00","#ca.wikipedia","PereBot",0,1,0,1,0,17,17,0);
```

返回信息如下。

```
Query OK, 2 rows affected (0.29 sec)
{'label':'insert_1f12c916-5ff8-4ba9-8452-6fc37fac2e75', 'status':'visible', 'txnId':'601'}
```

执行以下命令，通过SELECT语句导入数据。

```
INSERT INTO insert_wiki_edit WITH LABEL insert_load_wikipedia SELECT * FROM routine_wiki_edit;
```

返回信息如下。

```
Query OK, 18203 rows affected (0.40 sec)
{'label':'insert_load_wikipedia', 'status':'visible', 'txnId':'618'}
```

创建数据库与数据表

通过VALUES导入数据

通过SELECT导入数据

5.3.3.7. DataX Writer

DataX Writer插件实现了写入数据到StarRocks目的表的功能。在底层实现上，DataX Writer通过Stream Load以CSV或JSON格式导入数据至StarRocks。内部将Reader读取的数据进行缓存后批量导入至StarRocks，以提高写入性能。阿里云DataWorks已经集成了DataX导入的能力，可以同步MaxCompute数据到EMR StarRocks。本文为您介绍DataX Writer原理，以及如何使用DataWorks进行离线同步任务。

背景信息

DataX Writer总体的数据流为Source -> Reader -> DataX channel -> Writer -> StarRocks。

功能说明

环境准备

您可以下载DataX插件和DataX源码进行测试：

测试时可以使用命令 `python datax.py --jvm="-Xms6G -Xmx6G" --loglevel=debug job.json` 。

配置样例

从MySQL读取数据后导入至StarRocks。

```
{
    "job": {
        "setting": {
            "speed": {
                 "channel": 1
            },
            "errorLimit": {
                "record": 0,
                "percentage": 0
            }
        },
        "content": [
            {
                "reader": {
                    "name": "mysqlreader",
                    "parameter": {
                        "username": "xxxx",
                        "password": "xxxx",
                        "column": [ "k1", "k2", "v1", "v2" ],
                        "connection": [
                            {
                                "table": [ "table1", "table2" ],
                                "jdbcUrl": [
                                     "jdbc:mysql://127.0.0.1:3306/datax_test1"
                                ]
                            },
                            {
                                "table": [ "table3", "table4" ],
                                "jdbcUrl": [
                                     "jdbc:mysql://127.0.0.1:3306/datax_test2"
                                ]
                            }
                        ]
                    }
                },
                "writer": {
                    "name": "starrockswriter",
                    "parameter": {
                        "username": "xxxx",
                        "password": "xxxx",
                        "database": "xxxx",
                        "table": "xxxx",
                        "column": ["k1", "k2", "v1", "v2"],
                        "preSql": [],
                        "postSql": [],
                        "jdbcUrl": "jdbc:mysql://172.28.**.**:9030/",
                        "loadUrl": ["172.28.**.**:8030", "172.28.**.**:8030"],
                        "loadProps": {}
                    }
                }
            }
        ]
    }
}
```

相关参数描述如下表所示。

参数	描述	是否必选	默认值
username	StarRocks数据库的用户名。	是	无
password	StarRocks数据库的密码。	是	无
database	StarRocks数据库的名称。	是	无
table	StarRocks表的名称。	是	无
loadUrl	StarRocks FE的地址，用于Stream Load，可以为多个FE地址，格式为 `fe_ip:fe_http_port` 。	是	无

参数	描述	是否必选	默认值
column	目的表需要写入数据的字段，字段之间用英文逗号（,）分隔。例如，`"column": ["id","name","age"]`。 注意 该参数必须指定。如果希望导入所有字段，可以使用`["*"]`。	是	无
preSql	写入数据到目的表前，会先执行设置的标准语句。	否	无
postSql	写入数据到目的表后，会先执行设置的标准语句。	否	无
jdbcUrl	目的数据库的JDBC连接信息，用于执行preSql和postSql。	否	无
maxBatchRows	单次Stream Load导入的最大行数。	否	500000（50W）
maxBatchSize	单次Stream Load导入的最大字节数。	否	104857600 （100M）
flushInterval	上一次Stream Load结束至下一次开始的时间间隔。单位为ms。	否	300000（ms）
loadProps	Stream Load的请求参数，详情请参见Stream Load。	否	无

类型转换

默认传入的数据均会被转为字符串，并以`\t`作为列分隔符，`\n`作为行分隔符，组成CSV文件进行Stream Load导入操作。类型转换示例如下：

- 更改列分隔符，则loadProps配置如下。

```
"loadProps": {
    "column_separator": "\\x01",
    "row_delimiter": "\\x02"
}
```

- 更改导入格式为JSON，则loadProps配置如下。

```
"loadProps": {
    "format": "json",
    "strip_outer_array": true
}
```

DataWorks离线同步使用方式

1. 在DataWorks上创建工作空间，详情请参见创建工作空间。
2. 在DataWorks上创建测试表并上传数据到MaxCompute数据源，详情请参见建表并上传数据。
3. 创建StarRocks数据源。
   i. 在DataWorks的**工作空间列表**页面，单击目标工作空间操作列的**数据集成**。
   ii. 在左侧导航栏，单击**数据源**。
   iii. 单击右上角的**新增数据源**。
   iv. 在**新增数据源**对话框中，新增StarRocks类型的数据源。

4. 创建离线同步任务流程。
   i. 新建业务流程，详情请参见管理业务流程。

ii. 在目录业务流程，新建离线同步任务，详情请参见离线同步节点。

5. 在StarRocks集群中查看数据，详情请参见基础使用。

常见问题

Q：源库与目标库时区不一致，该如何处理？

A：当源库与目标库时区不同时，执行 `datax.py` 命令时需要添加如下参数。

```
"-Duser.timezone=xx时区"
```

例如，DataX导入PostgreSQL中的数据，源库是UTC时间，在dataX启动时加上参数 `"-Duser.timezone=GMT+0"` 。

5.3.3.8. Flink Connector

Flink Connector内部实现是通过缓存并批量由Stream Load导入。本文为您介绍Flink Connector的使用方式及示例。

背景信息

因为Flink官方只提供了flink-connector-jdbc，不足以满足导入性能要求，所以新增了flink-connector-starrocks，其内部实现是通过缓存并批量由Stream Load导入。

使用方式

您可以下载源码进行测试：下载flink-connector-starrocks源码。

将以下内容添加*pom.xml*中。

```
<dependency>
    <groupId>com.starrocks</groupId>
    <artifactId>flink-connector-starrocks</artifactId>
    <!-- for flink-1.11, flink-1.12 -->
    <version>x.x.x_flink-1.11</version>
    <!-- for flink-1.13 -->
    <version>x.x.x_flink-1.13</version>
</dependency>
```

> **说明** 您可以在版本信息页面，查看Latest Version信息，替换代码中的 `x.x.x` 。

代码示例如下：

- 方式一

```
// -------- sink with raw json string stream --------
fromElements(new String[]{
    "{\"score\": \"99\", \"name\": \"stephen\"}",
    "{\"score\": \"100\", \"name\": \"lebron\"}"
}).addSink(
    StarRocksSink.sink(
        // the sink options
        StarRocksSinkOptions.builder()
            .withProperty("jdbc-url", "jdbc:mysql://fe1_ip:query_port,fe2_ip:query_port,fe3_ip:query_port?xxxxx")
            .withProperty("load-url", "fe1_ip:http_port;fe2_ip:http_port;fe3_ip:http_port")
            .withProperty("username", "xxx")
            .withProperty("password", "xxx")
            .withProperty("table-name", "xxx")
            .withProperty("database-name", "xxx")
            .withProperty("sink.properties.format", "json")
            .withProperty("sink.properties.strip_outer_array", "true")
            .build()
    )
);
// -------- sink with stream transformation --------
class RowData {
    public int score;
    public String name;
    public RowData(int score, String name) {
        ......
    }
}
fromElements(
    new RowData[]{
        new RowData(99, "stephen"),
        new RowData(100, "lebron")
    }
).addSink(
    StarRocksSink.sink(
        // the table structure
        TableSchema.builder()
            .field("score", DataTypes.INT())
            .field("name", DataTypes.VARCHAR(20))
            .build(),
        // the sink options
        StarRocksSinkOptions.builder()
            .withProperty("jdbc-url", "jdbc:mysql://fe1_ip:query_port,fe2_ip:query_port,fe3_ip:query_port?xxxxx")
            .withProperty("load-url", "fe1_ip:http_port;fe2_ip:http_port;fe3_ip:http_port")
            .withProperty("username", "xxx")
            .withProperty("password", "xxx")
            .withProperty("table-name", "xxx")
            .withProperty("database-name", "xxx")
            .withProperty("sink.properties.column_separator", "\\x01")
            .withProperty("sink.properties.row_delimiter", "\\x02")
            .build(),
        // set the slots with streamRowData
        (slots, streamRowData) -> {
            slots[0] = streamRowData.score;
            slots[1] = streamRowData.name;
        }
    )
);
```

- 方式二

```
// create a table with `structure` and `properties`
// Needed: Add `com.starrocks.connector.flink.table.StarRocksDynamicTableSinkFactory` to: `src/main/resources/META-INF/services/org.apache.flink.table.factories.Factory`
tEnv.executeSql(
    "CREATE TABLE USER_RESULT(" +
        "name VARCHAR," +
        "score BIGINT" +
    ") WITH ( " +
        "'connector' = 'starrocks'," +
        "'jdbc-url'='jdbc:mysql://fe1_ip:query_port,fe2_ip:query_port,fe3_ip:query_port?xxxxx'," +
        "'load-url'='fe1_ip:http_port;fe2_ip:http_port;fe3_ip:http_port'," +
        "'database-name' = 'xxx'," +
        "'table-name' = 'xxx'," +
        "'username' = 'xxx'," +
        "'password' = 'xxx'," +
        "'sink.buffer-flush.max-rows' = '1000000'," +
        "'sink.buffer-flush.max-bytes' = '300000000'," +
        "'sink.buffer-flush.interval-ms' = '5000'," +
        "'sink.properties.column_separator' = '\\x01'," +
        "'sink.properties.row_delimiter' = '\\x02'," +
        "'sink.max-retries' = '3'" +
        "'sink.properties.*' = 'xxx'" + // stream load properties like `'sink.properties.columns' = 'k1, v1'`
    ")"
);
```

其中，Sink选项如下表所示。

参数	是否必选	默认值	类型	描述
connector	是	无	String	starrocks。
jdbc-url	是	无	String	用于在StarRocks中执行查询操作。
load-url	是	无	String	指定FE的IP和HTTP端口，格式为 `fe_ip:http_port;fe_ip:http_port` ，多个时使用半角分号（;）分隔。
database-name	是	无	String	StarRocks数据库名称。
table-name	是	无	String	StarRocks表名称。
username	是	无	String	StarRocks连接用户名。
password	是	无	String	StarRocks连接密码。
sink.semantic	否	at-least-once	String	取值为at-least-once或exactly-once。
sink.buffer-flush.max-bytes	否	94371840（90M）	String	Buffer可容纳的最大数据量，取值范围为64 MB~10 GB。
sink.buffer-flush.max-rows	否	500000	String	Buffer可容纳的最大数据行数，取值范围为64,000~5000,000。
sink.buffer-flush.interval-ms	否	300000	String	Buffer刷新时间间隔，取值范围为 1000 ms~3600000 ms。
sink.max-retries	否	1	String	最大重试次数，取值范围为0~10。
sink.connect.timeout-ms	否	1000	String	连接到load-url的超时时间，取值范围为100~60000。
sink.properties.*	否	无	String	Sink属性。

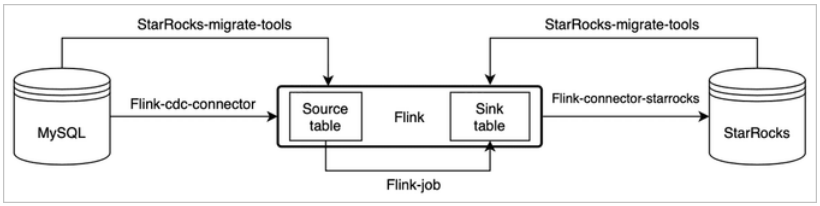
注意

- 支持Exactly-once的数据Sink保证，需要外部系统的Two-phase Commit机制。由于StarRocks无此机制，所以需要依赖Flink的checkpoint-interval，在每次Checkpoint时保存批数据以及其Label，在Checkpoint完成后的第一次invoke中阻塞flush所有缓存在state中的数据，以此达到Exactly-once。但如果StarRocks终止了，会导致用户的Flink Sink Stream算子长时间阻塞，并引起Flink的监控报警或强制退出。
- 默认使用CSV格式进行导入，您可以通过指定sink.properties.row_delimiter（该参数自StarRocks 1.15.0版本开始支持）为\\x02，sink.properties.column_separator为\\x01，来自定义行分隔符与列分隔符。
- 如果遇到导入停止的情况，请尝试增加Flink任务的内存。
- 如果代码运行正常且能接收到数据，但是写入不成功时，请确认当前机器是否能访问BE的http_port端口，即能ping通BE服务使用的IP地址。

  例如，如果一台机器有外网和内网IP，且FE或BE的http_port均可通过外网 `ip:port` 访问，集群里绑定的IP为内网IP，任务里loadurl写的FE外网 `ip:http_port` ，FE会将写入任务转发给BE内网 `ip:port` ，此时如果Client机器ping不通BE的内网IP就会写入失败。

示例：使用Flink-connector写入实现MySQL数据同步

通过Flink-cdc和StarRocks-migrate-tools（简称smt）可以实现MySQL数据的秒级同步。



smt可以根据MySQL和StarRocks的集群信息和表结构自动生成Source table和Sink table的建表语句。

1. 准备工作
   - 下载Flink。

     推荐使用1.13，最低支持1.11版本。
   - 下载Flink CDC connector。

     下载对应Flink版本的Flink-MySQL-CDC。
   - 下载Flink StarRocks connector。

     注意　1.13、1.11和1.12版本使用不同的connector。
2. 复制*flink-sql-connector-mysql-cdc-xxx.jar*，*flink-connector-starrocks-xxx.jar*到*flink-xxx/lib/*下。
3. 下载smt.tar.gz，解压缩并修改配置文件。

   相关配置项描述如下：
   - `db` ：修改为MySQL的连接信息。
   - `be_num` ：需要配置成StarRocks集群的节点数。
   - `[table-rule.1]` ：匹配规则，可以根据正则表达式匹配数据库和表名生成建表的SQL，也可以配置多个规则。
   - `flink.starrocks.*` ：StarRocks的集群配置信息。

   代码示例如下。

```
[db]
host = 192.168.**.**
port = 3306
user = root
password =
[other]
# number of backends in StarRocks
be_num = 3
# `decimal_v3` is supported since StarRocks-1.18.1
use_decimal_v3 = false
# file to save the converted DDL SQL
output_dir = ./result
[table-rule.1]
# pattern to match databases for setting properties
database = ^console_19321.*$
# pattern to match tables for setting properties
table = ^.*$
########################################
### flink sink configurations
### DO NOT set `connector`, `table-name`, `database-name`, they are auto-generated
########################################
flink.starrocks.jdbc-url=jdbc:mysql://192.168.**.**:9030
flink.starrocks.load-url= 192.168.**.**:8030
flink.starrocks.username=root
flink.starrocks.password=
flink.starrocks.sink.properties.column_separator=\x01
flink.starrocks.sink.properties.row_delimiter=\x02
flink.starrocks.sink.buffer-flush.interval-ms=15000
```

4. 执行 `starrocks-migrate-tool` 命令，将所有建表语句都生成在*result*目录下。

   您可以通过 `ls result` 命令，查看result目录。

   ```
   flink-create.1.sql    smt.tar.gz              starrocks-create.all.sql
   flink-create.all.sql  starrocks-create.1.sql
   ```

5. 执行以下命令，生成StarRocks的表结构。

   ```
   Mysql -h<FE的IP地址> -P9030 -uroot -p < starrocks-create.1.sql
   ```

6. 并开始同步，执行以下命令，生成Flink table。同步任务会持续执行。

   ```
   bin/sql-client.sh -f flink-create.1.sql
   ```

   > **注意** 如果是Flink 1.13之前的版本可能无法直接执行该脚本，需要逐行提交，并且需要打开MySQL binlog。

7. 执行以下命令，查看任务状态。

   ```
   bin/flink list
   ```

   如果有任务请查看log日志，或者调整conf中的系统配置中内存和slot。

使用本示例时，需要注意以下信息：

- 如果有多组规则，则需要给每一组规则匹配database，table和flink-connector的配置。

  代码示例如下。

```
[table-rule.1]
# pattern to match databases for setting properties
database = ^console_19321.*$
# pattern to match tables for setting properties
table = ^.*$
############################################
### flink sink configurations
### DO NOT set `connector`, `table-name`, `database-name`, they are auto-generated
############################################
flink.starrocks.jdbc-url=jdbc:mysql://192.168.**.**:9030
flink.starrocks.load-url= 192.168.**.**:8030
flink.starrocks.username=root
flink.starrocks.password=
flink.starrocks.sink.properties.column_separator=\x01
flink.starrocks.sink.properties.row_delimiter=\x02
flink.starrocks.sink.buffer-flush.interval-ms=15000
[table-rule.2]
# pattern to match databases for setting properties
database = ^database2.*$
# pattern to match tables for setting properties
table = ^.*$
############################################
### flink sink configurations
### DO NOT set `connector`, `table-name`, `database-name`, they are auto-generated
############################################
flink.starrocks.jdbc-url=jdbc:mysql://192.168.**.**:9030
flink.starrocks.load-url= 192.168.**.**:8030
flink.starrocks.username=root
flink.starrocks.password=
# 如果导入数据不方便选出合适的分隔符，则可以考虑使用JSON格式，但是会有一定的性能损失，使用方法：用以下参数替换flink.starrocks.sink.properties.column_separator和flink.starrocks.sink.properties.row_delimiter参数。
flink.starrocks.sink.properties.strip_outer_array=true
flink.starrocks.sink.properties.format=json
```

> **说明** Flink.starrocks.sink系列的参数，可以给不同的规则配置不同的导入频率等。

- 针对分库分表的大表可以单独配置一个规则。

  例如，有两个数据库edu_db_1和edu_db_2，每个数据库下面分别有course_1、course_2两张表，并且所有表的数据结构都是相同的，通过如以下配置把他们导入StarRocks的一张表中进行分析。

```
[table-rule.3]
# pattern to match databases for setting properties
database = ^edu_db_[0-9]*$
# pattern to match tables for setting properties
table = ^course_[0-9]*$
############################################
### flink sink configurations
### DO NOT set `connector`, `table-name`, `database-name`, they are auto-generated
############################################
flink.starrocks.jdbc-url=jdbc:mysql://192.168.**.**:9030
flink.starrocks.load-url= 192.168.**.**:8030
flink.starrocks.username=root
flink.starrocks.password=
flink.starrocks.sink.properties.column_separator=\x01
flink.starrocks.sink.properties.row_delimiter=\x02
flink.starrocks.sink.buffer-flush.interval-ms=5000
```

  配置后会自动生成一个多对一的导入关系，在StarRocks默认生成的表名是course__auto_shard，也可以自行在生成的配置文件中修改。

- 如果在 `sql-client` 中命令行执行建表和同步任务，需要做对反斜线（\）字符进行转义。

  代码示例如下。

```
'sink.properties.column_separator' = '\\x01'
'sink.properties.row_delimiter' = '\\x02'
```

基本原理

操作步骤

常见问题

Q：如何开启MySQL Binlog？

A：您可以通过以下步骤开启。

1. 修改*/etc/my.cnf*。

```
#开启Binlog日志。
log-bin=/var/lib/mysql/mysql-bin
#log_bin=ON
##binlog日志的基本文件名。
#log_bin_basename=/var/lib/mysql/mysql-bin
##binlog文件的索引文件，管理所有Binlog文件.
#log_bin_index=/var/lib/mysql/mysql-bin.index
#配置server-id。
server-id=1
binlog_format = row
```

2. 重启mysqld。
3. 通过 `SHOW VARIABLES LIKE 'log_bin'` 命令确认是否已经开启MySQL Binlog。

5.3.4. 性能测试

5.3.4.1. TPC-H Benchmark性能测试

TPC-H Benchmark是由国际事务处理性能委员会（Transaction Processing Performance Council）发布的数据库领域权威测试标准之一，是被工业界和学术界普遍认可的决策支持测试标准，也是数据库选型的重要参考指标之一。本文介绍如何在EMR StarRocks集群通过TPC-H Benchmark进行性能测试。

TPC-H简介

以下文字描述引用自TPC Benchmark™ H (TPC-H)：

“TPC-H是一个决策支持基准，由一套面向业务的临时查询和并发数据修改组成。选择的查询和填充数据库的数据具有广泛的行业相关性。该基准测试说明了决策支持系统可以检查大量数据，执行高度复杂的查询，并解答关键的业务问题。”

详情请参见TPCH Specification。

> **说明** 本文的TPC-H的实现基于TPC-H的基准测试，并不能与已发布的TPC-H基准测试结果相比较，本文中的测试并不符合TPC-H基准测试的所有要求。

前提条件

- 已在E-MapReduce上创建用于测试的StarRocks集群，详情请参见创建集群。
- （可选）已在OSS上创建存储空间，详情请参见创建存储空间。

> **说明** 如果您要将测试数据集生成到OSS，则需要在OSS上创建存储空间。

操作步骤

1. 获取工具包。
   i. 通过SSH方式连接StarRocks集群，请参见登录集群。
   ii. 执行以下命令，切换为emr-user用户。

   ```
   su - emr-user
   ```

   iii. 执行以下命令，给emr-user用户授权。

   ```
   sudo chown emr-user:emr-user /home/emr-user
   ```

   iv. 执行以下命令，下载*starrocks-benchmark.tgz*包。

   ```
   wget https://emr-public.oss-cn-beijing.aliyuncs.com/packages/starrocks-benchmark.tgz
   ```

   v. 执行以下命令，解压缩*starrocks-benchmark.tgz*包。

   ```
   tar xzf starrocks-benchmark.tgz
   ```

2. 配置参数。
   i. 执行以下命令，进入*starrocks-benchmark*目录。

   ```
   cd starrocks-benchmark
   ```

ii. 修改以下配置文件。

- *hosts*配置

  执行 `vim hosts` 命令，配置集群Master和Core节点的内网IP地址。

```
[master]
192.168.**.**
[core]
192.168.**.**
[master:vars]
ansible_ssh_common_args='-o StrictHostKeyChecking=no'
[core:vars]
ansible_ssh_common_args='-o StrictHostKeyChecking=no'
```

> **说明** IP地址：需要您在新版控制台节点管理页签的emr_master和emr_core机器组下查看。当Master节点有多个内网IP地址时，只需填写一个内网IP地址即可。

- *all*配置

  执行 `vim group_vars/all` 命令，配置Benchmark所需参数。

```
# mysql client config
login_host: 127.0.0.1
login_port: 9030
login_user: root
login_password: ""
# oss config
bucket: ""
endpoint: ""
access_key_id: ""
access_key_secret: ""
# benchmark config
scale_factor: 1
work_dir_root: /mnt/disk1/starrocks-benchmark/workdirs
dataset_generate_root_path: /mnt/disk1/starrocks-benchmark/datasets
```

  参数详细描述如下表所示。

参数	描述	备注
login_host	默认值为127.0.0.1。	客户端连接StarRocks服务的参数。
login_port	默认值为9030。	
login_user	为StarRocks的初始用户，默认值为root。	
login_password	默认值为空。您可以自定义，为root用户设置密码。	
bucket	OSS Bucket名称。	OSS配置。可选参数，配置后测试数据集会生成到OSS。
endpoint	OSS的访问域名。	
access_key_id	阿里云账号的AccessKey ID。	
access_key_secret	阿里云账号的AccessKey Secret。	
scale_factor	数据集的比例因子。默认值为1。	Benchmark配置。
work_dir_root	工作目录的根目录。默认值为*/mnt/disk1/starrocks-benchmark/workdirs*。	
dataset_generate_root_path	存放生成测试数据集的路径。默认值为*/mnt/disk1/starrocks-benchmark/datasets*。 如果配置了OSS，则会将对应的Bucket mount到该路径。	

3. 进行测试。

   执行以下命令，进行TPC-H测试。

```
bin/run_tpch.sh
```

   全流程自动化进行TPC-H测试，包括Datasets、SQL Queries、tables的自动生成和加载、并执行Query。

   其他操作：

   - 执行以下命令，重新加载数据集。

```
bin/run_tpch.sh reload
```

     说明 主要用于加载失败reload的场景。

   - 执行以下命令，单独执行TPC-H查询测试。

```
bin/run_tpch.sh query
```

4. 查看测试结果。

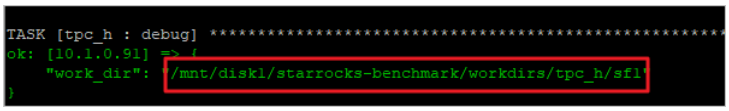
   - 测试结果概览。

     `bin/run_tpch.sh` 命令执行完后会直接输出测试结果。测试结果类似下所示。

```
TASK [tpc_h : debug] ***********************************************************************************************************
***************************************
ok: [10.1.**.**] => {
    "command_output.stdout_lines": [
        "[info] 2022-03-01 09:51:23.295 | Run sql queries started.",
        "[info] 2022-03-01 09:51:23.330 | Run q10.sql started.",
        "[info] 2022-03-01 09:51:23.913 | Run q10.sql finished. Time token: 0:00:00, .557 seconds",
        "[info] 2022-03-01 09:51:23.923 | Run q11.sql started.",
        "[info] 2022-03-01 09:51:24.026 | Run q11.sql finished. Time token: 0:00:00, .100 seconds",
        "[info] 2022-03-01 09:51:24.038 | Run q12.sql started.",
        "[info] 2022-03-01 09:51:24.192 | Run q12.sql finished. Time token: 0:00:00, .151 seconds",
        "[info] 2022-03-01 09:51:24.204 | Run q13.sql started.",
        "[info] 2022-03-01 09:51:24.553 | Run q13.sql finished. Time token: 0:00:00, .347 seconds",
        "[info] 2022-03-01 09:51:24.563 | Run q14.sql started.",
        "[info] 2022-03-01 09:51:24.665 | Run q14.sql finished. Time token: 0:00:00, .098 seconds",
        "[info] 2022-03-01 09:51:24.675 | Run q15.sql started.",
        "[info] 2022-03-01 09:51:24.852 | Run q15.sql finished. Time token: 0:00:00, .175 seconds",
        "[info] 2022-03-01 09:51:24.864 | Run q16.sql started.",
        "[info] 2022-03-01 09:51:25.008 | Run q16.sql finished. Time token: 0:00:00, .142 seconds",
        "[info] 2022-03-01 09:51:25.018 | Run q17.sql started.",
        "[info] 2022-03-01 09:51:25.269 | Run q17.sql finished. Time token: 0:00:00, .248 seconds",
        "[info] 2022-03-01 09:51:25.280 | Run q18.sql started.",
        "[info] 2022-03-01 09:51:25.800 | Run q18.sql finished. Time token: 0:00:00, .518 seconds",
        "[info] 2022-03-01 09:51:25.810 | Run q19.sql started.",
        "[info] 2022-03-01 09:51:25.943 | Run q19.sql finished. Time token: 0:00:00, .130 seconds",
        "[info] 2022-03-01 09:51:25.953 | Run q1.sql started.",
        "[info] 2022-03-01 09:51:26.295 | Run q1.sql finished. Time token: 0:00:00, .339 seconds",
        "[info] 2022-03-01 09:51:26.305 | Run q20.sql started.",
        "[info] 2022-03-01 09:51:26.708 | Run q20.sql finished. Time token: 0:00:00, .400 seconds",
        "[info] 2022-03-01 09:51:26.720 | Run q21.sql started.",
        "[info] 2022-03-01 09:51:27.323 | Run q21.sql finished. Time token: 0:00:00, .600 seconds",
        "[info] 2022-03-01 09:51:27.334 | Run q22.sql started.",
        "[info] 2022-03-01 09:51:27.403 | Run q22.sql finished. Time token: 0:00:00, .065 seconds",
        "[info] 2022-03-01 09:51:27.415 | Run q2.sql started.",
        "[info] 2022-03-01 09:51:27.632 | Run q2.sql finished. Time token: 0:00:00, .213 seconds",
        "[info] 2022-03-01 09:51:27.648 | Run q3.sql started.",
        "[info] 2022-03-01 09:51:27.917 | Run q3.sql finished. Time token: 0:00:00, .262 seconds",
        "[info] 2022-03-01 09:51:27.936 | Run q4.sql started.",
        "[info] 2022-03-01 09:51:28.150 | Run q4.sql finished. Time token: 0:00:00, .210 seconds",
        "[info] 2022-03-01 09:51:28.172 | Run q5.sql started.",
        "[info] 2022-03-01 09:51:28.954 | Run q5.sql finished. Time token: 0:00:00, .778 seconds",
        "[info] 2022-03-01 09:51:28.976 | Run q6.sql started.",
        "[info] 2022-03-01 09:51:29.080 | Run q6.sql finished. Time token: 0:00:00, .103 seconds",
        "[info] 2022-03-01 09:51:29.096 | Run q7.sql started.",
        "[info] 2022-03-01 09:51:29.445 | Run q7.sql finished. Time token: 0:00:00, .346 seconds",
        "[info] 2022-03-01 09:51:29.460 | Run q8.sql started.",
        "[info] 2022-03-01 09:51:32.692 | Run q8.sql finished. Time token: 0:00:03, 3.229 seconds",
        "[info] 2022-03-01 09:51:32.703 | Run q9.sql started.",
        "[info] 2022-03-01 09:51:33.318 | Run q9.sql finished. Time token: 0:00:00, .611 seconds",
        "[info] 2022-03-01 09:51:33.324 | Run sql queries finished. Time token: 0:00:10, 10.026 seconds"
    ]
}
TASK [tpc_h : debug] ***********************************************************************************************************
*********************************************************************************
ok: [10.1.0.91] => {
    "work_dir": "/mnt/disk1/starrocks-benchmark/workdirs/tpc_h/sf1"
}
```

- 测试结果详情。

  成功执行 `bin/run_tpch.sh` 命令后，系统会构建好整个TPC-H测试的工作目录并输出*<work_dir>*目录的路径信息，您可以切换到该路径下查看相关的Query语句、建表语句和执行日志等信息。



> **说明** 本示例的*<work_dir>*为*/mnt/disk1/starrocks-benchmark/workdirs/tpc_h/sf1*。

  目录结果如下所示。

```
<work_dir>/
├── config # run.sh run_mysql.sh脚本的配置。
├── logs # 最近一次TPC-H执行日志。
│   ├── *.sql.err
│   ├── *.sql.out
│   └── run.log
├── queries # TPC-H涉及的22条SQL Query。
│   ├── ddl
│   │   └── create_tables.sql # TPC-H建表语句。
│   └── *.sql
├── run_mysql.sh
├── run.sh # 全量执行TPC-H的Query。
└── tpch_tools # dbgen工具包。
```

您也可以通过命令 `cd <work_dir>/logs` 进入工作目录的Logs下，然后执行以下命令查看测试结果以及执行SQL的详细结果。

```
cat run.log
```

5.3.5. StarRocks运维

5.3.5.1. 参数配置

本文为您介绍FE、BE、Broker以及系统参数的部分配置项以及修改方式。

背景信息

配置项分为动态参数和静态参数。动态参数主要是FE有部分动态配置，支持在线修改。静态参数是需要重启服务生效的配置项。

本文为您介绍以下内容：

- FE配置项
  - FE动态参数
  - FE静态参数
- BE配置项

FE动态参数

动态参数可以通过如下命令修改。

```
ADMIN SET FRONTEND CONFIG ("key" = "value");
```

LOG相关配置

配置项	默认值	描述
qe_slow_log_ms	5000	Slow query的认定时长，单位为ms。

元数据与集群管理相关配置

配置项	默认值	描述
catalog_try_lock_timeout_ms	5000	Catalog Lock获取的超时时长，单位为ms。
edit_log_roll_num	50000	Image日志拆分大小。
ignore_unknown_log_id	FALSE	是否忽略未知的logID： • TRUE：FE会忽略这些未知的logID。 • FALSE：针对未知的logID，FE会退出进程。 当FE回滚到低版本时，可能存在低版本BE无法识别的logID。
ignore_meta_check	FALSE	是否忽略元数据落后的情形： • FALSE：忽略。 • TRUE：不忽略。
max_backend_down_time_second	3600	BE和FE失联之后，FE能够容忍BE重新加回来的最长时间，单位为s。

配置项	默认值	描述
drop_backend_after_decommission	TRUE	BE被下线后，是否删除该BE： • TRUE：删除该BE。 • FALSE：不删除该BE。
catalog_try_lock_timeout_ms	5000	Catalog Lock获取的超时时长，单位为ms。

查询引擎相关配置

配置项	默认值	描述
expr_children_limit	10000	查询中IN谓词中可以涉及的数目。
expr_depth_limit	3000	查询嵌套的层次。
max_allowed_in_element_num_of_delete	10000	DELETE语句中IN谓词最多允许的元素数量。
max_layout_length_per_row	2147483647	单行最大的长度。
disable_cluster_feature	TRUE	是否禁用逻辑集群功能： • TRUE：禁用。 • FALSE：不禁用。
enable_materialized_view	TRUE	是否允许创建物化视图： • TRUE：允许。 • FALSE：不允许。
enable_decimal_v3	TRUE	是否开启Decimal V3： • TRUE：开启。 • FALSE：不开启。
enable_sql_blacklist	FALSE	是否开启SQL Query黑名单校验： • TRUE：开启。 • FALSE：不开启。 说明 如果开启该功能，则在黑名单中的Query不能被执行。
dynamic_partition_check_interval_seconds	600	动态分区检查的时间周期，单位为s。
dynamic_partition_enable	TRUE	是否开启动态分区功能： • TRUE：开启动态分区功能。 • FALSE：不开启动态分区功能。
max_partitions_in_one_batch	4096	批量创建分区时，分区数目的最大值。
max_query_retry_time	2	FE上查询重试的次数。
max_create_table_timeout_second	60	建表最大超时时间，单位为s。
max_running_rollup_job_num_per_table	1	每个Table执行Rollup任务的最大并发度。
max_planner_scalar_rewrite_num	10_0000	优化器重写ScalarOperator允许的最大次数。
statistics_manager_sleep_time_sec	60*10	自动创建统计信息表的周期，单位为s。
statistic_collect_interval_sec	120*60	统计信息功能执行周期，单位为s。
statistic_update_interval_sec	24 *60* 60	统计信息Job的默认收集间隔时间，单位为s。
statistic_sample_collect_rows	200000	采样统计信息Job的默认采样行数，默认为200000行。

配置项	默认值	描述
enable_statistic_collect	TRUE	统计信息收集功能开关： • TRUE：开启统计信息收集功能。 • FALSE：不开启统计信息收集功能。
enable_local_replica_selection	FALSE	优化器是否优先选择与该FE相同IP的BE节点上的tablet： • TRUE：是。 • FALSE：否。
max_distribution_pruner_recursion_depth	100	分区裁剪允许的最大递归深度。

导入和导出相关配置

配置项	默认值	描述
load_straggler_wait_second	300	控制BE副本最大容忍的导入落后时长，单位为s。 如果超过该时长，则进行克隆。
desired_max_waiting_jobs	100	最多等待的任务。 适用于所有的任务，例如建表、导入和是Schema Change。
max_running_txn_num_per_db	100	并发导入的任务数。
max_load_timeout_second	259200	适用于所有导入，单位为s。
min_load_timeout_second	1	适用于所有导入，单位为s。
load_parallel_instance_num	1	单个BE上并发实例数，默认1个。
disable_hadoop_load	FALSE	是否禁用从Hadoop导入： • TRUE：禁用从Hadoop导入。 • FALSE：不禁用从Hadoop导入。
disable_load_job	FALSE	如果集群异常时，是否接受导入任务： • TRUE：接受导入任务。 • FALSE：不接受导入任务。
db_used_data_quota_update_interval_secs	300	更新数据库使用配额的时间周期，单位为s。
history_job_keep_max_second	604800	历史任务最大的保留时长，单位为s。
label_keep_max_num	1000	一定时间内所保留导入任务的最大数量。 保留时间在label_keep_max_second中设置。
label_keep_max_second	259200	label保留时长，单位为s。
max_routine_load_job_num	100	最大的Routine Load作业数。
max_routine_load_task_concurrent_num	5	每个Routine Load作业最大并发执行的task数。
max_routine_load_task_num_per_be	5	每个BE最大并发执行的Routine Load task数，需要小于等于BE的routine_load_thread_pool_size配置。
max_routine_load_batch_size	524288000	每个Routine Load task导入的最大数据量。
routine_load_task_consume_second	3	每个Routine Load task消费数据的最大时间，单位为s。
routine_load_task_timeout_second	15	每个Routine Load task超时时间，单位为s。

配置项	默认值	描述
max_tolerable_backend_down_num	0	如果故障的BE节点数超过该阈值，则不能自动恢复Routine Load作业。
period_of_auto_resume_min	5	自动恢复Routine Load的时间间隔。
spark_load_default_timeout_second	86400	Spark导入的超时时间，单位为s。
spark_home_default_dir	STARROCKS_HOME_DIR/lib/spark2x	Spark客户端根目录。
stream_load_default_timeout_second	600	StreamLoad超时时间，单位为s。
max_stream_load_timeout_second	259200	Stream导入的超时时间允许设置的最大值，单位为s。
insert_load_default_timeout_second	3600	Insert Into语句的超时时间，单位为s。
broker_load_default_timeout_second	14400	Broker Load的超时时间，单位为s。
min_bytes_per_broker_scanner	67108864	单个实例处理的最小数据量，默认64 MB。
max_broker_concurrency	100	单个任务最大并发实例数，默认100个。
export_max_bytes_per_be_per_task	268435456	单个导出任务在单个BE上导出的最大数据量，默认256 MB。
export_running_job_num_limit	5	导出作业最大的运行数目。
export_task_default_timeout_second	7200	导出作业超时时长，单位为s，默认2小时。

存储相关配置

配置项	默认值	描述
enable_strict_storage_medium_check	FALSE	在创建表时，FE是否检查BE的可用的存储介质空间： • TRUE：检查。 • FALSE：不检查。
capacity_used_percent_high_water	0.75	Backend上磁盘使用容量的度量值。 超过0.75之后，尽量不再往该tablet上发送建表和克隆的任务，直至恢复正常。
storage_high_watermark_usage_percent	85	BE存储目录下空间使用率的最大值。
storage_min_left_capacity_bytes	2 *1024* 1024*1024	BE存储目录下剩余空间的最小值，默认2 GB。
storage_flood_stage_left_capacity_bytes	1 *1024* 1024*1024	BE存储目录的剩余空间。 如果剩余空间小于该值，则会拒绝Load Restore作业。默认1 GB。
storage_flood_stage_usage_percent	95	BE存储目录下空间使用率。 如果空间使用率超过该值，则会拒绝Load和Restore作业。
catalog_trash_expire_second	86400	删表或数据库之后，元数据在回收站中保留的时长，单位为s，默认1天。 如果超过该时长，则数据无法恢复。
alter_table_timeout_second	86400	Schema change超时时间，单位为s，默认1天。

配置项	默认值	描述
balance_load_disk_safe_threshold	0.5	仅对disk_and_tablet策略有效。 如果所有BE的磁盘使用率低于50%，则认为磁盘使用均衡。
balance_load_score_threshold	0.1	• 针对be_load_score策略，负载比平均负载低10%的BE处于低负载状态，比平均负载高10%的BE处于高负载状态。 • 针对disk_and_tablet策略，如果最大和最小BE磁盘使用率之差高于10%，则认为磁盘使用不均衡，会触发tablet重新均衡。
disable_balance	FALSE	是否禁用Tablet调度： • FALSE：禁用。 • TRUE：不禁用。
max_scheduling_tablets	2000	正在调度的tablet数量。 如果正在调度的tablet数量超过该值，则跳过tablet均衡检查。
max_balancing_tablets	100	正在均衡的tablet数量。 如果正在均衡的tablet数量超过该值，则跳过tablet重新均衡。
disable_colocate_balance	FALSE	是否禁用Colocate Table的副本均衡： • TRUE：禁用。 • FALSE：不禁用。
recover_with_empty_tablet	FALSE	在tablet副本丢失或损坏时，是否使用空的tablet代替： • TRUE：使用空的tablet代替。 • FALSE：不使用空的tablet代替。 使用空的tablet代替可以保证在有tablet副本丢失或损坏时，query依然能被执行（但是由于缺失了数据，结果可能是错误的）。
min_clone_task_timeout_sec	3*60	克隆Tablet的最小超时时间，单位为s，默认3min。
max_clone_task_timeout_sec	2 *60* 60	克隆 Tablet 的最大超时时间，单位为s，默认2h。
tablet_create_timeout_second	1	建表超时时长，单位为s。
tablet_delete_timeout_second	2	删除表的超时时间，单位为s。
tablet_repair_delay_factor_second	60	FE 控制进行副本修复的间隔，单位为s。
consistency_check_start_time	23	FE 发起副本一致性检测的起始时间，默认是23:00。
consistency_check_end_time	4	FE 发起副本一致性检测的终止时间，默认是4:00。
check_consistency_default_timeout_second	600	副本一致性检测的超时时间，单位为s。

其他配置

配置项	默认值	描述
plugin_enable	TRUE	是否开启了插件功能： • TRUE：开启了插件功能。 • FALSE：没有开启插件功能。 **说明** 只能在Master节点安装或卸载插件。
max_small_file_number	100	允许存储小文件数目的最大值。
max_small_file_size_bytes	1024*1024	存储文件的大小上限，默认1 MB。
backup_job_default_timeout_ms	86400*1000	Backup作业的超时时间，单位为s，默认1天。

配置项	默认值	描述
report_queue_size	100	Disk、Task或Tablet的Report的等待队列长度。

FE静态参数

LOG相关配置

配置项	默认值	描述
log_roll_size_mb	1024	日志拆分的大小，每1 GB拆分一个日志。
sys_log_dir	StarRocksFe.STARROCKS_HOME_DIR/log	日志保留的目录。
sys_log_level	INFO	日志的级别。 可以配置的等级从宽松到严格依次为INFO、WARNING、ERROR和FATAL。
sys_log_verbose_modules	空字符串	日志打印的模块。 例如，如果填写为org.apache.starrocks.catalog，则只打印catalog模块下的日志。
sys_log_roll_interval	DAY	日志拆分的时间间隔。
sys_log_delete_age	7d	日志删除的间隔。
sys_log_roll_num	10	每个sys_log_roll_interval时间内，保留的日志文件数目。
audit_log_dir	starrocksFe.STARROCKS_HOME_DIR/log	审计日志保留的目录。
audit_log_roll_num	90	审计日志保留的数目。
audit_log_modules	"slow_query", "query"	审计日志打印的模块，默认保留slow_query和query。
audit_log_roll_interval	DAY	审计日志拆分的时间间隔，取值为DAY或HOUR。
audit_log_delete_age	30d	审计日志删除的间隔。
dump_log_dir	STARROCKS_HOME_DIR/log	Dump日志的目录。
dump_log_modules	"query"	Dump日志打印的模块，默认保留query。
dump_log_roll_interval	DAY	Dump日志拆分的时间间隔。 日志文件的后缀为yyyyMMdd（DAY）或yyyyMMddHH（HOUR）。
dump_log_roll_num	90	每个dump_log_roll_interval时间内，保留的Dump日志文件数目。
dump_log_delete_age	30d	Dump日志保留的时间长度。

Server进程相关配置

配置项	默认值	描述
frontend_address	0.0.0.0	FE IP地址。
priority_networks	空字符串	以CIDR形式10.10.**.**/24指定BE的IP地址，适用于机器有多个IP，需要指定优先使用的网络。
http_port	8030	Http Server的端口。
http_backlog_num	1024	HTTP Server的backlog队列长度。
cluster_name	StarRocks Cluster	Web页面中Title显示的集群名称。
rpc_port	9020	FE上的Thrift Server端口。

配置项	默认值	描述
thrift_backlog_num	1024	Thrift Server的backlog队列长度。
thrift_server_type	THREAD_POOL	FE的Thrift服务使用的服务模型。例如，SIMPLE、THREADED和THREAD_POOL。
thrift_server_max_worker_threads	4096	Thrift Server最大工作线程数。
thrift_client_timeout_ms	0	Client超时时间。 默认值为0，表示永远不会超时。
brpc_idle_wait_max_time	10000	BRPC的空闲等待时间，单位为ms，默认为10s。
query_port	9030	FE上的MySQL Server端口。
mysql_service_nio_enabled	FALSE	是否开启FE连接服务的nio： • TRUE：开启。 • FALSE：不开启。
mysql_service_io_threads_num	4	FE连接服务线程数。
mysql_nio_backlog_num	1024	MySQL Server的backlog队列长度。
max_mysql_service_task_threads_num	4096	MySQL Server处理任务的最大线程数。
max_connection_scheduler_threads_num	4096	连接定时器的线程池的最大线程数。
qe_max_connection	1024	FE上最多接收的连接数，适用于所有用户。
check_java_version	TRUE	是否检查执行时的版本与编译的Java版本的兼容性： • TRUE：检查。 • FALSE：不检查。

元数据与集群管理相关配置

配置项	默认值	描述
meta_dir	StarRocksFe.STARROCKS_HOME_DIR/meta	元数据保留目录。
heartbeat_mgr_threads_num	8	HeartbeatMgr中发送心跳任务的线程数。
heartbeat_mgr_blocking_queue_size	1024	HeartbeatMgr中发送心跳任务的线程池的队列长度。
metadata_failure_recovery	FALSE	强制重置FE的元数据。 **说明** 请谨慎使用该参数。
edit_log_port	9010	FE Group（Master、Follower、Observer）之间通信用的端口。
edit_log_type	BDB	Edit log的类型，只能为BDB。
bdbje_heartbeat_timeout_second	30	BDBJE心跳超时的间隔，单位为s。
bdbje_lock_timeout_second	1	BDBJE锁超时的间隔，单位为s。
max_bdbje_clock_delta_ms	5000	Master与Non-master最大容忍的时钟偏移，单位为ms。
txn_rollback_limit	100	事务回滚的上限。
bdbje_replica_ack_timeout_second	10	BDBJE Master等待足够多的FOLLOWER ACK的最长时间c
master_sync_policy	SYNC	Master日志刷盘的方式，默认是SYNC。

配置项	默认值	描述
replica_sync_policy	SYNC	Follower日志刷盘的方式，默认是SYNC。
meta_delay_toleration_second	300	非Master节点能够容忍的最大元数据落后的时间，单位为s。
cluster_id	-1	相同cluster_id的FE或BE节点属于haode同一个集群。 默认值-1，表示在Master FE第一次启动时随机生成一个cluster_id。

查询引擎相关配置

配置项	默认值	描述
disable_colocate_join	FALSE	是否开启Colocate Join： • FALSE：不开启Colocate Join。 • TRUE：开启Colocate Join。
enable_udf	FALSE	是否开启UDF： • FALSE：不开启UDF。 • TRUE：开启UDF。
publish_version_interval_ms	10	发送版本生效任务的时间间隔。
statistic_cache_columns	10_0000	缓存统计信息表的行数。

导入和导出相关配置

配置项	默认值	描述
async_load_task_pool_size	10	导入任务执行的线程池大小。
load_checker_interval_second	5	导入轮询的间隔，单位为s。
transaction_clean_interval_second	30	清理已结束事务的周期，单位为s。
label_clean_interval_second	14400	label清理的间隔，单位为s。
spark_dpp_version	1.0.0	Spark dpp版本。
spark_resource_path	空字符串	Spark依赖包的根目录。
spark_launcher_log_dir	sys_log_dir/spark_launcher_log	Spark日志目录。
yarn_client_path	STARROCKS_HOME_DIR/lib/yarn-client/hadoop/bin/yarn	YARN客户端根目录。
yarn_config_dir	STARROCKS_HOME_DIR/lib/yarn-config	YARN配置文件目录。
export_checker_interval_second	5	导出线程轮询间隔，单位为s。
export_task_pool_size	5	导出任务线程池大小。
export_checker_interval_second	5	导出作业调度器的调度周期，单位为s。

存储相关配置

配置项	默认值	描述
storage_cooldown_second	2592000	介质迁移的时间，单位为s，默认30天。

配置项	默认值	描述
default_storage_medium	HDD	默认的存储介质，取值为HDD和SSD。 在创建表或分区时，如果没有指定存储介质，则会使用该值。
schedule_slot_num_per_path	2	一个BE存储目录能够同时执行tablet相关任务的数目。
tablet_balancer_strategy	disk_and_tablet	Tablet均衡策略，取值为disk_and_tablet或be_load_score。
tablet_stat_update_interval_second	300	FE向每个BE请求收集tablet信息的时间间隔，单位为s，默认5min。

其他配置

配置项	默认值	描述
plugin_dir	STARROCKS_HOME_DIR/plugins	插件安装的目录。
small_file_dir	STARROCKS_HOME_DIR/small_files	小文件的根目录。
max_agent_task_threads_num	4096	代理任务的线程池的最大线程数。
authentication_ldap_simple_bind_base_dn	""	用户的base DN，指定用户的检索范围。
authentication_ldap_simple_bind_root_dn	""	检索用户时，使用的管理员账号DN。
authentication_ldap_simple_bind_root_pwd	""	检索用户时，使用的管理员账号密码。
authentication_ldap_simple_server_host	""	LDAP服务的host地址。
authentication_ldap_simple_server_port	389	LDAP服务的端口。
authentication_ldap_simple_user_search_attr	uid	LDAP对象中标识用户的属性名称。
tmp_dir	starrocksFe.STARROCKS_HOME_DIR/temp_ddir	临时文件保存目录，例如Backup和Restore等进程保留的目录。
locale	zh_CN.UTF-8	字符集。
hive_meta_load_concurrency	4	Hive元数据并发线程数。
hive_meta_cache_refresh_interval_s	4096	定时刷新Hive外表元数据缓存的周期，单位为s。
hive_meta_cache_ttl_s	3600 *2	Hive外表元数据缓存失效时间，单位为s，默认2h。
hive_meta_store_timeout_s	3600 *24	连接Hive MetaStore的超时时间，单位为s，默认24h。
es_state_sync_interval_second	10	FE获取ElasticSearch Index的时间，单位为s。
enable_auth_check	TRUE	是否开启鉴权： • TRUE：开启鉴权。 • FALSE：不开启鉴权。
enable_metric_calculator	TRUE	是否开启定期收集Metrics： • TRUE：开启。 • FALSE：不开启。

BE配置项

BE配置项全部是静态参数。

配置项	默认值	描述
be_port	9060	BE上Thrift Server的端口，用于接收来自FE的请求。
brpc_port	8060	BRPC的端口，可以查看BRPC的一些网络统计信息。
brpc_num_threads	-1	BRPC的bthreads线程数量。默认值-1表示和CPU核数一样。
priority_networks	空字符串	以CIDR形式10.10.**.**/24指定BE的IP地址，适用于机器有多个IP，需要指定优先使用的网络。
heartbeat_service_port	9050	心跳服务端口（Thrift），接收来自FE的心跳。
heartbeat_service_thread_count	1	心跳线程数。
create_tablet_worker_count	3	创建tablet的线程数。
drop_tablet_worker_count	3	删除tablet的线程数。
push_worker_count_normal_priority	3	导入线程数，处理NORMAL优先级任务。
push_worker_count_high_priority	3	导入线程数，处理HIGH优先级任务。
publish_version_worker_count	2	生效版本的线程数。
clear_transaction_task_worker_count	1	清理事务的线程数。
alter_tablet_worker_count	3	进行Schema Change的线程数。
clone_worker_count	3	克隆的线程数。
storage_medium_migrate_count	1	介质迁移的线程数。例如，热数据从SSD迁移到SATA盘的线程数。
check_consistency_worker_count	1	计算tablet的校验和checksum。
report_task_interval_seconds	10	汇报单个任务的间隔，单位为s。 建表、删除表、导入和Schema Change都可以被认定是任务。
report_disk_state_interval_seconds	60	汇报磁盘状态的间隔，单位为s。 汇报各个磁盘的状态及其数据量等。
report_tablet_interval_seconds	60	汇报tablet的间隔，单位为s。 汇报所有的tablet的最新版本。
alter_tablet_timeout_seconds	86400	Schema Change超时时间，单位为s。
sys_log_dir	${DORIS_HOME}/log	存放日志的目录。 日志级别包括INFO、WARNING、ERROR和FATAL。
user_function_dir	${DORIS_HOME}/lib/udf	存放UDF程序的目录。
sys_log_level	INFO	日志的等级。 可以配置的等级从宽松到严格依次为INFO、WARNING、ERROR和FATAL。
sys_log_roll_mode	SIZE-MB-1024	日志拆分的大小，每1 GB拆分一个日志。
sys_log_roll_num	10	日志保留的数目。
sys_log_verbose_modules	空字符串	日志打印的模块。如果写olap，则只打印olap模块下的日志。
sys_log_verbose_level	10	日志显示的级别，用于控制代码中VLOG开头的日志输出。
log_buffer_level	空字符串	日志刷盘的策略，默认保持在内存中。

配置项	默认值	描述
num_threads_per_core	3	每个CPU core启动的线程数。
compress_rowbatches	TRUE	BE之间RPC通信是否压缩RowBatch，用于查询层之间的数据传输。
serialize_batch	FALSE	BE之间RPC通信是否序列化RowBatch，用于查询层之间的数据传输。
status_report_interval	5	查询汇报profile的间隔，单位为s，用于FE收集查询统计信息。
doris_scanner_thread_pool_thread_num	48	存储引擎并发扫描磁盘的线程数，统一管理在线程池中。
doris_scanner_thread_pool_queue_size	102400	存储引擎最多接收的任务数。
doris_scan_range_row_count	524288	存储引擎拆分查询任务的粒度。
doris_scanner_queue_size	1024	存储引擎支持的扫描任务数。
doris_scanner_row_num	16384	每个扫描线程单次执行最多返回的数据行数。
doris_max_scan_key_num	1024	查询最多拆分的scan key数目。
column_dictionary_key_ratio_threshold	0	字符串类型的取值比例，小于这个比例采用字典压缩算法。
column_dictionary_key_size_threshold	0	字典压缩列大小，小于这个值采用字典压缩算法。
memory_limitation_per_thread_for_schema_change	2	单个Schema Change任务允许占用的最大内存。
file_descriptor_cache_clean_interval	3600	文件句柄缓存清理的间隔，单位为s，用于清理长期不用的文件句柄。
disk_stat_monitor_interval	5	磁盘状态检测的间隔，单位为s。
unused_rowset_monitor_interval	30	清理过期Rowset的时间间隔，单位为s。
storage_root_path	空字符串	存储数据的目录。
max_tablet_num_per_shard	1024	每个shard的tablet数目，用于划分tablet，防止单个目录下tablet子目录过多。
pending_data_expire_time_sec	1800	存储引擎保留的未生效数据的最大时长，单位为s。
inc_rowset_expired_sec	1800	在增量克隆场景下，已导入的数据，在存储引擎中保留的时间，单位为s。
max_garbage_sweep_interval	3600	磁盘进行垃圾清理的最大间隔，单位为s。
min_garbage_sweep_interval	180	磁盘进行垃圾清理的最小间隔，单位为s。
snapshot_expire_time_sec	172800	快照文件清理的间隔，单位为s，默认为48小时。
trash_file_expire_time_sec	259200	回收站清理的间隔，单位为s，默认为72小时。
file_descriptor_cache_capacity	16384	文件句柄缓存的容量。
min_file_descriptor_number	60000	BE进程的文件句柄limit要求的下线。
index_stream_cache_capacity	10737418240	BloomFilter、Min或Max等统计信息缓存的容量。
storage_page_cache_limit	0	PageCache的容量。
disable_storage_page_cache	TRUE	是否禁用Page Cache。 • TRUE：禁用Page Cache。 • FALSE：不禁用Page Cache。
base_compaction_check_interval_seconds	60	BaseCompaction线程轮询的间隔，单位为s。

配置项	默认值	描述
base_compaction_num_threads_per_disk	1	每个磁盘BaseCompaction线程的数目。
base_cumulative_delta_ratio	0.3	BaseCompaction触发条件之一：Cumulative文件大小达到Base文件的比例。
base_compaction_interval_seconds_since_last_operation	86400	BaseCompaction触发条件之一：上一轮BaseCompaction距今的间隔。
cumulative_compaction_check_interval_seconds	1	CumulativeCompaction线程轮询的间隔，单位为s。
min_cumulative_compaction_num_singleton_deltas	5	CumulativeCompaction触发条件之一：Singleton文件数目要达到的下限。
max_cumulative_compaction_num_singleton_deltas	1000	CumulativeCompaction 触发条件之一：Singleton文件数目要达到的上限。
cumulative_compaction_num_threads_per_disk	1	每个磁盘CumulativeCompaction线程的数目。
min_compaction_failure_interval_sec	120	Tablet Compaction失败之后，再次被调度的间隔，单位为s。
max_compaction_concurrency	-1	BaseCompaction + CumulativeCompaction的最大并发。 默认值-1表示没有限制。
webserver_port	8040	Http Server端口。
webserver_num_workers	48	Http Server线程数。
periodic_counter_update_period_ms	500	Counter统计信息的间隔，单位为ms。
load_data_reserve_hours	4	小批量导入生成的文件保留的时间，单位为h。
load_error_log_reserve_hours	48	导入数据信息保留的时长，单位为h。
number_tablet_writer_threads	16	流式导入的线程数。
streaming_load_max_mb	10240	流式导入单个文件大小的上限。
streaming_load_rpc_max_alive_time_sec	1200	流式导入RPC的超时时间。
fragment_pool_thread_num	64	查询线程数，默认启动64个线程，后续查询请求动态创建线程。
fragment_pool_queue_size	2048	单节点上能够处理的查询请求上限。
enable_partitioned_aggregation	TRUE	是否使用PartitionAggregation： • TRUE：使用PartitionAggregation。 • FALSE：不使用PartitionAggregation。
enable_token_check	TRUE	是否开启Token检验： • TRUE：开启Token检验。 • FALSE：不开启Token检验。
load_process_max_memory_limit_bytes	107374182400	单节点上所有的导入线程占据的内存上限，默认为100 GB。
load_process_max_memory_limit_percent	30	单节点上所有的导入线程占据的内存上限比例。
sync_tablet_meta	FALSE	存储引擎是否开sync保留到磁盘上。
thrift_rpc_timeout_ms	5000	Thrift超时的时长，单位为ms。
txn_commit_rpc_timeout_ms	10000	Txn超时的时长，单位为ms。

配置项	默认值	描述
routine_load_thread_pool_size	10	例行导入的线程池数目。
tablet_meta_checkpoint_min_new_rowsets_num	10	TabletMeta Checkpoint的最小Rowset数目。
tablet_meta_checkpoint_min_interval_secs	600	TabletMeta Checkpoint线程轮询的时间间隔，单位为s。
brpc_max_body_size	209715200	BRPC最大的包容量，默认为200 MB。
max_runnings_transactions	2000	存储引擎支持的最大事务数。
tablet_map_shard_size	32	Tablet分组数。
enable_bitmap_union_disk_format_with_set	FALSE	Bitmap新存储格式，可以优化bitmap_union性能。

5.3.5.2. 内存管理

本文为您介绍StarRocks BE（Backend）中内存使用分类、内存配置以及如何查看内存使用。

内存分类

您可以在E-MapReduce控制台目标集群的**集群服务**页面，**STARROCKS**服务的**配置**页面的**be.conf**页签，新增或修改BE相关配置，具体操作请参见管理配置项。

标识	Metric名称	描述	BE相关配置
process	starrocks_be_process_mem_bytes	BE进程实际使用的内存（不包含预留的空闲内存）。	mem_limit
query_pool	starrocks_be_column_pool_mem_bytes	BE查询层使用总内存。	无
load	starrocks_be_load_mem_bytes	导入使用的总内存。	load_process_max_memory_limit_bytes，load_process_max_memory_limit_percent
table_meta	starrocks_be_tablet_meta_mem_bytes	元数据总内存。	无
compaction	starrocks_be_compaction_mem_bytes	版本合并总内存。	compaction_max_memory_limit，compaction_max_memory_limit_percent
column_pool	starrocks_be_column_pool_mem_bytes	Column Pool内存池，用于加速存储层数据读取的Column Cache。	无
page_cache	starrocks_be_storage_page_cache_mem_bytes	BE存储层page缓存。	disable_storage_page_cache，storage_page_cache_limit
chunk_allocator	starrocks_be_chunk_allocator_mem_bytes	CPU per core缓存，用于加速小块内存申请的Cache。	chunk_reserved_bytes_limit
consistency	starrocks_be_consistency_mem_bytes	定期一致性校验使用的内存。	consistency_max_memory_limit_percent，consistency_max_memory_limit
schema_change	starrocks_be_schema_change_mem_bytes	Schema Change任务使用的总内存。	memory_limitation_per_thread_for_schema_change
clone	starrocks_be_clone_mem_bytes	Tablet Clone任务使用的总内存。	无
update	starrocks_be_update_mem_bytes	主键模型使用的总内存。	无

内存相关配置

BE配置

名称	默认值	描述
mem_limit	90%	BE进程内存上限，默认硬限为BE所在机器内存的90%，软限为BE所在机器内存的80%。 如果是BE独立部署的话，不需要配置，如果是和其它占用内存比较多的服务混合部署的话，要合理配置。
load_process_max_memory_limit_bytes	107374182400	分别表示导入最大可用内存上限和最大可用内存百分比上限。 系统会在 `mem_limit * load_process_max_memory_limit_percent / 100` 和load_process_max_memory_limit_bytes中取较小者作为最终的可用内存上限。 如果导入内存到达限制，则会触发刷盘和反压逻辑。
load_process_max_memory_limit_percent	30	
compaction_max_memory_limit	-1	分别表示Compaction内存上限和内存百分比上限。 系统会在 `mem_limit * compaction_max_memory_limit_percent / 100` 和compaction_max_memory_limit中取较小者作为最终的Compaction内存上限。默认值-1表示没有限制，不建议修改默认配置。 如果Compaction内存到达限制，则会导致Compaction任务失败。
compaction_max_memory_limit_percent	100	
disable_storage_page_cache	true	是否禁用BE存储层page缓存，和storage_page_cache_limit配合使用，在内存资源充足和有大数据量查询的场景中可以打开，能够加速查询性能。
storage_page_cache_limit	0	BE存储层page缓存可以使用的内存上限。
chunk_reserved_bytes_limit	2147483648	用于加速小块内存分配的Cache，默认上限为2147483648（2 GB），在内存资源充足的情况下可以考虑打开。
consistency_max_memory_limit	10G	一致性校验任务使用的内存上限和内存上限百分比。 系统会在 `mem_limit * consistency_max_memory_limit_percent / 100` 和consistency_max_memory_limit中取较小者作为最终的内存上限。 如果内存使用超限，则会导致任务失败。
consistency_max_memory_limit_percent	20	
memory_limitation_per_thread_for_schema_change	2	单个Schema Change任务的内存使用上限。 如果内存使用超限，则会导致Schema Change任务失败。
tc_use_memory_min	10737418240	TCmalloc最小预留内存。 如果小于该值，则StarRocks不会将空闲内存返还给操作系统。

Session变量

名称	默认值	描述
exec_mem_limit	2147483648	单个instance内存限制。
load_mem_limit	0	单个导入任务的内存限制。 默认为0，即表示不使用该变量，而使用exec_mem_limit作为内存限制。

查看内存使用

您可以通过以下两种方法分析BE内存使用情况：

方式一：通过Metrics接口分析BE内存使用

Metrics统计每10秒更新一次。通过curl访问示例如下。

```
# BE webserver_port默认是8040。
curl -XGET -s http://BE_IP:8040/metrics | grep "^starrocks_be_.*_mem_bytes\|^starrocks_be_tcmalloc_bytes_in_use"
```

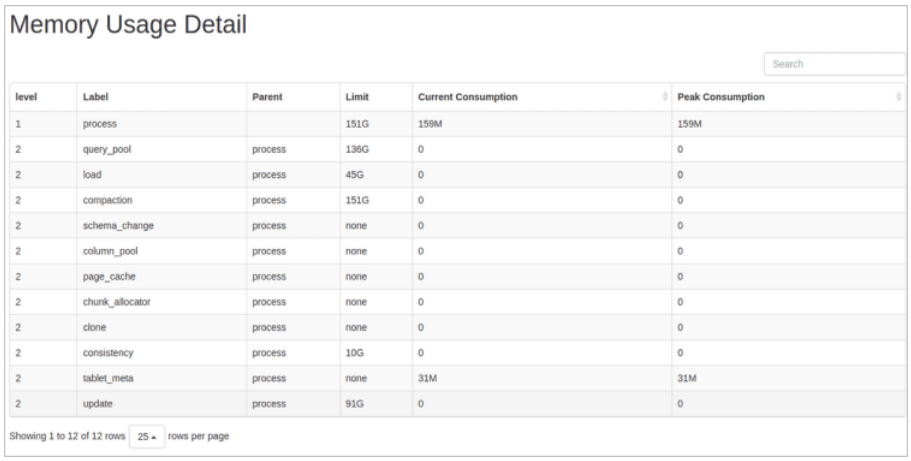
> **说明** 对应的指标含义请参见内存分类。

方式二：通过mem_tracker接口分析BE内存使用

您可以通过浏览器或curl访问。本示例通过浏览器访问。

```
# BE webserver_port默认是8040。
http://be_ip:8040/mem_tracker
```

访问情况如下图所示。



各参数含义如下：

- level：MemTracker是一个树型结构，第一级是BE使用总内存，第二级是分类内存使用。
- Label：标识内存分类，对应的指标含义请参见内存分类。
- Parent：父结点Label。
- Limit：内存使用限制，none表示没有限制。
- Current Consumption：当前内存使用。
- Peak Consumption：峰值内存使用。

5.3.5.3. 资源隔离

本文为您介绍StarRocks资源隔离的功能、基本概念和使用方式。

使用限制

该功能适用于EMR-5.7.0及后续版本的集群。

功能介绍

资源隔离功能可以限制查询任务对计算资源的消耗，目标是让不同租户的查询任务在同一集群执行能兼顾资源隔离并且保证集群资源利用率。

当您发起查询任务时，分类器会根据查询任务的信息进行匹配。匹配度最高的分类器才会生效，最终该分类器所属资源组为这次查询任务的资源组。

基本概念

资源组（Resource Group）

资源组包括BE（Backend）节点计算资源（CPU和内存）的配额，并且一个资源组可以绑定一个或多个分类器。 资源配额的计算方式如下：

- cpu_core_limit：该资源组分配到的CPU核数占BE节点CPU核数的比例，取值为正整数。按照比例，向各个资源组分配执行线程和IO线程的CPU时间片。

  例如，16核的机器，设置三个资源组rg1、rg2和rg3。三个资源组分别对应的cpu_core_limit为2、6和8，则资源组rg1、rg2和rg3分别能分配到的CPU核数为 `BE节点CPU核数×（2/16）` 、 `BE节点CPU 核数×（6/16）` 、 `BE节点CPU核数×（8/16）` 。如果资源空闲，rg1和rg2有负载，但是rg3没有请求，则rg1、rg2分配到的CPU核数分别为 `BE节点CPU核数×（2/8）` 和 `BE节点CPU 核数×（6/8）` 。
- mem_limit：使用query_pool（BE节点中用于查询的内存）的上限，取值范围为0.0~1.0。

  query_pool的查看方式请参见内存管理。

分类器（Classifier）

- 分类器用于匹配查询任务的信息。 匹配度最高的分类器才会生效，最终该分类器所属资源组为查询任务的资源组。一个资源组可以绑定一个或多个分类器。
- 分类器的条件如下：
  - user：用户名。
  - role：用户所属的角色。
  - query_type：查询类型，目前仅支持SELECT。

- source_ip：发起查询的IP地址，类型为CIDR。

- 分类器与查询任务的匹配方式
  - 分类器与查询任务匹配时， 分类器的条件需要与查询任务的信息完全匹配。
  - 如果存在多个分类器的条件与查询任务完全匹配，则需要计算不同分类器的匹配度。只有匹配度最高的分类器才会生效。匹配度的计算方式如下：
    - 如果用户名一致，则匹配度加1。
    - 如果用户所属Role一致，则匹配度加1。
    - 如果查询类型一致，则该部分匹配度为 `1 + 1/分类器的query_type数量` 。
    - 如果发起查询的IP地址一致，则该部分匹配度为 `1 + （32-cidr_prefix）/64` 。

  示例如下：
  - 多个与查询任务匹配的分类器中，分类器的条件数量越多，则其匹配度越高。

    ```
    --B的匹配度比A高，因为B的条件数量多。
    classifier A (user='Alice') classifier B (user='Alice', source_ip = '192.168.**.**/24')
    ```

  - 如果分类器的条件数量相等，则分类器的条件描述越精确，其匹配度越高。

    ```
    --B的匹配度比A高，因为192.168.**.**/24相对于192.168.**.**/16限定的source_ip地址范围更小。
    classifier A (user='Alice', source_ip = '192.168.**.**/16')
    classifier B (user='Alice', source_ip = '192.168.**.**/24')
    --C的匹配度比D高，因为（'select'）比（'insert','select', 'ctas'）限定的查询类型数量少。
    classifier C (user='Alice', query_type in ('select'))
    classifier D (user='Alice', query_type in ('insert','select', 'ctas'))
    ```

功能使用

开启资源组

通过设置相应会话变量开启Pipeline引擎以及资源组功能，设置方式如下。

```
SET enable_pipeline_engine = true;
SET enable_resource_group = true;
```

> **说明** 如果需要设置全局变量，则需要运行 `SET GLOBAL enable_resource_group = true;` 命令。

创建资源组和分类器

- 语法

```
CREATE RESOURCE GROUP group_name
TO (
    user='string',
    role='string',
    query_type in ('select'),
    source_ip='cidr'
) --创建分类器，多个分类器间用英文逗号（,）分隔。
WITH (
    "cpu_core_limit" = "INT",
    "mem_limit" = "m%",
    "concurrency_limit" = "INT",
    "type" = "normal" --资源组的类型，固定取值为normal。
);
```

- 示例

```
CREATE RESOURCE GROUP rg1
TO
    (user='rg1_user1', role='rg1_role1', query_type in ('select'), source_ip='192.168.**.**/24'),
    (user='rg1_user2', query_type in ('select'), source_ip='192.168.**.**/24'),
    (user='rg1_user3', source_ip='192.168.**.**/24'),
    (user='rg1_user4'),
    (db='db1')
WITH (
    'cpu_core_limit' = '10',
    'mem_limit' = '20%',
    'type' = 'normal',
    'big_query_cpu_second_limit' = '100',
    'big_query_scan_rows_limit' = '100000',
    'big_query_mem_limit' = '1073741824'
);
```

指定资源组（可选）

除通过分类器自动指定资源组外，您也可以通过会话变量直接指定资源组。

```
SET resource_group = 'group_name';
```

查看资源组和分类器

语法如下：

- 查询所有的资源组和分类器。

  ```
  SHOW RESOURCE GROUPS ALL;
  ```

- 查询和当前用户匹配的资源组和分类器。

  ```
  SHOW RESOURCE GROUPS;
  ```

- 查询指定的资源组和分类器。

  ```
  SHOW RESOURCE GROUP <yourResourceName>;
  ```

管理资源组配额和分类器

- 修改资源组的配额

  语法如下。

  ```
  ALTER RESOURCE GROUP <yourResourceName> WITH (
      'cpu_core_limit' = '10',
      'mem_limit' = '20%',
      'type' = 'normal'
  );
  ```

- 增加或删除分类器

  语法如下：

  - 添加新的分类器。

    ```
    ALTER RESOURCE GROUP <yourResourceName> ADD CLASSIFIER[,...];
    ```

  - 删除指定的分类器。

    ```
    ALTER RESOURCE GROUP <yourResourceName> DROP (CLASSIFER_ID_1, CLASSIFIER_ID_2, ...);
    ```

  - 删除所有的分类器。

    ```
    ALTER RESOURCE GROUP <yourResourceName> DROP ALL;
    ```

删除资源组

语法如下。

```
DROP RESOURCE GROUP <yourResourceName>;
```

5.3.6. 应用场景

5.3.6.1. 数据湖分析

本文为您介绍StarRocks数据湖分析的常用场景和架构。

背景信息

随着数字产业化和产业数字化成为经济驱动的重要动力，企业的数据分析场景越来越丰富，对数据分析架构的要求也越来越高。新的数据分析场景催生了新的需求，主要包括三个方面：

- 用户希望用更加低廉的成本，更加实时的方式导入并存储任何数量的关系数据（例如，来自业务线应用程序的运营数据库和数据）和非关系数据（例如，来自移动应用程序、IoT设备和社交媒体的运营数据库和数据）。
- 用户希望自己的数据资产受到严密的保护。
- 用户希望数据分析的速度变得更快、更灵活和更实时。

数据湖的出现很好的满足了用户前两方面的需求：

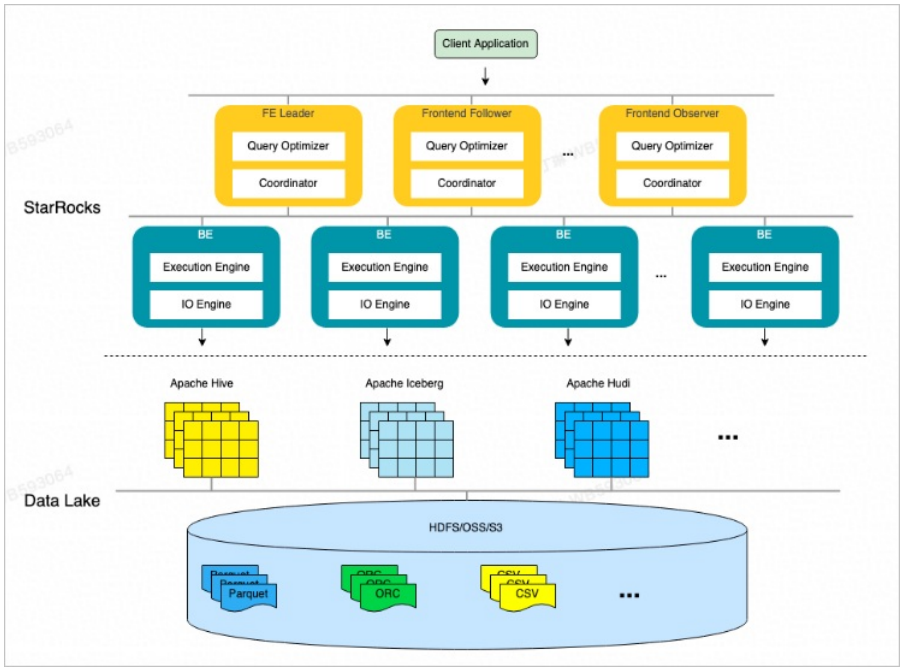
- 存储成本低：他允许用户导入任何数量的实时获得的数据。用户可以从多个来源收集数据，并以其原始形式存储到数据湖中。数据湖拥有极高的水平扩展能力，使得用户能够存储任何规模的数据。同时其底层通常使用廉价的存储方案，使得用户存储数据的成本大大降低。
- 数据安全性高：数据湖通过敏感数据识别、分级分类、隐私保护、资源权限控制、数据加密传输、加密存储、数据风险识别以及合规审计等措施，帮助用户建立安全预警机制，增强整体安全防护能力，让数据可用不可得和安全合规。

什么是数据湖

什么是数据湖，根据Wikipedia的定义，“A data lake is a system or repository of data stored in its natural/raw format, usually object blobs or files”。通俗来说可以将数据湖理解为在廉价的对象存储或分布式文件系统之上包了一层，使这些存储系统中离散的object或者file结合在一起对外展现出一个统一的语义，例如关系型数据库常见的“表”语义等。

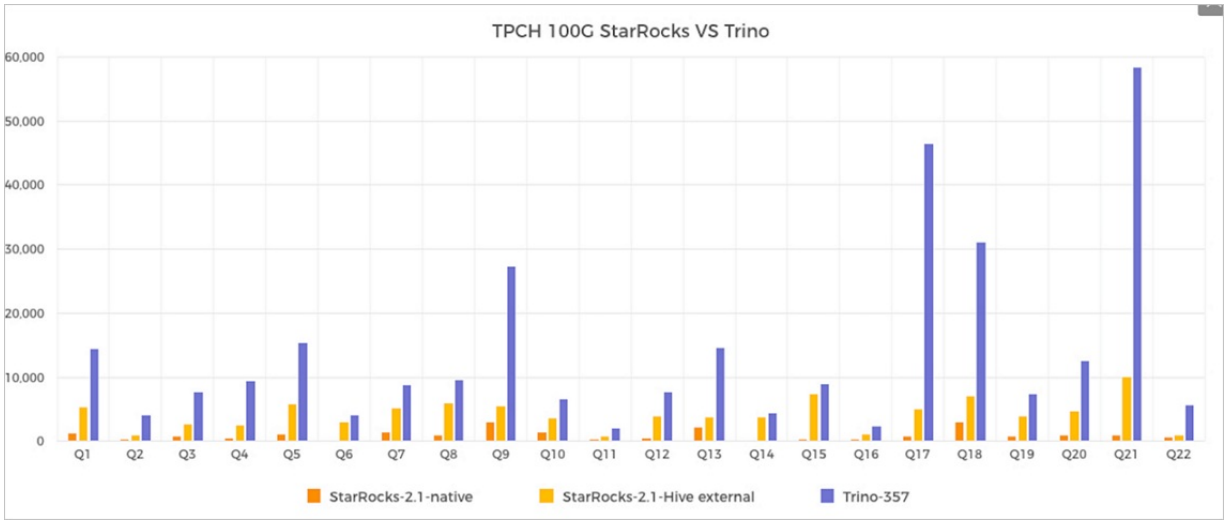
用一句不太准确的话描述数据湖，就是一个存储成本更廉价的“AP数据库”，但是数据湖仅仅提供数据存储和组织的能力，一个完整的数据库不仅要有数据存储的能力，还需要有数据分析能力。

StarRocks数据湖分析架构



从上图可以看到，目前StarRocks基于向量化引擎、Pipeline单机引擎、CBO支持了多种场景数据湖格式的分析。包括，Hive、IceBerg和Hudi。

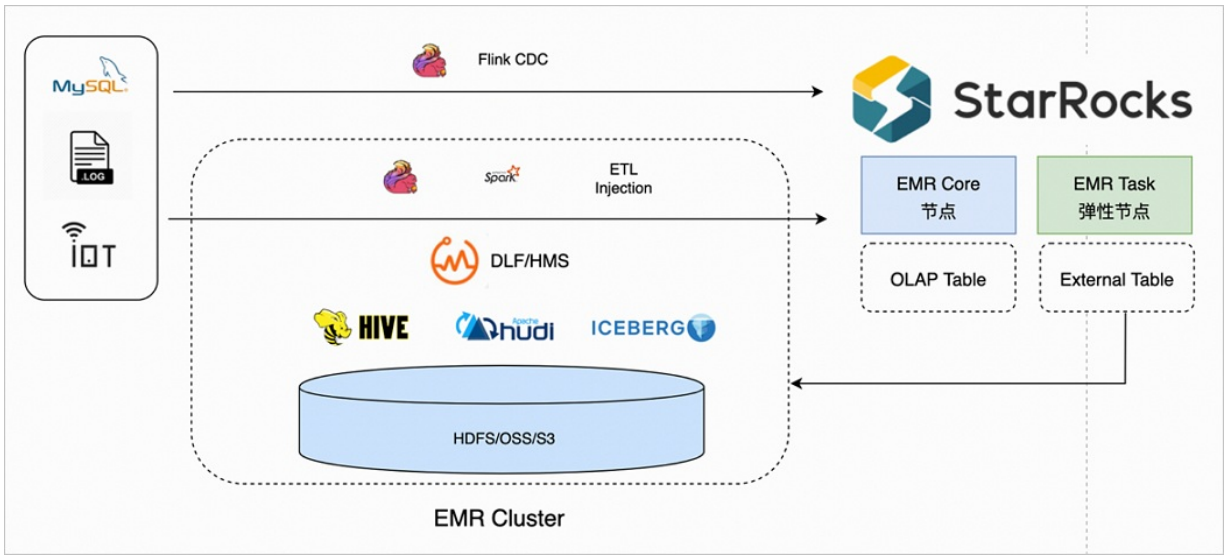
根据测试结果，StarRocks相对于Presto，在数据湖分析领域有比较大的性能优势。StarRocks和Presto（Trino）性能对比如下图所示。



说明 本文的TPC-H的实现基于TPC-H的基准测试，并不能与已发布的TPC-H基准测试结果相比较，本文中的测试并不符合TPC-H基准测试的所有要求。

StarRocks数据湖分析方案

方案1：湖仓一体



从上图可以看到，一个StarRocks集群可以有如下两种节点：

- Core节点：挂载云盘，存储内部OLAP表。
- Task节点：弹性节点，只用于计算。

借助StarRocks对Hive、Hudi和Iceberg等湖格式的分析能力，可以实现同一个集群既有内部OLAP表也有数据湖的表。可以实现以下两个目标：

- 内部OLAP表和数据湖的表进行联合分析。
- Task节点可以进行弹性伸缩，并且不影响内部OLAP表的分析。

方案2：联邦查询

如上图所示，StarRocks支持多种数据源的查询，例如，Hive、Elasticsearch、MySQL、Iceberg和Hudi等。

5.3.6.2. 数据仓库：即席查询场景

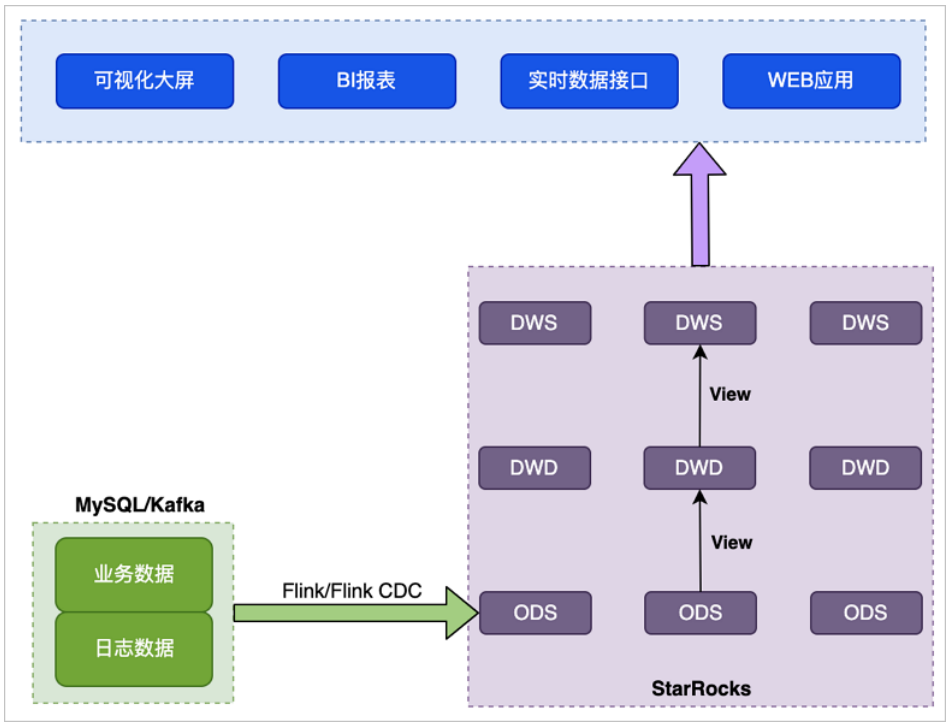
本文为您介绍实时数仓场景中的即席查询场景。即席查询是StarRocks实时数仓场景下非常重要的一种使用方式。

背景信息

随着向量化、CBO（Cost Based Optimizer，基于代价的优化器）、单机多核调度等技术的应用，StarRocks的计算能力逐步提升。很多时候您在使用StarRocks进行数仓分层建模时，大部分将数据建模到DWD层（基础整合层）或DWS层（维度宽度）。随后在实际业务中，运用StarRocks的计算能力，直接查询DWD或DWS层数据，还可以支持灵活地交互式即席查询。

方案架构

使用StarRocks实现实时数仓即席查询的基本架构如下图所示。



整体数据流如下：

1. Flink清洗导入Kafka的日志或者通过Flink-CDC-StarRocks工具读取MySQL Binlog导入StarRocks。根据需要选用明细、聚合、更新或主键各种模型，只物理落地ODS层（格式整理层）。
2. 向上采用StarRocks View视图能力，利用StarRocks向量化极速查询和CBO优化器满足多表关联、嵌套子查询等复杂SQL，查询时现场计算指标结果，保证指标上卷和下钻高度同源一致。

方案特点

该方案主要特点是，计算逻辑在StarRocks侧（现场查询），适用于业务库高频数据更新的场景，实体数据只在ODS或DWD层存储。

方案优势

- 灵活性强，可随时根据业务逻辑调整View。
- 指标修正简单，上层都是View逻辑封装，只需要更新底表数据。

方案缺点

当View的逻辑较为复杂，数据量较多时，查询性能较低。

适用场景

- 数据来源于数据库和埋点系统，适合对QPS要求不高，对灵活性要求比较高，且计算资源较为充足的场景。
- 实时要求非常高，要求写入即可查，更新即反馈。适合有即席查询需求，且资源较为充足，查询复杂度较低的场景。

快速入门示例

具体操作，请参见数据仓库场景一：即席查询。

5.3.6.3. 数据仓库：分钟级准实时场景

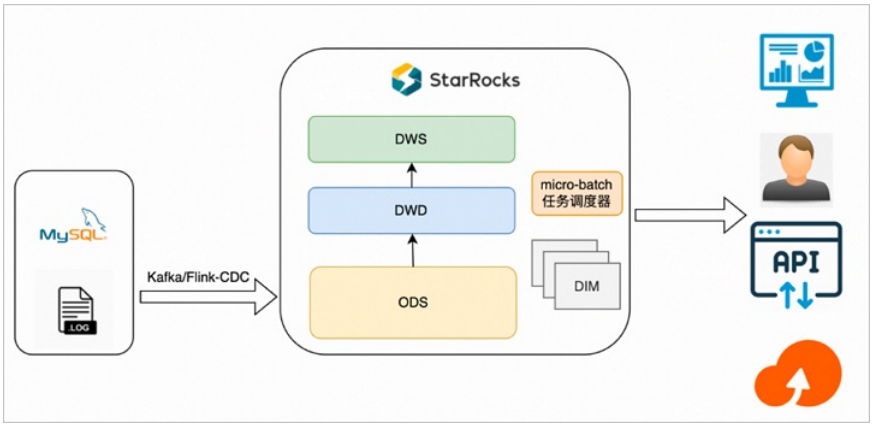
本文为您介绍实时数仓场景中的分钟级准实时场景。

场景介绍

该场景与数据仓库：即席查询场景构建数仓的逻辑基本一致，都是直接在StarRocks中进行数仓分层建模，区别在于分钟级准实时场景将即席查询场景中的视图部分物化成了表，因此具有更高的计算效率，可以支撑更高的QPS查询。

方案架构

分钟级准实时场景的基本架构如下图所示。



整体数据流如下：

1. Flink清洗导入Kafka的日志或者通过Flink-CDC-StarRocks工具读取MySQL Binlog导入StarRocks，根据需要选用明细、聚合、更新、主键各种模型，只物理落地ODS层。
2. 利用第三方任务调度器（例如Airflow）将各层数据表按血缘关系进行任务编排，再按具体的分钟间隔作为一个微批粒度进行任务调度，依次构建ODS之上的各层数据表。

方案特点

该方案主要特点是：计算逻辑在StarRocks侧，适用于高频查询场景，各层数据表按具体的分钟间隔时间作为微批粒度的数据同步。

- 将操作层（ODS层）的数据经过简单的清理、关联，然后存储到明细数据，暂不做过多的二次加工汇总，明细数据直接写入StarRocks。
- DWD或DWS层为实际的物理表，可以通过DataWorks或Airflow等调度工具调度周期性写入数据。
- StarRocks通过表的形式直接对接上层应用，实现应用实时查询。
- 前端实时请求实际的物理表，数据的实时性依赖DataWorks或Airflow调度周期配置，例如5分钟调度、10分钟调度等。

方案优势

- 查询性能强，上层应用只查询最后汇总的数据，相比View，查询的数据量更大，性能会更强。
- 数据重刷快，当某一个环节或者数据有错误时，重新运行DataWorks或Airflow调度任务即可。因为所有的逻辑都是固化好的，无需复杂的订正链路操作。
- 业务逻辑调整快，当需要新增或者调整各层业务，可以基于SQL所见即所得开发对应的业务场景，业务上线周期缩短。

方案缺点

因为引入了更多的加工和调度，所以时效性低于即席查询场景。

适用场景

数据来源于数据库和埋点系统，对QPS和实时性均有要求，适合80%实时数仓场景使用，能满足大部分业务场景需求。

快速入门示例

具体操作，请参见数据仓库场景二：分钟级准实时数仓。

5.3.6.4. 数据仓库：增量数据实时统计场景

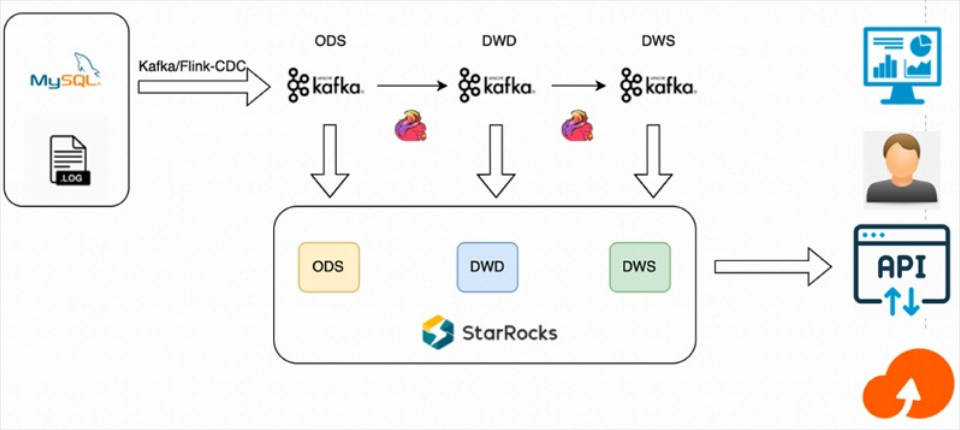
本文为您介绍实时数仓场景中的增量数据实时统计场景。

场景介绍

因为部分场景对数据延迟非常敏感，数据产生的时候必须完成加工，所以此时您可以通过增量数据实时统计的方式，提前使用Flink将明细层、汇总层等层数据进行汇聚，汇聚之后把结果集存下来再对外提供服务。

方案架构

增量数据实时统计的基本架构如下图所示。



整体数据流如下：

1. 直接使用Flink构建实时数仓，由Flink进行清洗加工转换和聚合汇总，将各层结果集写入Kafka中。
2. StarRocks从Kafka分别订阅各层数据，将各层数据持久化到StarRocks中，用于之后的查询分析。

方案特点

该方案主要特点如下：

- 增量计算的数据由Flink进行清洗加工转换和聚合汇总，各层应用数据通过Kafka分别持久化到StarRocks中。
- Flink加工的结果集可以采取双写的方式，一方面继续投递给下一层消息流Topic，一方面Sink到同层的StarRocks中；也可以采用单写Kafka再通过StarRocks实时消费Kafka对应Topic上的数据，方便后续历史数据的状态检查与刷新。
- StarRocks通过表的形式直接对接上层应用，实现应用实时查询。

方案优势

- 实时性强，能满足业务对实时性敏感的场景。
- 指标修正简单，与传统增量计算方式不一样的是，该方案将中间的状态也持久存储在StarRocks中，提升了后续分析的灵活性，当中间数据质量有问题时，直接对表修正，重刷数据即可。

方案缺点

- 大部分实时增量计算都依赖Flink，对使用者Flink的技能和熟练度要求会更高一些。
- 不适合数据频繁更新，无法累加计算的场景。
- 不适合多流Join等计算复杂资源开销大场景。

适用场景

实时需求简单，数据量不大，以埋点数据统计为主的数据，实时性最强。

快速入门示例

具体操作，请参见数据仓库场景三：增量数据实时统计。

5.3.7. 常见问题

本文汇总了StarRocks使用时的常见问题。

- 业务测试评估
  - 硬件资源有什么要求？
  - 软件配置有什么要求？
  - 数据模型和表定义
    - 生产环境下的副本数应该设置为多少？
    - 如何分区？
    - 如何分桶？
    - 如何设计排序键？
    - 如何合理的选择数据类型？
  - 导入
    - 如何选择导入方式？
    - 影响导入性能的因素都有哪些？
- 业务调优
- 常见使用问题
  - 报错提示ERROR 1064 (HY000): Failed to find enough host in all backends. need: 3，该如何处理？
  - 导入数据时发现BE服务日志中出现Too many open files问题，该如何处理？
  - 报错提示中包含increase config 'load_process_max_memory_limit_percent'，该如何处理？
  - 如何选择数据模型？
  - StarRocks数据模型count的实现机制是怎么样的？
  - 如何减少/mnt/disk1/starrocks/storage/trash/目录磁盘占用？
  - 创建物化视图时报错，该如何处理？
  - 数据查询缓慢，如何处理？

硬件资源有什么要求？

- 整体机器配置

  BE推荐16核64 GB以上，FE推荐8核16 GB以上。

  生产环境FE通常是16核，32 GB或64 GB内存，200~500 GB NVMe。
- 磁盘
  - HDD和SSD都可以。
  - 磁盘容量评估，可以按照压缩比3，磁盘利用率70%~75%作为上限来进行评估。
  - 如果导入到SR的数据是Parquet或Orc的Hive数据，则压缩比按照1：1来评估。

    例如，Hive数据是3T，则导入SR的数据也是3T。
- CPU
  - CPU必须支持AVX2指令集，可以通过 `cat /proc/cpuinfo |grep avx2` 命令判断。
  - 向量化技术需要配合CPU指令集才能发挥更好地作用，所以没有相关指令集的话建议更换机器。
- 网络

  建议选择万兆网卡和万兆交换机。

软件配置有什么要求？

软件配置的详细的信息，请参见参数配置。

生产环境下的副本数应该设置为多少？

通常生产环境下副本数2~3即可，建议设置为3。

如何分区？

合理的分区可以有效的裁剪scan的数据量。

- 通常是从业务本身对数据进行管理的角度来选择分区键。例如选用时间或者区域作为分区键。
- 如果需要自动创建分区，则可以使用动态分区。

如何分桶？

- 选择高基数的列来作为分桶键，避免Bucket之间数据倾斜。
  - 如果有唯一ID，则建议使用唯一ID分桶。
  - 如果碰到数据倾斜严重的，则可以使用多列作为分桶键，但通常不要使用太多列作为分桶键。
- Tablet的最佳大小可以按下面进行评估，基于以下参数值和总数据量可以预估出Bucket的数目。
  - 原始非压缩数据，例如CSV格式，通常每个tablet设置为1 GB ~10 GB之间。
  - Parquet格式的数据，建议1 GB左右。
- 在机器比较少的情况下，如果想充分利用机器资源可以考虑使用 `BE数量 * cpu core / 2` 来设置Bucket数量。

如何设计排序键？

排序键要根据查询的特点来设计：

- 将经常作为过滤条件和Group BY的列作为排序键，可以加速查询。
- 如果是有大量点查，建议把查询点查的ID放到第一列。

  例如，查询命令为 `select sum(revenue) from lineorder where user_id='aaa100';` ，并且有很高的并发，强烈推荐把 `user_id` 作为排序键的第一列。
- 如果查询的主要是聚合和scan比较多，建议把低基数的列放在前面。

  例如，查询命令为 `select region, nation, count(*) from lineorder_flat group by region, nation` ，则建议把 `region` 作为第一列， `nation` 作为第二列会更合适。

如何合理的选择数据类型？

尽量使用精准的数据类型。例如，能够使用整形就不要使用字符串类型，能够使用int就不要使用bigint，精确的数据类型能够更好的发挥数据库的性能。

如何选择导入方式？

导入方式的选择可以参见导入概述。

影响导入性能的因素都有哪些？

通常影响导入性能的因素有下面几个，正常到50 Mbit/s~100 Mbit/s是没有问题的：

- 机器内存

  当tablet比较多的时候，对于内存消耗比较大。建议单个tablet大小按照如何分桶？中介绍的进行评估。
- 磁盘IO能力和网络带宽
- 导入批次和频率
  - Stream Load批次大小建议10 MB~100 MB。
  - Broker Load还好，因为Broker Load针对的场景都是批次大小比较大的情况。
  - 导入频率不要太高，SATA盘1s不超过一个任务。

当Routine Load出现性能问题时，如何进行参数调优？

参数调优策略

当Routine Load出现性能问题时，您可以考虑从如下几个维度来进行参数调优：

- 任务调度周期

  您可以通过缩短任务调度周期（即修改参数max_batch_interval）加速数据消费。但是，缩短任务调度周期可能会带来更多的CPU资源消耗。

  > **注意** 任务调度周期最小值为5s。
- 任务并行度

  在Partition数量和BE数量较多时，您可以调大以下参数来加速任务执行。但是，增加并行度可能会带来更多的CPU资源消耗。
  - max_routine_load_task_concurrent_num
  - desired_concurrent_number

  单个Routine Load任务会根据Kafka Topic Partition数和BE数等被拆分为若干个子任务，分发至多个BE执行。此处的任务并行度，实际上是指单个Routine Load拆分成的子任务个数。

  实际的任务并行度参照如下的计算公式。

  ```
  concurrent_num = Min( Min( partition_num, Min( desired_concurrent_num, alive_be_num ) ),Config.max_routine_load_task_concurrent_num )
  ```
- 任务批量大小

- routine_load_task_consume_second：通过增大单次读取持续时间加速数据消费。
- max_routine_load_batch_size：通过增大单次读取的数据量加速数据消费。

您可以根据以下日志来判定当前的批量参数设置是否过小。正常情况下，该日志的 `left_bytes` 字段应该>=0，表示一次读取的数据量还未超过max_routine_load_batch_size上限。否则，说明max_routine_load_batch_size过小。

```
I0325 20:27:50.410579 15259 data_consumer_group.cpp:131] consumer group done:
41448fb1a0ca59ad-30e34dabfa7e47a0. consume time(ms)=3261, received rows=179190,
received bytes=9855450, eos: 1, left_time: -261, left_bytes: 514432550, blocking
get time(us): 3065086, blocking put time(us): 24855
1
```

Routine Load参数

参数	类型	默认值	描述
max_routine_load_job_num	fe.conf	100	NEED_SCHEDULE、RUNNING、PAUSED状态的routine load任务上限。
max_routine_load_task_concurrent_num	fe.conf	5	单个Routine Load任务的最大并行度。
max_routine_load_task_num_per_be	fe.conf	5	单个BE节点上调度分配的最大Routine Load任务数。
max_routine_load_batch_size	fe.conf	500 MB	最大单次批量读取Kafka数据量。
routine_load_task_consume_second	fe.conf	3	最大单次批量读取Kafka数据时间。
routine_load_task_timeout_second	fe.conf	15	单个Routine Load任务running超时时间。
max_consumer_num_per_group	be.conf	3	单个consumer group最大consumer数。
desired_concurrent_number	properties	3	期望Routine Load任务的并行度。实际的任务并行度参照如下的计算公式 `concurrent_num = Min( Min( partition_num, Min(desired_concurrent_num, alive_be_num ) ),Config.max_routine_load_task_concurrent_num )` 。
max_batch_interval	properties	10s	Routine Load任务调度周期。
max_batch_rows	properties	200000	该参数只用于定义错误检测窗口范围，窗口的范围是 `10 * maxbatch-rows` 。
max_error_number	properties	0	采样窗口内，允许的最大错误行数。必须大于等于0。默认是0，即不允许有错误行。 注意 被where条件过滤掉的行不算错误行。
strict_mode	properties	true	是否开启严格模式，默认为开启。如果开启后，非空原始数据的列类型变换为NULL，则会被过滤。

报错提示 `ERROR 1064 (HY000): Failed to find enough host in all backends. need: 3` ，该如何处理？

您可以在建表属性中添加 `"replication_num" = "1"` 信息。

导入数据时发现BE服务日志中出现Too many open files问题，该如何处理？

您可以根据以下步骤处理：

1. 调整系统句柄数。
2. 调小base_compaction_num_threads_per_disk和cumulative_compaction_num_threads_per_disk（默认都是1）的参数值。修改配置项的具体操作，请参见修改配置项。
3. 如果问题还未解决，建议扩容或者降低导入频率。

报错提示中包含 `increase config 'load_process_max_memory_limit_percent'` ，该如何处理？

导入数据的时候出现类似如下错误时，建议您查看并调大load_process_max_memory_limit_bytes和load_process_max_memory_limit_percent的参数值。修改配置项的具体操作，请参见修改配置项。

```
/root/starrocks/be/src/runtime/plan_fragment_executor.cpp:209 _sink->open(runtime_state())
W0516 15:58:30.892597 28779 fragment_mgr.cpp:193] Fail to open fragment 334f7f8c-0727-5909-afa3-9eb1db88af95: Internal error: intolerable failure in opening node channels
/root/starrocks/be/src/runtime/plan_fragment_executor.cpp:209 _sink->open(runtime_state())
W0516 15:58:30.898306 28779 stream_load_executor.cpp:94] fragment execute failed, query_id=334f7f8c07275909-afa39eb1db88af94, err_msg=intolerable failure in opening node channels, id=334f7f8c07275909-afa3
9eb1db88af94, job_id=-1, txn_id=633657, label=156936fa-1134-4c4c-b473-fa05b4b8bf8c
W0516 15:58:30.907560 28985 stream_load.cpp:315] append body content failed. errmsg=cancelledid=334f7f8c07275909-afa39eb1db88af94, job_id=-1, txn_id=633657, label=156936fa-1134-4c4c-b473-fa05b4b8bf8c
W0516 15:58:34.353019 28993 data_sink.cpp:104] Ignore option use_vectorized=false
W0516 15:58:34.353847 28795 tablet_sink.cpp:605] NodeChannel[15381-10004]: tablet open failed, load_id=cb4e78a6-f026-c3d9-a700-aba44764c288, txn_id=633662, node=10.28.  8060, errmsg=memory limit exc
eeded, please reduce load frequency or increase config `load_process_max_memory_limit_percent` or add more BE nodes
W0516 15:58:34.353901 28795 tablet_sink.cpp:605] NodeChannel[15381-10003]: tablet open failed, load_id=cb4e78a6-f026-c3d9-a700-aba44764c288, txn_id=633662, node=10.28.  8060, errmsg=memory limit exc
eeded, please reduce load frequency or increase config `load_process_max_memory_limit_percent` or add more BE nodes
W0516 15:58:34.353909 28795 tablet_sink.cpp:605] NodeChannel[15381-149009]: tablet open failed, load_id=cb4e78a6-f026-c3d9-a700-aba44764c288, txn_id=633662, node=10.28  8060, errmsg=memory limit exc
eeded, please reduce load frequency or increase config `load_process_max_memory_limit_percent` or add more BE nodes
W0516 15:58:34.353914 28795 tablet_sink.cpp:605] NodeChannel[15381-10002]: tablet open failed, load_id=cb4e78a6-f026-c3d9-a700-aba44764c288, txn_id=633662, node=10.28.  8060, errmsg=memory limit exc
eeded, please reduce load frequency or increase config `load_process_max_memory_limit_percent` or add more BE nodes
W0516 15:58:34.353976 28795 tablet_sink.cpp:605] NodeChannel[15381-149010]: tablet open failed, load_id=cb4e78a6-f026-c3d9-a700-aba44764c288, txn_id=633662, node=10.28  8060, errmsg=memory limit exc
eeded, please reduce load frequency or increase config `load_process_max_memory_limit_percent` or add more BE nodes
W0516 15:58:34.353983 28795 tablet_sink.cpp:613] open failed, load_id=cb4e78a6f026c3d9-a700aba44764c288
W0516 15:58:34.353992 28795 plan_fragment_executor.cpp:188] fail to open fragment, instance_id=cb4e78a6-f026-c3d9-a700-aba44764c289, status=Internal error: intolerable failure in opening node channels
/root/starrocks/be/src/runtime/plan_fragment_executor.cpp:209 _sink->open(runtime_state())
W0516 15:58:34.354202 28795 fragment_mgr.cpp:193] Fail to open fragment cb4e78a6-f026-c3d9-a700-aba44764c289: Internal error: intolerable failure in opening node channels
/root/starrocks/be/src/runtime/plan_fragment_executor.cpp:209 _sink->open(runtime_state())
W0516 15:58:34.600633 28795 stream_load_executor.cpp:94] fragment execute failed, query_id=cb4e78a6f026c3d9-a700aba44764c288, err_msg=intolerable failure in opening node channels, id=cb4e78a6f026c3d9-a700
aba44764c288, job_id=-1, txn_id=633662, label=156936fa-1134-4c4c-b473-fa05b4b8bf8c
W0516 15:58:34.600664 28993 stream_load.cpp:132] Fail to handle streaming load, id=cb4e78a6f026c3d9-a700aba44764c288 errmsg=intolerable failure in opening node channels
W0516 15:58:39.555845 28995 data_sink.cpp:104] Ignore option use_vectorized=false
```

如何选择数据模型？

StarRocks的数据模型主要有以下四种，您可以根据实际情况选择。

数据模型	适用场景
duplicate key	• 数据更新不频繁。 • 查询模式灵活没有预聚合的模式。 • 需要保留原始数据。
agg模型	• 只追加不更新数据。 • 业务方查询都包含聚合函数，例如min、max或sum等。 • 不需要查询原始明细数据。
uniq key	适合于有更新和实时分析的场景。
primary key	适合于有更新和实时分析的场景。 如果有部分列更新的场景建议列在200列以内。

StarRocks数据模型count的实现机制是怎么样的？

StarRocks的数据模型主要有四种，分别为duplicate key、uniq key、agg模型和primary key模型，他们对于count的实现有比较大的区别。具体区别如下：

- duplicate key：该模型不需要做merge操作，所以count比较快。
- uniq key和agg模型：对count操作的实现涉及多版本merge的操作，所以相对要慢一些。

  如果key是string类型，则理论上count操作会更慢。
- primary key：在读取数据的时候因为有内存中的索引和delete vector，不需要做merge操作，所以count操作比uniq key和agg模型会快些。

  如果有更新操作，建议使用该模型。

如何减少 `/mnt/disk1/starrocks/storage/trash/` 目录磁盘占用？

`/mnt/disk1/starrocks/storage/trash/` 目录中存储的是删除的数据。如果您想减少该目录的磁盘占用，可以通过调小*be.conf*中的trash_file_expire_time_sec参数，控制trash目录保留时间。默认值是259200（72小时）。

创建物化视图时报错，该如何处理？

- 问题现象：具体报错如下图所示。

```
mysql> CREATE MATERIALIZED VIEW test_bl1_mvte AS select timed, event, count(userId) as PV from test_bl1 group by tim
ed, event;
ERROR 1064 (HY000): table [test_bl1] is not stable. Some tablets of this table may not be healthy or are being sched
uled. You need to repair the table first or stop cluster balance. See 'help admin;'.
mysql>
```

- 解决方法：
  i. 执行命令 `show proc "/cluster_balance";` 和 `show proc "/statistic";` 。
  ii. 查看是否有tablet正在进行rebalance：
  - 如果有，则等待执行完成。
  - 如果没有，则可以执行命令 `set disable_balance=true` ，然后发起创建物化视图操作。

数据查询缓慢，如何处理？

建议处理方法如下：

- 开启Profile，之后在StarRocks UI上查看Profile信息。

  执行命令 `SET is_report_success = true;` 开启Profile。

- 基本排查方法：
  i. 调整并行度。
  - Pipeline

    ```
    SET pipeline_dop = 8;
    SET enable_pipeline_engine = true;
    ```

  - 非Pipeline

    ```
    SET enable_pipeline_engine=false;
    SET parallel_fragment_exec_instance_num=8;
    ```

  ii. 查看tablet分布。

  ```
  show data xxx;
  ```

  > **说明** 建议tablet大小在1 GB~10 GB。

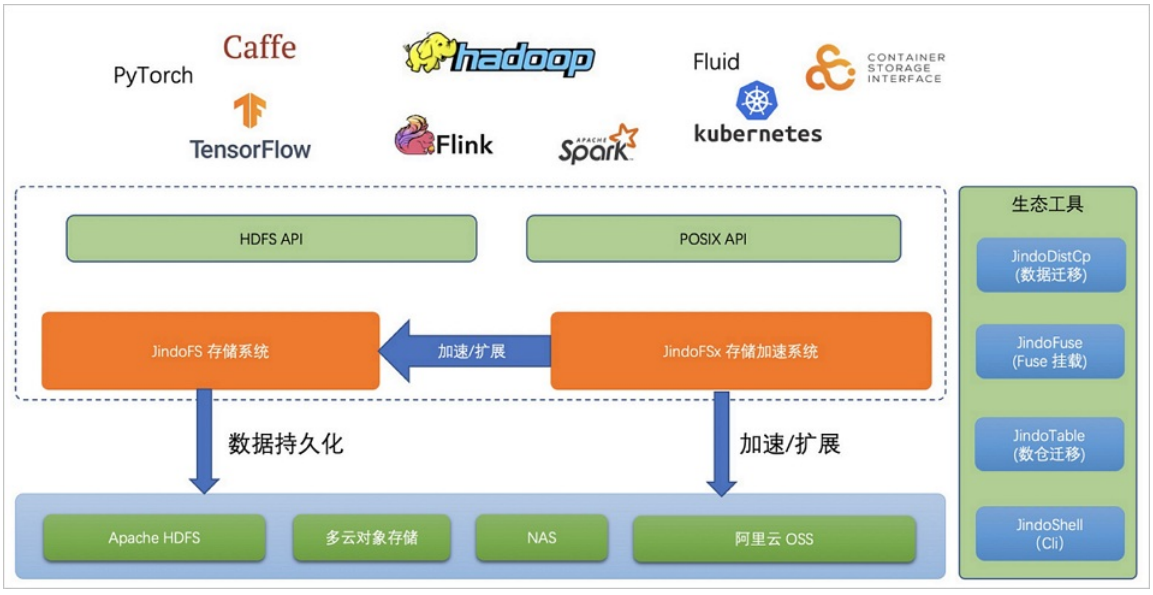
  iii. 查看建表。
  a. 通过Profile判断iotime。如果很大，可以删除一些不必要的索引，例如，删除建得比较多的bitmap索引。
  b. 查看表数据模型，选择合适的数据模型。例如，uniq key模型在compaction没有做完的时候下推无法适用，容易导致查询慢。

5.4. JindoData

5.4.1. JindoData概述

JindoData是阿里云开源大数据团队自研的数据湖存储加速套件，面向大数据和AI生态，为阿里云和业界主要数据湖存储系统提供全方位访问加速解决方案。

JindoData套件基于统一架构和内核实现，主要包括JindoFS存储系统（原JindoFS Block模式）、JindoFSx存储加速系统（原JindoFS Cache模式），JindoSDK大数据万能SDK和全面兼容的生态工具（JindoFuse、JindoDistCp）以及插件支持。



JindoData主要组件如下：

- JindoFS存储系统
- JindoFSx存储加速系统
- 生态支持和工具

JindoFS存储系统

基于阿里云OSS的云原生存储系统，二进制兼容Apache HDFS，并且与Apache HDFS基本功能对齐，提供优化的HDFS使用和平迁体验。JindoFS存储系统是原JindoFS Block模式的全新升级版本。

阿里云OSS-HDFS服务（JindoFS服务）是JindoFS存储系统在阿里云上的服务化部署形态，和阿里云OSS深度融合，开箱即用，无须在自建集群部署维护JindoFS，即免运维。

OSS-HDFS服务的详细信息，请参见OSS-HDFS服务概述。

JindoFSx存储加速系统

JindoFSx（JindoData服务）是原JindoFS Cache模式的全新升级版本，是面向大数据和AI生态的云原生数据湖存储加速系统，为大数据和AI应用访问各种云存储提供访问加速，支持数据缓存、元数据缓存和P2P加速等功能。JindoFSx支持管理多个后端存储系统，可以通过统一命名空间进行管理，也可以兼容各系统原生的访问协议，也支持为这些系统提供统一的权限管理。原生优化支持阿里云OSS和阿里云OSS-HDFS服务，同时也支持业界多云对象存储（例如，AWS S3）、Apache HDFS和NAS。

生态支持和工具

- 支持JindoSDK。

  支持面向云时代的大数据Hadoop SDK和HDFS接口，内置优化访问阿里云OSS，较Hadoop社区版本性能大幅提升。同时支持JindoFS存储系统和服务、JindoFSx存储加速系统，支持多云对象存储。

- 支持JindoShell CLI。

  JindoData除了支持HDFS Shell命令，还提供了一套JindoShell CLI命令，从功能、性能上大幅扩展和优化一些数据访问操作。

- 支持JindoFuse POSIX。

  JindoData为阿里云OSS、JindoFS存储系统和服务、JindoFSx存储加速系统提供了POSIX支持。

- 支持JindoDistCp数据迁移。

  IDC机房数据（HDFS）上云迁移和多云迁移利器，支持多种存储数据迁移到阿里云OSS和JindoFS服务，使用上类似Hadoop DistCp。

- 支持JindoTable。

  结合计算引擎的使用推出的一套解决方案，支持Spark、Hive和Presto等引擎，以及表格式数据的管理功能。

- 生态插件。

  除了默认提供JindoSDK支持Hadoop，另外还支持Flink Connector等插件。

5.4.2. JindoData 4.3.0

5.4.2.1. JindoData 4.3.0版本简介

JindoData 4.x是阿里云E-MapReduce产品SmartData自研组件（SmartData 3.8.0版本）架构升级之后首次发布的版本，重点对接和支持了阿里云OSS存储产品和阿里云OSS-HDFS服务（JindoFS服务）。本文为您介绍JindoData 4.3.0版本支持的功能。

功能介绍

JindoData是阿里云开源大数据团队自研的数据湖存储加速套件，面向大数据和AI生态，为阿里云和业界主要数据湖存储系统提供全方位访问加速解决方案。

JindoSDK和工具支持

- JindoSDK支持多云存储，包括Amazon S3、COS和OBS。
- JindoSDK提供JindoTable工具。
- JindoSDK优化了Flink Connector插件。
- JindoSDK完善了JindoDistCp。

JindoFSx存储加速系统

- JindoFSx支持多云存储，包括Amazon S3、COS和OBS。
- JindoFSx优化了数据缓存及元数据缓存。
- JindoFSx支持Kerberos + Ranger的鉴权方案。
- JindoFSx大幅完善了可观测性指标。
- JindoFSx完成与Fluid的对接。

JindoFS存储系统

- JindoFS支持POSIX Lock和Fallocate能力。
- JindoFS支持老版本JindoFS Block模式集群升级。

JindoFuse POSIX支持

- JindoFuse新增XAttr相关接口支持，包括setxattr、getxattr、listxattr和removexattr。

- JindoFuse支持POSIX Lock和Fallocate能力。
- JindoFuse支持OSS可追加写对象，包括append、flush和边写边读功能。

5.4.2.2. 基础功能

5.4.2.2.1. 阿里云OSS透明缓存加速

JindoFSx存储加速系统提供了透明缓存的使用方式，兼容原生OSS存储方式，文件以对象的形式存储在OSS上，每个文件根据实际访问情况会在本地进行缓存，提升访问OSS的效率，同时兼容了原有OSS文件形式，数据访问上能够与其他OSS客户端完全兼容，作业访问OSS的方式无需做任何修改。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> **说明** 本文以EMR-3.40.0版本为例介绍。

操作流程

1. 步骤一：配置AccessKey
2. 步骤二：配置JindoSDK
3. 步骤三：磁盘空间水位控制

步骤一：配置AccessKey

1. 进入JindoData服务的**common**页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 单击JindoData服务区域的**配置**。
   v. 在JindoData服务的**服务配置**区域，单击**common**页签。
2. 新增配置。
   i. 单击**新增配置项**。
   ii. 在**新增配置项**对话框中，新增以下配置项。

   新增配置项的具体操作，请参见添加配置项。

   - 全局方式配置（所有Bucket使用同一种方式）

参数	描述
jindofsx.oss.accessKeyId	OSS的AccessKey ID。
jindofsx.oss.accessKeySecret	OSS的AccessKey Secret。
jindofsx.oss.endpoint	OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。

   - 按照Bucket配置

参数	描述
jindofsx.oss.bucket.XXX.accessKeyId	`XXX`的Bucket的AccessKey ID。
jindofsx.oss.bucket.XXX.accessKeySecret	`XXX`的Bucket的AccessKey Secret。
jindofsx.oss.bucket.XXX.endpoint	`XXX`的Bucket的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
jindofsx.oss.bucket.YYY.accessKeyId	`YYY`的Bucket的AccessKey ID。
jindofsx.oss.bucket.YYY.accessKeySecret	`YYY`的Bucket的AccessKey Secret。
jindofsx.oss.bucket.YYY.endpoint	`YYY`的Bucket的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。

   > **说明** `XXX`和`YYY`为OSS Bucket的名称。

   iii. 单击**确定**。

iv. 在**确认修改配置**对话框中，输入执行原因，单击**确定**。

3. 重启服务。
   i. 在JindoData服务页面，选择**更多操作 > 重启**。
   ii. 在**重启JindoData服务**对话框中，输入执行原因，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

步骤二：配置JindoSDK

> **注意** 此配置为客户端配置，无需重启JindoData服务。

1. 进入HDFS服务的**core-site**页签。
   i. 在**集群服务**页签，单击HDFS服务区域的**配置**。
   ii. 在HDFS服务的**服务配置**区域，单击**core-site**页签。
2. 修改以下配置项。

   修改配置项的具体操作，请参见修改配置项。

内容	参数	描述
配置OSS实现类	fs.AbstractFileSystem.oss.impl	固定值为com.aliyun.jindodata.oss.OSS。
	fs.oss.impl	固定值为com.aliyun.jindodata.oss.JindoOssFileSystem。
	fs.xengine	固定值为jindofsx。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
启用缓存加速功能	fs.jindofsx.data.cache.enable	数据缓存开关： ○ false（默认值）：禁用数据缓存。 ○ true：启用数据缓存。 **说明** 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。

3. 保存配置。
   i. 单击**服务配置**区域的**保存**。
   ii. 在**确认修改配置**对话框中，输入执行原因，开启**自动更新配置**，单击**确定**。
4. 新增配置项。
   i. 单击**服务配置**区域的**新增配置项**。

ii. 在**新增配置项**对话框中，新增以下配置项。

新增配置项的具体操作，请参见添加配置项。

内容	参数	描述
配置Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
其他缓存加速功能	fs.jindofsx.meta.cache.enable	元数据缓存开关： ■ false（默认值）：禁用元数据缓存。 ■ true：启用元数据缓存。
	fs.jindofsx.slice.cache.enable	小文件缓存优化开关： ■ false（默认值）：禁用小文件缓存。 ■ true：启用小文件缓存。
	fs.jindofsx.ram.cache.enable	内存缓存开关： ■ false（默认值）：禁用内存缓存。 ■ true：启用内存缓存。
	fs.jindofsx.short.circuit.enable	短路读开关： ■ true（默认值）：打开短路读开关。 ■ false：关闭短路读开关。

iii. 单击**确定**。

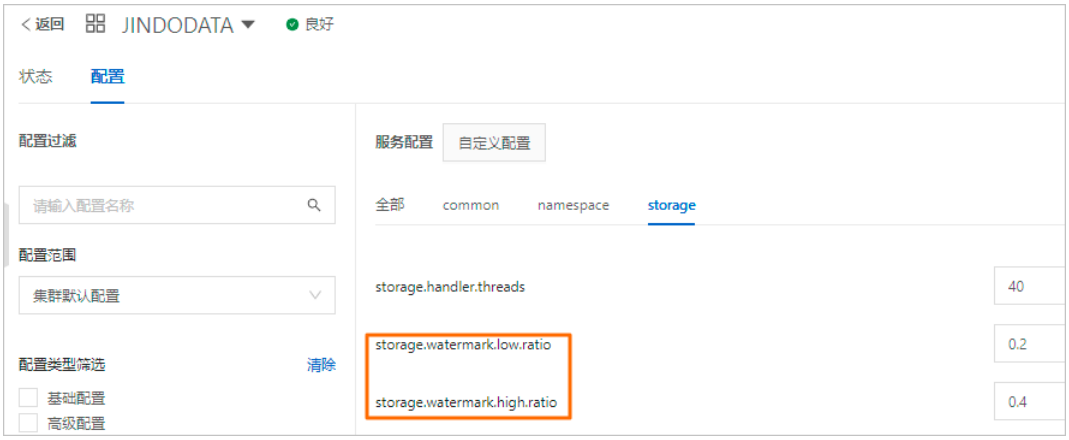
iv. 在**确认修改配置**对话框中， 输入执行原因，单击**确定**。

步骤三：磁盘空间水位控制

缓存启用后，JindoFSx服务会自动管理本地缓存备份，通过水位清理本地缓存，请您根据需求配置一定的比例用于缓存。JindoFSx后端基于OSS，可以提供海量的存储，但是本地盘的容量是有限的，因此JindoFSx会自动淘汰本地较冷的数据备份。您可以通过修改**storage.watermark.high.ratio**和**storage.watermark.low.ratio**两个参数来调节本地存储的使用容量，值均为0～1的小数，表示使用磁盘空间的比例。

1. 修改磁盘水位配置。

在JindoData服务的**服务配置**区域的**storage**页签，修改以下参数。



参数	描述
storage.watermark.low.ratio	表示使用量的下水位比例，触发清理后会自动清理冷数据，将JindoFS数据目录占用空间清理到下水位。默认值：0.2。
storage.watermark.high.ratio	表示磁盘使用量的上水位比例，每块数据盘的JindoFS数据目录占用的磁盘空间到达上水位即会触发清理。默认值：0.4。

> **说明** 修改该参数时，下水位必须小于上水位，设置合理的值即可。

2. 保存配置。
   i. 单击**服务配置**区域的**保存**。
   ii. 在**确认修改配置**对话框中，输入执行原因，开启**自动更新配置**，单击**确定**。
3. 重启服务。
   i. 选择右上角的**更多操作 > 重启**。
   ii. 在**重启JindoData服务**对话框中，输入执行原因，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

相关文档

JindoSDK包含一些高级调优参数，配置方式以及配置项的详细信息，请参见JindoSDK高级参数配置。

5.4.2.2.2. 阿里云OSS统一挂载缓存加速

本文为您介绍JindoFSx支持阿里云OSS统一挂载缓存加速的使用方式。

背景信息

JindoSDK包含一些高级调优参数，配置方式以及配置项请参见JindoSDK高级参数配置。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> **说明** 本文以EMR-3.40.0版本为例介绍。

操作流程

1. 步骤一：配置阿里云OSS AccessKey
2. 步骤二：配置JindoSDK
3. 步骤三：磁盘空间水位控制
4. 步骤四：挂载阿里云OSS
5. 步骤五：访问阿里云OSS

步骤一：配置AccessKey

1. 进入JindoData服务的**common**页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 单击JindoData服务区域的**配置**。
   v. 在JindoData服务的**服务配置**区域，单击**common**页签。
2. 新增配置。
   i. 单击**新增配置项**。

ii. 在**新增配置项**对话框中，新增以下配置项。

新增配置项的具体操作，请参见添加配置项。

- 全局方式配置（所有Bucket使用同一种方式）

参数	描述
jindofsx.oss.accessKeyId	OSS的AccessKey ID。
jindofsx.oss.accessKeySecret	OSS的AccessKey Secret。
jindofsx.oss.endpoint	OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。

- 按照Bucket配置

参数	描述
jindofsx.oss.bucket.XXX.accessKeyId	`XXX`的Bucket的AccessKey ID。
jindofsx.oss.bucket.XXX.accessKeySecret	`XXX`的Bucket的AccessKey Secret。
jindofsx.oss.bucket.XXX.endpoint	`XXX`的Bucket的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
jindofsx.oss.bucket.YYY.accessKeyId	`YYY`的Bucket的AccessKey ID。
jindofsx.oss.bucket.YYY.accessKeySecret	`YYY`的Bucket的AccessKey Secret。
jindofsx.oss.bucket.YYY.endpoint	`YYY`的Bucket的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。

> **说明** `XXX`和`YYY`为OSS Bucket的名称。

iii. 单击**确定**。

iv. 在**确认修改配置**对话框中，输入执行原因，单击**确定**。

3. 重启服务。

i. 在JindoData服务页面，选择**更多操作 > 重启**。

ii. 在**重启JindoData服务**对话框中，输入执行原因，单击**确定**。

iii. 在**确认**对话框中，单击**确定**。

步骤二：配置JindoSDK

> **注意** 此配置为客户端配置，无需重启JindoData服务。

1. 进入HDFS服务的**core-site**页签。

i. 在**集群服务**页签，单击HDFS服务区域的**配置**。

ii. 在HDFS服务的**服务配置**区域，单击**core-site**页签。

2. 修改以下配置项。

修改配置项的具体操作，请参见修改配置项。

内容	参数	描述
配置JINDO实现类	fs.AbstractFileSystem.jindo.impl	固定值为com.aliyun.jindodata.jindo.JINDO。
	fs.jindo.impl	固定值为com.aliyun.jindodata.jindo.JindoFileSystem。
	fs.xengine	固定值为jindofsx。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。

内容	参数	描述
启用缓存加速功能	fs.jindofsx.data.cache.enable	数据缓存开关： ○ false（默认值）：禁用数据缓存。 ○ true：启用数据缓存。 **说明** 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。

3. 保存配置。
   i. 单击**服务配置**区域的**保存**。
   ii. 在**确认修改配置**对话框中，输入执行原因，开启**自动更新配置**，单击**确定**。
4. 新增配置项。
   i. 单击**服务配置**区域的**新增配置项**。
   ii. 在**新增配置项**对话框中，新增以下配置项。
   新增配置项的具体操作，请参见添加配置项。

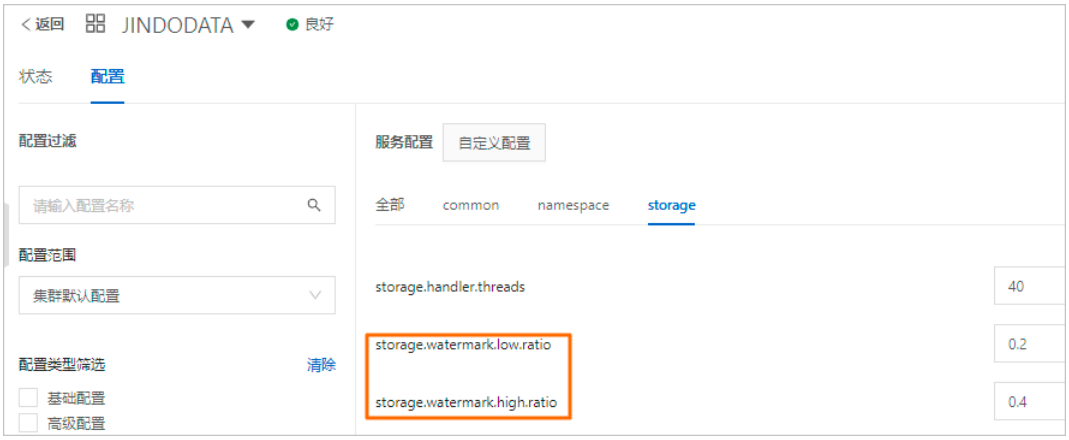
内容	参数	描述
配置Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
其他缓存加速功能	fs.jindofsx.meta.cache.enable	元数据缓存开关： ■ false（默认值）：禁用元数据缓存。 ■ true：启用元数据缓存。
	fs.jindofsx.slice.cache.enable	小文件缓存优化开关： ■ false（默认值）：禁用小文件缓存。 ■ true：启用小文件缓存。
	fs.jindofsx.ram.cache.enable	内存缓存开关： ■ false（默认值）：禁用内存缓存。 ■ true：启用内存缓存。
	fs.jindofsx.short.circuit.enable	短路读开关： ■ true（默认值）：打开短路读开关。 ■ false：关闭短路读开关。

   iii. 单击**确定**。
   iv. 在**确认修改配置**对话框中， 输入执行原因，单击**确定**。

步骤三：磁盘空间水位控制

缓存启用后，JindoFSx服务会自动管理本地缓存备份，通过水位清理本地缓存，请您根据需求配置一定的比例用于缓存。JindoFSx后端基于OSS，可以提供海量的存储，但是本地盘的容量是有限的，因此JindoFSx会自动淘汰本地较冷的数据备份。您可以通过修改**storage.watermark.high.ratio**和**storage.watermark.low.ratio**两个参数来调节本地存储的使用容量，值均为0～1的小数，表示使用磁盘空间的比例。

1. 修改磁盘水位配置。
   在JindoData服务的**服务配置**区域的**storage**页签，修改以下参数。



参数	描述
storage.watermark.low.ratio	表示使用量的下水位比例，触发清理后会自动清理冷数据，将JindoFS数据目录占用空间清理到下水位。默认值：0.2。
storage.watermark.high.ratio	表示磁盘使用量的上水位比例，每块数据盘的JindoFS数据目录占用的磁盘空间到达上水位即会触发清理。默认值：0.4。

> **说明** 修改该参数时，下水位必须小于上水位，设置合理的值即可。

2. 保存配置。
   i. 单击**服务配置**区域的**保存**。
   ii. 在**确认修改配置**对话框中，输入执行原因，开启**自动更新配置**，单击**确定**。
3. 重启服务。
   i. 选择右上角的**更多操作 > 重启**。
   ii. 在**重启JindoData服务**对话框中，输入执行原因，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

步骤四：挂载阿里云OSS

您可以执行以下命令，挂载OSS目录：

- 语法

```
jindo admin -mount <path> <realpath>
```

- 示例

```
jindo admin -mount /jindooss oss://<yourBucketName>/
```

执行以下命令，返回信息为 `jindo://emr-header-1:8101/jindooss` 。

```
hdfs dfs -ls jindo://emr-header-1:8101/
```

即访问 `jindo://emr-header-1:8101/jindooss/` 等价于访问 `oss://<yourBucketName>/` 。

步骤五：访问阿里云OSS

您通过 `jindo://` 前缀读取OSS上的数据后，在数据缓存开关打开时，会自动缓存到JindoFSx存储加速系统中，后续通过 `jindo://` 访问相同的数据就能够命中缓存。

5.4.2.2.3. 阿里云OSS-HDFS服务透明缓存加速

本文为您介绍JindoFSx支持OSS-HDFS服务（JindoFS服务）透明缓存加速的使用方式。

背景信息

OSS-HDFS服务的详细信息，请参见OSS-HDFS服务概述。

前提条件

- 已在E-MapReduce上创建EMR-3.40.0及后续版本的集群，具体操作请参见创建集群。

> 注意　本文以EMR-3.40.0版本为例介绍。

- 已开通并授权访问OSS-HDFS服务，具体操作请参见开通并授权访问OSS-HDFS服务。

操作流程

1. 步骤一：配置OSS-HDFS AccessKey
2. 步骤二：配置JindoSDK
3. 步骤三：磁盘空间水位控制

步骤一：配置OSS-HDFS AccessKey

1. 进入JindoData服务的**common**页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 单击JindoData服务区域的**配置**。
   v. 在JindoData服务的**服务配置**区域，单击**common**页签。
2. 在**新增配置项**对话框中，新增以下配置项，单击**确定**。
   新增配置项的具体操作，请参见添加配置项。
   - 全局方式配置（所有Bucket使用同一种方式）

参数	描述
jindofsx.oss.accessKeyId	OSS的AccessKey ID。
jindofsx.oss.accessKeySecret	OSS的AccessKey Secret。
jindofsx.oss.endpoint	OSS-HDFS服务的Endpoint。例如，cn-***.oss-dls.aliyuncs.com。

   - 按照Bucket配置

参数	描述
jindofsx.oss.bucket.XXX.accessKeyId	`XXX`的Bucket的AccessKey ID。
jindofsx.oss.bucket.XXX.accessKeySecret	`XXX`的Bucket的AccessKey Secret。
jindofsx.oss.bucket.XXX.endpoint	`XXX`的Bucket的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
jindofsx.oss.bucket.YYY.accessKeyId	`YYY`的Bucket的AccessKey ID。
jindofsx.oss.bucket.YYY.accessKeySecret	`YYY`的Bucket的AccessKey Secret。
jindofsx.oss.bucket.YYY.endpoint	`YYY`的Bucket的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
jindofsx.oss.bucket.YYY.user	Storage Service访问OSS-HDFS服务使用的用户名。

   > **说明**　`XXX`和`YYY`为OSS Bucket的名称。

3. 重启服务。
   i. 在JindoData服务页面，选择**更多操作 > 重启**。
   ii. 在**重启JindoData服务**对话框中，输入执行原因，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

步骤二：配置JindoSDK

> 注意　此配置为客户端配置，无需重启JindoData服务。

1. 进入HDFS服务的**core-site**页签。
   i. 在**集群服务**页签，单击HDFS服务区域的**配置**。
   ii. 在HDFS服务的**服务配置**区域，单击**core-site**页签。

2. 修改以下配置项。

   修改配置项的具体操作，请参见修改配置项。

内容	参数	描述
配置OSS实现类	fs.AbstractFileSystem.oss.impl	固定值为com.aliyun.jindodata.oss.OSS。
	fs.oss.impl	固定值为com.aliyun.jindodata.oss.JindoOssFileSystem。
	fs.xengine	固定值为jindofsx。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。 说明 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
启用缓存加速功能	fs.jindofsx.data.cache.enable	数据缓存开关： false（默认值）：禁用数据缓存。 true：启用数据缓存。 说明 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。

3. 保存配置。
   i. 单击**服务配置**区域的**保存**。
   ii. 在**确认修改配置**对话框中，输入执行原因，开启**自动更新配置**，单击**确定**。
4. 新增配置项。
   i. 单击**服务配置**区域的**新增配置项**。
   ii. 在**新增配置项**对话框中，新增以下配置项。

   新增配置项的具体操作，请参见添加配置项。

内容	参数	描述
配置Accesskey	fs.oss.accessKeyId	OSS的Bucket的AccessKey ID。
	fs.oss.accessKeySecret	OSS的Bucket的AccessKey Secret。
	fs.oss.endpoint	OSS-HDFS服务的Endpoint。格式为oss://<yourBucketName>.<yourBucketEndpoint>/<yourBucketObject>。 例如，`oss://mydlsbucket.cn-shanghai.oss-dls.aliyuncs.com/Test`。
其他缓存加速	fs.jindofsx.meta.cache.enable	元数据缓存开关： false（默认值）：禁用元数据缓存。 true：启用元数据缓存。
	fs.jindofsx.slice.cache.enable	小文件缓存优化开关： false（默认值）：禁用小文件缓存。 true：启用小文件缓存。
	fs.jindofsx.ram.cache.enable	内存缓存开关： false（默认值）：禁用内存缓存。 true：启用内存缓存。
	fs.jindofsx.short.circuit.enable	短路读开关： true（默认值）：打开短路读开关。 false：关闭短路读开关。

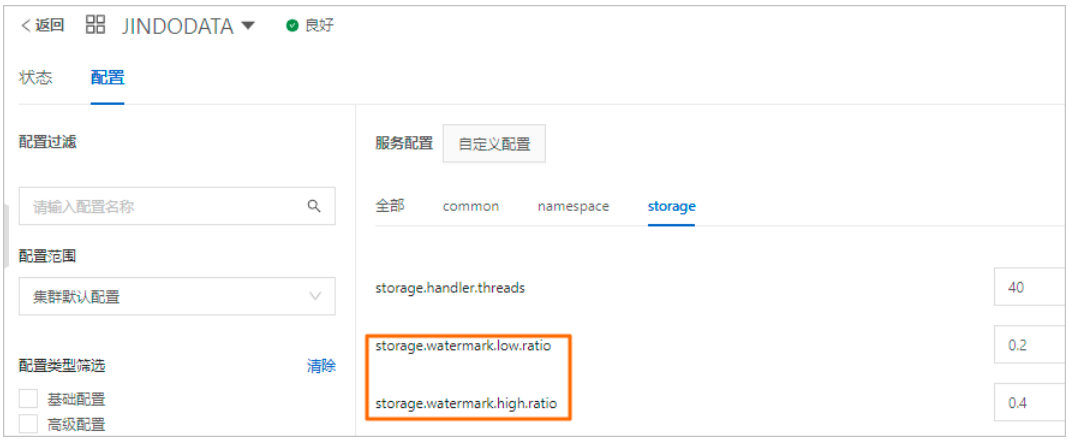
   iii. 单击**确定**。
   iv. 在**确认修改配置**对话框中， 输入执行原因，单击**确定**。

步骤三：磁盘空间水位控制

缓存启用后，JindoFSx服务会自动管理本地缓存备份，通过水位清理本地缓存，请您根据需求配置一定的比例用于缓存。JindoFSx后端基于OSS，可以提供海量的存储，但是本地盘的容量是有限的，因此JindoFSx会自动淘汰本地较冷的数据备份。您可以通过修改**storage.watermark.high.ratio**和**storage.watermark.low.ratio**两个参数来调节本地存储的使用容量，值均为0～1的小数，表示使用磁盘空间的比例。

1. 修改磁盘水位配置。

   在JindoData服务的**服务配置**区域的**storage**页签，修改以下参数。



参数	描述
storage.watermark.low.ratio	表示使用量的下水位比例，触发清理后会自动清理冷数据，将JindoFS数据目录占用空间清理到下水位。默认值：0.2。
storage.watermark.high.ratio	表示磁盘使用量的上水位比例，每块数据盘的JindoFS数据目录占用的磁盘空间到达上水位即会触发清理。默认值：0.4。

> **说明** 修改该参数时，下水位必须小于上水位，设置合理的值即可。

2. 保存配置。
   i. 单击**服务配置**区域的**保存**。
   ii. 在**确认修改配置**对话框中，输入执行原因，开启**自动更新配置**，单击**确定**。
3. 重启服务。
   i. 选择右上角的**更多操作 > 重启**。
   ii. 在**重启JindoData服务**对话框中，输入执行原因，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

5.4.2.2.4. 阿里云OSS-HDFS服务统一挂载缓存加速

本文为您介绍JindoFSx支持OSS-HDFS服务（JindoFS服务）统一挂载缓存加速的使用方式。

前提条件

- 已在E-MapReduce上创建EMR-3.40.0及后续版本的集群，具体操作请参见创建集群。

> **注意** 本文以EMR-3.40.0版本为例介绍。

- 已开通并授权访问OSS-HDFS服务，具体操作请参见开通并授权访问OSS-HDFS服务。

操作流程

1. 步骤一：配置OSS-HDFS AccessKey
2. 步骤二：配置JindoSDK
3. 步骤三：磁盘空间水位控制
4. 步骤四：挂载OSS-HDFS服务
5. 步骤五：访问阿里云OSS

步骤一：配置OSS-HDFS AccessKey

1. 进入JindoData服务的**common**页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 单击JindoData服务区域的**配置**。
   v. 在JindoData服务的**服务配置**区域，单击**common**页签。
2. 在**新增配置项**对话框中，新增以下配置项，单击**确定**。

   新增配置项的具体操作，请参见添加配置项。

   - 全局方式配置（所有Bucket使用同一种方式）

参数	描述
jindofsx.oss.accessKeyId	OSS的AccessKey ID。
jindofsx.oss.accessKeySecret	OSS的AccessKey Secret。
jindofsx.oss.endpoint	OSS-HDFS服务的Endpoint。例如，cn-***.oss-dls.aliyuncs.com。

   - 按照Bucket配置

参数	描述
jindofsx.oss.bucket.XXX.accessKeyId	`XXX`的Bucket的AccessKey ID。
jindofsx.oss.bucket.XXX.accessKeySecret	`XXX`的Bucket的AccessKey Secret。
jindofsx.oss.bucket.XXX.endpoint	`XXX`的Bucket的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
jindofsx.oss.bucket.YYY.accessKeyId	`YYY`的Bucket的AccessKey ID。
jindofsx.oss.bucket.YYY.accessKeySecret	`YYY`的Bucket的AccessKey Secret。
jindofsx.oss.bucket.YYY.endpoint	`YYY`的Bucket的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
jindofsx.oss.bucket.YYY.user	Storage Service访问OSS-HDFS服务使用的用户名。

   > **说明** `XXX`和`YYY`为OSS-HDFS服务Bucket的名称。

3. 重启服务。
   i. 在JindoData服务页面，选择**更多操作 > 重启**。
   ii. 在**重启JindoData服务**对话框中，输入执行原因，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

步骤二：配置JindoSDK

> **注意** 此配置为客户端配置，无需重启JindoData服务。

1. 进入HDFS服务的**core-site**页签。
   i. 在**集群服务**页签，单击HDFS服务区域的**配置**。
   ii. 在HDFS服务的**服务配置**区域，单击**core-site**页签。
2. 新增和修改配置项。

   新增配置项的具体操作，请参见添加配置项。修改配置项的具体操作，请参见修改配置项。

内容	参数	描述
配置统一名字空间使用的实现类	fs.AbstractFileSystem.jindo.impl	固定值为com.aliyun.jindodata.jindo.JINDO。
	fs.jindo.impl	固定值为com.aliyun.jindodata.jindo.JindoFileSystem。
	fs.xengine	固定值为jindofsx。

内容	参数	描述
配置OSS-HDFS服务的AccessKey	fs.oss.accessKeyId	OSS-HDFS服务Bucket对应的AccessKey ID。
	fs.oss.accessKeySecret	OSS-HDFS服务Bucket对应的AccessKey Secret。
配置OSS-HDFS服务的Endpoint	访问OSS-HDFS服务时需要配置Endpoint。 ○ （推荐）方式一：在访问路径中指定Endpoint。 推荐访问路径格式为 `oss://<Bucket>.<Endpoint>/<Object>` ，例如 `oss://examplebucket.cn-shanghai.oss-dls.aliyuncs.com/exampleobject.txt` 。 该方式在访问路径中包含Endpoint，JindoSDK会根据路径中的Endpoint访问对应接口。 ○ 方式二：配置Bucket级别的Endpoint。 如果使用 `oss://<Bucket>/<Object>` 格式的访问路径，即访问路径中未设置Endpoint，JindoSDK会在配置中查找Bucket级别的Endpoint。 您可以在HDFS服务的**core-site**页签添加自定义配置，从而指向JindoFS服务的Endpoint。设置**Key**为fs.oss.bucket.XXX.endpoint，**Value**为cn-***.oss-dls.aliyuncs.com。 ○ 方式三：配置全局默认Endpoint。 如果使用 `oss://<Bucket>/<Object>` 格式的访问路径，而且访问路径中未设置Bucket级别的Endpoint，则会用全局Endpoint访问。 您可以在HDFS服务的**core-site**页签添加自定义配置，从而指向JindoFS服务的Endpoint。设置**Key**为fs.oss.endpoint，**Value**为cn-***.oss-dls.aliyuncs.com。 **说明** `XXX` 为OSS-HDFS服务Bucket的名称。 配置完成后，JindoSDK会根据访问路径中的Endpoint访问对应的OSS-HDFS服务接口。	
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
启用缓存加速功能	fs.jindofsx.data.cache.enable	数据缓存开关： ○ false（默认值）：禁用数据缓存。 ○ true：启用数据缓存。 **说明** 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。
	fs.jindofsx.meta.cache.enable	元数据缓存开关： ○ false（默认值）：禁用元数据缓存。 ○ true：启用元数据缓存。
	fs.jindofsx.slice.cache.enable	小文件缓存优化开关： ○ false（默认值）：禁用小文件缓存。 ○ true：启用小文件缓存。
	fs.jindofsx.ram.cache.enable	内存缓存开关： ○ false（默认值）：禁用内存缓存。 ○ true：启用内存缓存。
	fs.jindofsx.short.circuit.enable	短路读开关： ○ true（默认值）：打开短路读开关。 ○ false：关闭短路读开关。

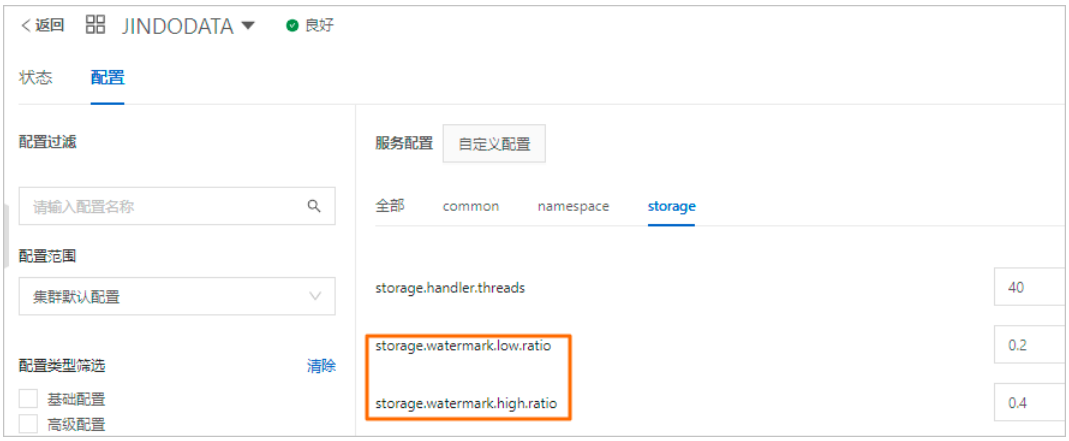
说明 完成以上配置，作业读取JindoFS上的数据后，会自动缓存到JindoFSx存储加速系统中，作业访问JindoFS的方式无需做任何修改，后续访问相同的数据就能够命中缓存。

步骤三：磁盘空间水位控制

缓存启用后，JindoFSx服务会自动管理本地缓存备份，通过水位清理本地缓存，请您根据需求配置一定的比例用于缓存。JindoFSx后端基于OSS，可以提供海量的存储，但是本地盘的容量是有限的，因此JindoFSx会自动淘汰本地较冷的数据备份。您可以通过修改**storage.watermark.high.ratio**和**storage.watermark.low.ratio**两个参数来调节本地存储的使用容量，值均为0～1的小数，表示使用磁盘空间的比例。

1. 修改磁盘水位配置。

   在JindoData服务的**服务配置**区域的**storage**页签，修改以下参数。



参数	描述
storage.watermark.low.ratio	表示使用量的下水位比例，触发清理后会自动清理冷数据，将JindoFS数据目录占用空间清理到下水位。默认值：0.2。
storage.watermark.high.ratio	表示磁盘使用量的上水位比例，每块数据盘的JindoFS数据目录占用的磁盘空间到达上水位即会触发清理。默认值：0.4。

> **说明** 修改该参数时，下水位必须小于上水位，设置合理的值即可。

2. 保存配置。
   i. 单击**服务配置**区域的**保存**。
   ii. 在**确认修改配置**对话框中，输入执行原因，开启**自动更新配置**，单击**确定**。
3. 重启服务。
   i. 选择右上角的**更多操作 > 重启**。
   ii. 在**重启JindoData服务**对话框中，输入执行原因，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

步骤四：挂载OSS-HDFS服务

挂载OSS-HDFS服务目录语法如下。

```
jindo admin -mount <path> <realpath>
```

例如，执行以下命令挂载OSS-HDFS服务目录。

```
jindo admin -mount /jindodls oss://<yourBucketName>.<yourBucketEndpoint>/
```

> **说明** 执行上述命令后，则 `/jindodls` 目录真正挂载的文件路径是 `oss://<yourBucketName>.<yourBucketEndpoint>/` 。

执行以下命令，返回信息为 `jindo://emr-header-1:8101/jindodls` 。

```
hdfs dfs -ls jindo://emr-header-1:8101/
```

即访问 `jindo://emr-header-1:8101/jindodls/` 等价于访问 `oss://<yourBucketName>/<yourBucketObject>` 。

步骤五：访问阿里云OSS

您通过 `jindo://` 前缀读取OSS-HDFS服务上的数据后，在数据缓存开关打开时，会自动缓存到JindoFSx存储加速系统中，后续通过 `jindo://` 访问相同的数据就能够命中缓存。

5.4.2.2.5. Apache HDFS透明缓存加速

Apache HDFS透明缓存加速可以利用计算集群的闲置存储资源对远端HDFS集群进行数据缓存，避免了计算集群或服务占用核心集群过多带宽。适用于在HDFS集群和计算集群分离，HDFS集群访问性能不及预期时，您可以通过在计算集群或靠近计算集群的地方缓存数据来进行加速。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> **说明** 本文以EMR-3.40.0版本为例介绍。

操作流程

1. 步骤一：配置服务端
2. 步骤二：配置JindoSDK
3. 步骤三：访问HDFS

步骤一：配置服务端

1. 进入JindoData服务的**common**页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 单击JindoData服务区域的**配置**。
   v. 在JindoData服务的**服务配置**区域，单击**common**页签。
2. 新增配置。
   i. 单击**新增配置项**。
   ii. 在**新增配置项**对话框中，新增以下配置项。

   新增配置项的具体操作，请参见添加配置项。

集群类型	参数	描述
普通集群	jindofsx.hdfs.user	Storage Service访问HDFS使用的用户名。
HA集群	jindofsx.hdfs.XXX.dfs.ha.namenodes	填写格式为 `nn1,nn2` 。
	jindofsx.hdfs.XXX.dfs.namenode.rpc-address.nn1	填写格式为 `nn1-host1:nn1-rpc-port` 。
	jindofsx.hdfs.XXX.dfs.namenode.rpc-address.nn2	填写格式为 `nn2-host1:nn2-rpc-port` 。

   > **说明** 根据您集群的类型，新增相应的配置项。 `XXX` 为集群中配置的dfs.nameservices参数值。

   iii. 单击**确定**。
   iv. 在**确认修改配置**对话框中，输入执行原因，单击**确定**。
3. 重启服务。
   i. 在JindoData服务页面，选择**更多操作 > 重启**。
   ii. 在**重启JindoData服务**对话框中，输入执行原因，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

步骤二：配置JindoSDK

> **注意** 此配置为客户端配置，无需重启JindoData服务。

1. 进入HDFS服务的**core-site**页签。
   i. 在**集群服务**页签，单击HDFS服务区域的**配置**。
   ii. 在HDFS服务的**服务配置**区域，单击**core-site**页签。
2. 新增和修改配置项。

新增配置项的具体操作，请参见添加配置项。修改配置项的具体操作，请参见修改配置项。

内容	参数	描述
配置统一名字空间使用的实现类	fs.hdfs.impl	固定值为com.aliyun.jindodata.hdfs.JindoHdfsFileSystem。
	fs.AbstractFileSystem.hdfs.impl	固定值为com.aliyun.jindodata.hdfs.HDFS。
	fs.xengine	固定值为jindofsx。
（可选）配置HA Namenodes **说明** 如果为HA集群，则需要配置该类参数。	fs.jindofsx.hdfs.XXX.dfs.ha.namenodes	格式为 `nn1,nn2` 。
	fs.jindofsx.hdfs.XXX.dfs.namenode.rpc-address.nn1	格式为 `nn1-host1:nn1-rpc-port` 。
	fs.jindofsx.hdfs.XXX.dfs.namenode.rpc-address.nn2	格式为 `nn2-host1:nn2-rpc-port` 。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
开启缓存加速 **说明** 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为禁用，即所有OSS都可直接访问OSS上的数据。	fs.jindofsx.data.cache.enable	数据缓存开关： ○ false（默认值）：禁用数据缓存。 ○ true：启用数据缓存。
	fs.jindofsx.meta.cache.enable	元数据缓存开关： ○ false（默认值）：禁用元数据缓存。 ○ true：启用元数据缓存。
	fs.jindofsx.slice.cache.enable	小文件缓存优化开关： ○ false（默认值）：禁用小文件缓存。 ○ true：启用小文件缓存。
	fs.jindofsx.ram.cache.enable	内存缓存开关： ○ false（默认值）：禁用内存缓存。 ○ true：启用内存缓存。
	fs.jindofsx.short.circuit.enable	短路读开关： ○ true（默认值）：打开。 ○ false：关闭。

步骤三：访问HDFS

您通过 `hdfs://` 前缀读取HDFS上的数据后，在数据缓存开关打开时，会自动缓存到JindoFSx存储加速系统中，后续通过 `hdfs://` 访问相同的数据就能够命中缓存。

5.4.2.2.6. Apache HDFS统一挂载缓存加速

Apache HDFS统一挂载缓存加速可以利用计算集群的闲置存储资源进行数据缓存来加速计算服务，避免了计算集群或服务占用核心集群过多带宽。在计算集群与待访问的HDFS集群网络带宽有限的情况下，可以通过缓存加速提升计算集群访问HDFS集群的速度。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

说明 本文以EMR-3.40.0版本为例介绍。

操作流程

1. 步骤一：配置服务端
2. 步骤二：配置JindoSDK

3. 步骤三：挂载HDFS
4. 步骤四：访问HDFS

步骤一：配置服务端

1. 进入JindoData服务的**common**页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 单击JindoData服务区域的**配置**。
   v. 在JindoData服务的**服务配置**区域，单击**common**页签。
2. 新增配置。
   i. 单击**新增配置项**。
   ii. 在**新增配置项**对话框中，新增以下配置项。
   新增配置项的具体操作，请参见添加配置项。

集群类型	参数	描述
普通集群	jindofsx.hdfs.user	Storage Service访问HDFS使用的用户名。
HA集群	jindofsx.hdfs.XXX.dfs.ha.namenodes	填写格式为 `nn1,nn2` 。
	jindofsx.hdfs.XXX.dfs.namenode.rpc-address.nn1	填写格式为 `nn1-host1:nn1-rpc-port` 。
	jindofsx.hdfs.XXX.dfs.namenode.rpc-address.nn2	填写格式为 `nn2-host1:nn2-rpc-port` 。

> **说明** 根据您集群的类型，新增相应的配置项。 `XXX` 为集群中配置的dfs.nameservices参数值。

   iii. 单击**确定**。
   iv. 在**确认修改配置**对话框中，输入执行原因，单击**确定**。
3. 重启服务。
   i. 在JindoData服务页面，选择**更多操作 > 重启**。
   ii. 在**重启JindoData服务**对话框中，输入执行原因，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

步骤二：配置JindoSDK

> **注意** 此配置为客户端配置，无需重启JindoData服务。

1. 进入HDFS服务的**core-site**页签。
   i. 在**集群服务**页签，单击HDFS服务区域的**配置**。
   ii. 在HDFS服务的**服务配置**区域，单击**core-site**页签。
2. 新增和修改配置项。
   新增配置项的具体操作，请参见添加配置项。修改配置项的具体操作，请参见修改配置项。

内容	参数	描述
配置统一名字空间使用的实现类	fs.AbstractFileSystem.jindo.impl	固定值为com.aliyun.jindodata.jindo.JINDO。
	fs.jindo.impl	固定值为com.aliyun.jindodata.jindo.JindoFileSystem。
	fs.xengine	固定值为jindofsx。
（可选）配置HA Namenodes > **说明** 如果为HA集群，则需要配置该类参数。	fs.jindofsx.hdfs.XXX.dfs.ha.namenodes	格式为 `nn1,nn2` 。
	fs.jindofsx.hdfs.XXX.dfs.namenode.rpc-address.nn1	格式为 `nn1-host1:nn1-rpc-port` 。
	fs.jindofsx.hdfs.XXX.dfs.namenode.rpc-address.nn2	格式为 `nn2-host1:nn2-rpc-port` 。

内容	参数	描述
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
开启缓存加速 **说明** 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为禁用，即所有OSS都可直接访问OSS上的数据。	fs.jindofsx.data.cache.enable	数据缓存开关： false（默认值）：禁用数据缓存。 true：启用数据缓存。
	fs.jindofsx.meta.cache.enable	元数据缓存开关： false（默认值）：禁用元数据缓存。 true：启用元数据缓存。
	fs.jindofsx.slice.cache.enable	小文件缓存优化开关： false（默认值）：禁用小文件缓存。 true：启用小文件缓存。
	fs.jindofsx.ram.cache.enable	内存缓存开关： false（默认值）：禁用内存缓存。 true：启用内存缓存。
	fs.jindofsx.short.circuit.enable	短路读开关： true（默认值）：打开。 false：关闭。

步骤三：挂载HDFS

- 语法

```
jindo admin -mount <path> <realpath>
```

- 示例

  执行以下命令，挂载HDFS目录。

```
jindo admin -mount /jindohdfs hdfs://emr-header-1:9000/
```

  执行如上命令后，则 `jindo:///jindohdfs` 目录下真正挂载的文件路径是 `hdfs://emr-header-1:9000/` 。

步骤四：访问HDFS

您通过 `jindo://` 前缀读取HDFS上的数据后，在数据缓存开关打开时会自动缓存到JindoFSx存储加速系统中，后续通过 `jindo://` 访问相同的数据就能够命中缓存。

5.4.2.2.7. 阿里云文件存储NAS统一挂载缓存加速

阿里云文件存储NAS（Apsara File Storage NAS）是一个可大规模共享访问，弹性扩展的高性能云原生分布式文件系统。支持智能冷热数据分层，有效降低数据存储成本。本文主要为您介绍JindoFSx支持阿里云文件存储NAS统一挂载缓存加速的使用方式。

背景信息

文件存储NAS的详细信息，请参见什么是文件存储NAS。

前提条件

- 已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

  说明 本文以EMR-3.40.0版本为例介绍。

- 集群已挂载NFS文件系统，详情请参见Linux系统挂载NFS文件系统。

操作流程

1. 步骤一：配置JindoSDK

2. 步骤二：挂载NAS文件系统目录
3. 步骤三：访问NAS文件系统目录

步骤一：配置JindoSDK

1. 进入HDFS服务的**core-site**页签。
   i. 在**集群服务**页签，单击HDFS服务区域的**配置**。
   ii. 在HDFS服务的**服务配置**区域，单击**core-site**页签。
2. 新增和修改配置项。

   新增配置项的具体操作，请参见添加配置项。修改配置项的具体操作，请参见修改配置项。

内容	参数	描述
配置统一名字空间使用的实现类	fs.jindo.impl	固定值为com.aliyun.jindodata.jindo.JindoFileSystem。
	fs.AbstractFileSystem.jindo.impl	固定值为com.aliyun.jindodata.jindo.JINDO。
	fs.xengine	固定值为jindofsx。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
启用缓存加速功能	fs.jindofsx.data.cache.enable	数据缓存开关： false（默认值）：禁用数据缓存。 true：启用数据缓存。 **说明** 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。
	fs.jindofsx.meta.cache.enable	元数据缓存开关： false（默认值）：禁用元数据缓存。 true：启用元数据缓存。
	fs.jindofsx.slice.cache.enable	小文件缓存优化开关： false（默认值）：禁用小文件缓存。 true：启用小文件缓存。
	fs.jindofsx.ram.cache.enable	内存缓存开关： false（默认值）：禁用内存缓存。 true：启用内存缓存。
	fs.jindofsx.short.circuit.enable	短路读开关： true（默认值）：打开短路读开关。 false：关闭短路读开关。

步骤二：挂载NAS文件系统目录

- 语法

```
jindo admin -mount <path> <realpath>
```

- 示例

  执行以下命令，挂载NAS文件系统目录。

```
jindo admin -mount /jindonas /mnt/nas
```

假设NAS文件系统挂载在服务器上的本地路径为 `/mnt/nas` 。执行如上命令后，则 `/jindonas` 目录下真正挂载的文件路径是 `/mnt/nas` 。

执行 `hdfs dfs -ls jindo://emr-header-1:8101/` 命令，返回如下信息，即访问 `jindo://emr-header-1:8101` 等价于访问 `/mnt/nas/` 。

```
----------   1          0 1970-01-01 08:00 jindo://emr-header-1:8101/jindonas
```

步骤三：访问NAS文件系统目录

您通过 `jindo://` 前缀读取NAS上的数据后，在数据缓存开关打开时，会自动缓存到JindoFSx存储加速系统中，后续通过 `jindo://` 访问相同的数据就能够命中缓存。

5.4.2.2.8. P2P分布式下载缓存

JindoFSx客户端P2P可以被视作一种本地缓存（LocalCache）。与原有的LocalCache相比，P2P缓存中的本地数据块会优先从其他持有该数据的客户端拉取，只有无法向其他客户端请求时，才会从STS或远端读取。本文为您介绍P2P分布式下载缓存的使用方法。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> **说明** 本文以EMR-3.40.0版本为例介绍。

操作流程

1. 进入JindoData服务的**common**页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 单击JindoData服务区域的**配置**。
   v. 在JindoData服务的**服务配置**区域，单击**common**页签。
2. 新增配置。
   i. 单击**自定义配置**。
   ii. 在**新增配置项**对话框中，新增以下配置项。

   新增配置项的具体操作，请参见添加配置项。

配置项	参数	描述
服务端配置	jindofsx.p2p.tracker.thread.number	TrackerService的处理线程数。 如果要开启P2P功能，则该参数值必须设置大于1。如果小于等于1，则不会创建TrackerService，也不会开启P2P功能。
	jindofsx.p2p.file.prefix	使用P2P下载的前缀列表。当包含多个文件路径时，使用半角逗号（,）隔开，文件路径只有匹配到其中任一个前缀，才会以P2P方式下载。在应用层使用统一挂载路径进行下载时，此处仍应配置为真实的对象路径。 例如，`oss://bucket1/data-dir1/,oss://bucket2/data-dir2/`。
客户端配置	fs.jindofsx.p2p.cache.capacity.limit	P2P下载最大占用的缓存大小，单位为字节，默认为5 GB，最小值为1 GB。 例如，取值为5 * 1024 * 1024 * 1024。
	fs.jindofsx.p2p.download.parallelism.per.file	P2P下载单个文件使用的并发数。 例如，取值为5。
	fs.jindofsx.p2p.download.thread.pool.size	P2P下载使用的线程池总大小。 例如，取值为5。

   iii. 单击**确定**。
   iv. 在**确认修改配置**对话框中，输入执行原因，单击**确定**。
3. 重启服务。
   i. 在JindoData服务页面，选择**更多操作 > 重启**。
   ii. 在**重启JindoData服务**对话框中，输入执行原因，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

5.4.2.2.9. JindoShell CLI支持JindoFSx使用说明

JindoShell CLI支持操作JindoFSx数据缓存、元数据缓存和统一命名空间等命令。

背景信息

本文为您介绍以下内容：

- 数据缓存命令
- 元数据缓存命令
- 清理缓存命令
- 统一命名空间命令
- 其他命令

数据缓存命令

数据缓存命令可以备份对应路径的数据至本集群的磁盘，以便于后续可以读取本地数据，无需读取OSS等后端上的数据。

```
jindo fsx -load -data <options> <path>
```

参数	描述
`<options>`	各种可选参数： • `-s` ：表示缓存过程同步执行。即缓存完成前命令不退出，日志直接打印在控制台上。推荐开启。 • `-replica` ：缓存副本数量，默认缓存1个副本。 • `-R` ：递归缓存文件，当 `<path>` 是文件夹时需开启。 • `-cachelist` ：接收本地文件，文件内容为cache列表。
`<path>`	数据缓存路径。

推荐使用以下组合命令。

```
jindo fsx -load -data -s -R <path>
```

元数据缓存命令

元数据缓存命令可以备份远端文件的元数据信息，从而后续无需从OSS等后端读取文件元数据信息。

```
jindo fsx -load -meta <options> <path>
```

参数	描述
`<options>`	各种可选参数： • `-s` ：表示缓存过程同步执行，即缓存完成前命令不退出，日志直接打印在控制台上。推荐开启。 • `-R` ：递归缓存文件，当 `<path>` 是文件夹时需开启。
`<path>`	元数据缓存路径。

推荐使用以下组合命令。

```
jindo fsx -load -meta -s -R <path>
```

数据缓存和元数据缓存可以组合使用，当需要同时进行二者缓存时，可以搭配可选参数使用。推荐使用以下组合命令。

```
jindo fsx -load -meta -data -s -R <path>
```

清理缓存命令

uncache命令可以删除本地集群中的本地备份，只存储数据在OSS标准存储上，以便于后续读取OSS上的数据。

```
jindo fsx -uncache <path>
```

统一命名空间命令

- 添加一个挂载点

  ```
  jindo fsxadmin -mount <path> <realpath>
  ```

- 移除一个挂载点

```
jindo fsxadmin -unmount <path>
```

其他命令

执行以下命令，输出当前缓存系统的信息，例如缓存大小，缓存容量等。

```
jindo fsx -report
```

输出信息如下。

```
Namespace Address: 127.0.0.1:8101
Rpc Port: 8101
Started: Mon Jan 10 15:23:51 2022
Version: 4.1.0
Live Nodes: 2
Decommission Nodes: 0
Total Disk Capacity: 438.17GB
Used Disk Capacity: 5120.00MB
Total MEM Capacity: 4096.00MB
Used MEM Capacity: 0B
```

5.4.2.2.10. S3、COS和OBS多云存储

JindoFSx存储加速系统提供了透明缓存和统一挂载命名空间的使用方式，支持多种Scheme的云存储。文件都以对象的形式存储在云存储上，每个文件根据实际访问情况会在本地进行缓存，提升访问云存储的效率，同时兼容了原有云存储文件形式，数据访问上能够与其他云存储客户端完全兼容，作业访问云存储的方式无需做任何修改。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> **说明** 本文以EMR-3.40.0版本为例介绍。

步骤一：配置多云的AccessKey

1. 进入JindoData服务的**common**页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 单击JindoData服务区域的**配置**。
   v. 在JindoData服务的**服务配置**区域，单击**common**页签。
2. 新增配置。
   i. 单击**新增配置项**。

ii. 在**新增配置项**对话框中，新增以下配置项。

新增配置项的具体操作，请参见添加配置项。

参数	描述
jindofsx.s3.bucket.XXX.accessKeyId	S3 Bucket的AccessKey ID。
jindofsx.s3.bucket.XXX.accessKeySecret	S3 Bucket的AccessKey Secret。
jindofsx.s3.bucket.XXX.endpoint	S3 Bucket的Endpoint。例如，s3.***.amazonaws.com。
jindofsx.cos.bucket.YYY.accessKeyId	COS Bucket的AccessKey ID。
jindofsx.cos.bucket.YYY.accessKeySecret	COS Bucket的AccessKey Secret。
jindofsx.cos.bucket.YYY.endpoint	COS Bucket的Endpoint。例如，cos.***.myqcloud.com。
jindofsx.obs.bucket.ZZZ.accessKeyId	OBS Bucket的AccessKey ID。
jindofsx.obs.bucket.ZZZ.accessKeySecret	OBS Bucket的AccessKey Secret。
jindofsx.obs.bucket.ZZZ.endpoint	OBS Bucket的Endpoint。例如，obs.***.***icloud.com。

> **说明** `XXX`、`YYY`和`ZZZ`分别为S3 Bucket、COS Bucket和OBS Bucket的名称。

iii. 单击**确定**。

iv. 在**确认修改配置**对话框中，输入执行原因，单击**确定**。

3. 重启服务。

i. 在JindoData服务页面，选择**更多操作 > 重启**。

ii. 在**重启JindoData服务**对话框中，输入执行原因，单击**确定**。

iii. 在**确认**对话框中，单击**确定**。

步骤二：配置JindoSDK

1. 进入HDFS服务的**core-site**页签。

i. 在**集群服务**页签，单击HDFS服务区域的**配置**。

ii. 在HDFS服务的**服务配置**区域，单击**core-site**页签。

2. 修改以下配置项。

修改配置项的具体操作，请参见修改配置项。

内容	参数	描述
配置多云实现类	fs.xengine	固定值为jindofsx。
	fs.AbstractFileSystem.s3.impl	固定值为com.aliyun.jindodata.s3.S3。
	fs.s3.impl	固定值为com.aliyun.jindodata.s3.JindoS3FileSystem。
	fs.AbstractFileSystem.cos.impl	固定值为com.aliyun.jindodata.cos.Cos。
	fs.cos.impl	固定值为com.aliyun.jindodata.cos.JindoCosFileSystem。
	fs.AbstractFileSystem.obs.impl	固定值为com.aliyun.jindodata.obs.OBS。
	fs.obs.impl	固定值为com.aliyun.jindodata.obs.JindoObsFileSystem。
配置AccessKey	fs.s3.accessKeyId	S3 Bucket的AccessKey ID。
	fs.s3.accessKeySecret	S3 Bucket的AccessKey Secret。
	fs.s3.endpoint	S3 Bucket的Endpoint。例如，s3.***.amazonaws.com。
	fs.cos.accessKeyId	COS Bucket的AccessKey ID。
	fs.cos.accessKeySecret	COS Bucket的AccessKey Secret。
	fs.cos.endpoint	COS Bucket的Endpoint。例如，cos.***.myqcloud.com。

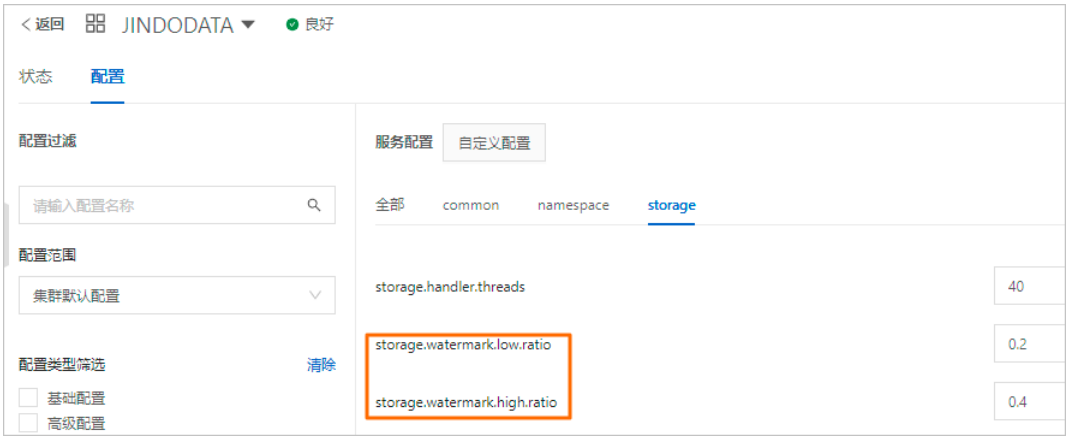
内容	参数	描述
	fs.obs.accessKeyId	OBS Bucket的AccessKey ID。
	fs.obs.accessKeySecret	OBS Bucket的AccessKey Secret。
	fs.obs.endpoint	OBS Bucket的Endpoint。例如，obs.***.***cloud.com。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 说明 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
缓存加速功能（可选）	fs.jindofsx.data.cache.enable	数据缓存开关： false（默认值）：禁用数据缓存。 true：启用数据缓存。 说明 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。
	fs.jindofsx.meta.cache.enable	元数据缓存开关： false（默认值）：禁用元数据缓存。 true：启用元数据缓存。
	fs.jindofsx.slice.cache.enable	小文件缓存优化开关： false（默认值）：禁用小文件缓存。 true：启用小文件缓存。
	fs.jindofsx.ram.cache.enable	内存缓存开关： false（默认值）：禁用内存缓存。 true：启用内存缓存。
	fs.jindofsx.short.circuit.enable	短路读开关： true（默认值）：打开。 false：关闭。

步骤三：磁盘空间水位控制

缓存启用后，JindoFSx服务会自动管理本地缓存备份，通过水位清理本地缓存，请您根据需求配置一定的比例用于缓存。JindoFSx后端基于OSS，可以提供海量的存储，但是本地盘的容量是有限的，因此JindoFSx会自动淘汰本地较冷的数据备份。您可以通过修改**storage.watermark.high.ratio**和**storage.watermark.low.ratio**两个参数来调节本地存储的使用容量，值均为0～1的小数，表示使用磁盘空间的比例。

1. 修改磁盘水位配置。

   在JindoData服务的**服务配置**区域的**storage**页签，修改以下参数。



参数	描述
storage.watermark.low.ratio	表示使用量的下水位比例，触发清理后会自动清理冷数据，将JindoFS数据目录占用空间清理到下水位。默认值：0.2。
storage.watermark.high.ratio	表示磁盘使用量的上水位比例，每块数据盘的JindoFS数据目录占用的磁盘空间到达上水位即会触发清理。默认值：0.4。

> **说明** 修改该参数时，下水位必须小于上水位，设置合理的值即可。

2. 保存配置。
   i. 单击**服务配置**区域的**保存**。
   ii. 在**确认修改配置**对话框中，输入执行原因，开启**自动更新配置**，单击**确定**。
3. 重启服务。
   i. 选择右上角的**更多操作 > 重启**。
   ii. 在**重启JindoData服务**对话框中，输入执行原因，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

（可选）

步骤四：配置统一命名空间挂载多云存储

> **说明** 如果您配置了挂载，则之后可以使用 `jindo://` 访问云存储。如果不配置挂载，则可以直接使用 `s3://` 、 `cos://` 或 `obs://` 访问云存储。

1. 配置统一命名空间使用的实现类。
   在HDFS服务的**core-site**页签，配置以下参数。

参数	描述
fs.AbstractFileSystem.jindo.impl	固定值为com.aliyun.jindodata.jindo.JINDO。
fs.jindo.impl	固定值为com.aliyun.jindodata.jindo.JindoFileSystem。
fs.xengine	固定值为jindofsx。

2. 挂载S3目录。
   本文以挂载AWS S3目录为例，COS和OBS使用方法类似。
   挂载语法如下。

```
jindo admin -mount <path> <realpath>
```

   例如，执行以下命令后，则*/jindos3*目录下真正挂载的文件路径是*s3://<Bucket>/*。

```
jindo admin -mount /jindos3 s3://<Bucket>/
```

步骤五：访问AWS S3

配置挂载后，作业在数据缓存开关打开时通过 `jindo://` 前缀读取AWS S3上的数据后，会自动缓存到JindoFSx存储加速系统中，后续通过 `jindo://` 访问相同的数据就能够命中缓存。

5.4.2.3. 大数据生态

5.4.2.3.1. Hadoop访问阿里云OSS+JindoFSx透明加速

JindoSDK为JindoFSx存储加速系统提供了Apache Hadoop支持。本文为您介绍Hadoop如何使用JindoSDK处理阿里云OSS上的数据。

背景信息

JindoSDK包含一些高级调优参数，配置方式以及配置项请参见JindoSDK高级参数配置。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> **说明** 本文以EMR-3.40.0版本为例介绍。

操作流程

1. 步骤一：配置AccessKey
2. 步骤二：配置JindoSDK
3. 步骤三：使用JindoSDK访问OSS

步骤一：配置AccessKey

1. 进入JindoData服务的**common**页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 单击JindoData服务区域的**配置**。
   v. 在JindoData服务的**服务配置**区域，单击**common**页签。
2. 新增配置。
   i. 单击**新增配置项**。
   ii. 在**新增配置项**对话框中，新增以下配置项。

   新增配置项的具体操作，请参见添加配置项。

   - 全局方式配置（所有Bucket使用同一种方式）

参数	描述
jindofsx.oss.accessKeyId	OSS的AccessKey ID。
jindofsx.oss.accessKeySecret	OSS的AccessKey Secret。
jindofsx.oss.endpoint	OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。

   - 按照Bucket配置

参数	描述
jindofsx.oss.bucket.XXX.accessKeyId	`XXX`的Bucket的AccessKey ID。
jindofsx.oss.bucket.XXX.accessKeySecret	`XXX`的Bucket的AccessKey Secret。
jindofsx.oss.bucket.XXX.endpoint	`XXX`的Bucket的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
jindofsx.oss.bucket.YYY.accessKeyId	`YYY`的Bucket的AccessKey ID。
jindofsx.oss.bucket.YYY.accessKeySecret	`YYY`的Bucket的AccessKey Secret。
jindofsx.oss.bucket.YYY.endpoint	`YYY`的Bucket的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。

   > **说明** `XXX`和`YYY`为OSS Bucket的名称。

   iii. 单击**确定**。
   iv. 在**确认修改配置**对话框中，输入执行原因，单击**确定**。
3. 重启服务。
   i. 在JindoData服务页面，选择**更多操作 > 重启**。
   ii. 在**重启JindoData服务**对话框中，输入执行原因，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

步骤二：配置JindoSDK

1. 进入HDFS服务的**core-site**页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。

iv. 在**集群服务**页签，单击HDFS服务区域的**配置**。

v. 在HDFS服务的**服务配置**区域，单击**core-site**页签。

2. 新增和修改配置项。

新增配置项的具体操作，请参见添加配置项。修改配置项的具体操作，请参见修改配置项。

内容	参数	描述
配置JindoSDK OSS实现类	fs.AbstractFileSystem.oss.impl	固定值为com.aliyun.jindodata.oss.OSS。
	fs.oss.impl	固定值为com.aliyun.jindodata.oss.JindoOssFileSystem。
	fs.xengine	固定值为jindofsx。
配置Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
开启缓存	fs.jindofsx.data.cache.enable	数据缓存开关： false（默认值）：禁用数据缓存。 true：启用数据缓存。 **说明** 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。

步骤三：使用JindoSDK访问OSS

您可以使用Hadoop Shell访问OSS，常用命令如下：

- put操作

```
hadoop fs -put <path> oss://<BucketName>/
```

- ls操作

```
hadoop fs -ls oss://<BucketName>/
```

- mkdir操作

```
hadoop fs -mkdir oss://<BucketName>/<path>
```

- rm操作

```
hadoop fs -rm oss://<BucketName>/<path>
```

5.4.2.3.2. Hadoop访问阿里云OSS-HDFS服务+JindoFSx透明加速

JindoSDK为JindoFSx存储加速系统提供了Apache Hadoop支持。本文为您介绍Hadoop如何使用JindoSDK查询OSS-HDFS服务（JindoFS服务）中的数据。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

说明 本文以EMR-3.40.0版本为例介绍。

操作流程

1. 步骤一：配置AccessKey
2. 步骤二：配置JindoSDK
3. 步骤三：使用JindoSDK访问OSS-HDFS服务

步骤一：配置AccessKey

1. 进入JindoData服务的**common**页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 单击JindoData服务区域的**配置**。
   v. 在JindoData服务的**服务配置**区域，单击**common**页签。
2. 新增配置。
   i. 单击**新增配置项**。
   ii. 在**新增配置项**对话框中，新增以下配置项。
   新增配置项的具体操作，请参见添加配置项。
   - 全局方式配置（所有Bucket使用同一种方式）

参数	描述
jindofsx.oss.accessKeyId	OSS的AccessKey ID。
jindofsx.oss.accessKeySecret	OSS的AccessKey Secret。
jindofsx.oss.endpoint	OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。

   - 按照Bucket配置

参数	描述
jindofsx.oss.bucket.XXX.accessKeyId	`XXX`的Bucket的AccessKey ID。
jindofsx.oss.bucket.XXX.accessKeySecret	`XXX`的Bucket的AccessKey Secret。
jindofsx.oss.bucket.XXX.endpoint	`XXX`的Bucket的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
jindofsx.oss.bucket.YYY.accessKeyId	`YYY`的Bucket的AccessKey ID。
jindofsx.oss.bucket.YYY.accessKeySecret	`YYY`的Bucket的AccessKey Secret。
jindofsx.oss.bucket.YYY.endpoint	`YYY`的Bucket的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。

   > **说明** `XXX`和`YYY`为OSS Bucket的名称。

   iii. 单击**确定**。
   iv. 在**确认修改配置**对话框中，输入执行原因，单击**确定**。
3. 重启服务。
   i. 在JindoData服务页面，选择**更多操作 > 重启**。
   ii. 在**重启JindoData服务**对话框中，输入执行原因，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

步骤二：配置JindoSDK

1. 进入HDFS服务的**core-site**页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 在**集群服务**页签，单击HDFS服务区域的**配置**。
   v. 在HDFS服务的**服务配置**区域，单击**core-site**页签。
2. 新增和修改配置项。
   新增配置项的具体操作，请参见添加配置项。修改配置项的具体操作，请参见修改配置项。

内容	参数	描述
配置JindoSDK OSS实现类	fs.AbstractFileSystem.oss.impl	固定值为com.aliyun.jindodata.oss.OSS。
	fs.oss.impl	固定值为com.aliyun.jindodata.oss.JindoOssFileSystem。
	fs.xengine	固定值为jindofsx。
配置Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	OSS-HDFS服务的Endpoint。例如，cn-***.oss-dls.aliyuncs.com。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 说明 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
开启缓存	fs.jindofsx.data.cache.enable	数据缓存开关： false（默认值）：禁用数据缓存。 true：启用数据缓存。 说明 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。

3. 重启HDFS服务，具体操作请参见重启服务。

步骤三：使用JindoSDK访问OSS-HDFS服务

您可以使用Hadoop Shell访问JindoFS，常用命令如下：

- put操作

```
hadoop fs -put <path> oss://<BucketName>.<Endpoint>/
```

- ls操作

```
hadoop fs -mkdir oss://<BucketName>.<Endpoint>/<path>
```

- mkdir操作

```
hadoop fs -rm oss://<BucketName>.<Endpoint>/<path>
```

- rm操作

```
hadoop fs -rm oss://<BucketName>/<path>
```

5.4.2.3.3. Hadoop访问JindoFSx统一挂载的数据

JindoSDK为JindoFSx存储加速系统提供了Apache Hadoop支持。本文介绍Hadoop如何处理JindoFSx统一挂载的数据。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> 说明 本文以EMR-3.40.0版本为例介绍。

操作步骤

1. （可选）如果想使用缓存功能，请先配置JindoData服务的AccessKey信息，具体操作请参见步骤一：配置AccessKey。
2. 进入HDFS服务的**core-site**页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 在**集群服务**页签，单击HDFS服务区域的**配置**。
   v. 在HDFS服务的**服务配置**区域，单击**core-site**页签。

3. 新增和修改配置项。

   新增配置项的具体操作，请参见添加配置项。修改配置项的具体操作，请参见修改配置项。

内容	参数	描述
配置实现类	fs.AbstractFileSystem.jindo.impl	固定值为com.aliyun.jindodata.jindo.JINDO。
	fs.jindo.impl	固定值为com.aliyun.jindodata.jindo.JindoFileSystem。
	fs.xengine	固定值为jindofsx。
配置OSS Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	○ OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。 ○ OSS-HDFS服务的Endpoint。例如，cn-***.oss-dls.aliyuncs.com。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 说明 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
缓存加速功能（可选） 注意 如果使用缓存功能，请执行步骤1。	fs.jindofsx.data.cache.enable	数据缓存开关： ○ false（默认值）：禁用数据缓存。 ○ true：启用数据缓存。 说明 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。
	fs.jindofsx.meta.cache.enable	元数据缓存开关： ○ false（默认值）：禁用元数据缓存。 ○ true：启用元数据缓存。
	fs.jindofsx.slice.cache.enable	小文件缓存优化开关： ○ false（默认值）：禁用小文件缓存。 ○ true：启用小文件缓存。
	fs.jindofsx.ram.cache.enable	内存缓存开关： ○ false（默认值）：禁用内存缓存。 ○ true：启用内存缓存。
	fs.jindofsx.short.circuit.enable	短路读开关： ○ true（默认值）：打开。 ○ false：关闭。

4. 重启Presto服务，具体操作请参见重启服务。
5. 挂载OSS或OSS-HDFS服务目录。

   挂载OSS-HDFS服务目录语法如下。

```
jindo admin -mount <path> <realpath>
```

   例如，执行以下命令挂载OSS-HDFS服务目录。

```
jindo admin -mount /jindodls oss://<yourBucketName>.<yourBucketEndpoint>/
```

   说明 执行上述命令后，则 `/jindodls` 目录真正挂载的文件路径是 `oss://<yourBucketName>.<yourBucketEndpoint>/` 。

6. 访问OSS或OSS-HDFS服务。

   您可以使用Hadoop Shell访问OSS或OSS-HDFS服务，常用命令如下：

- put操作

```
hadoop fs -put <path> jindo://emr-header-1:8101/jindooss/
```

- ls操作

```
hadoop fs -ls jindo://emr-header-1:8101/jindooss/
```

- mkdir操作

```
hadoop fs -mkdir jindo://emr-header-1:8101/jindooss/<path>
```

- rm操作

```
hadoop fs -rm jindo://emr-header-1:8101/jindooss/<path>
```

5.4.2.3.4. Spark处理阿里云OSS上的数据+JindoFSx透明加速

JindoSDK是一个简单易用面向Hadoop和Spark生态的OSS客户端，为阿里云OSS提供高度优化的Hadoop FileSystem实现，Spark使用JindoSDK相对于使用Hadoop社区OSS客户端，可以获得更好的性能，同时还能获得阿里云E-MapReduce产品的技术支持。JindoFSx存储加速系统提供了透明缓存的使用方式，兼容原生OSS存储方式，文件以对象的形式存储在OSS上，每个文件根据实际访问情况会在本地进行缓存，提升访问OSS的效率，同时兼容了原有OSS文件形式，数据访问上能够与其他OSS客户端完全兼容，作业访问OSS的方式无需做任何修改。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> **说明** 本文以EMR-3.40.0版本为例介绍。

操作步骤

1. （可选）如果想使用缓存功能，请先配置JindoData服务的AccessKey信息，具体操作请参见步骤一：配置AccessKey。
2. 进入HDFS服务的**core-site**页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 在**集群服务**页签，单击HDFS服务区域的**配置**。
   v. 在HDFS服务的**服务配置**区域，单击**core-site**页签。
3. 配置JindoSDK。

   新增配置项的具体操作，请参见添加配置项。修改配置项的具体操作，请参见修改配置项。

内容	参数	描述
配置JindoSDK OSS实现类	fs.AbstractFileSystem.oss.impl	固定值为com.aliyun.jindodata.oss.OSS。
	fs.oss.impl	固定值为com.aliyun.jindodata.oss.JindoOssFileSystem。
	fs.xengine	固定值为jindofsx。
配置Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。

内容	参数	描述
开启透明缓存加速（可选） 注意 如果使用缓存功能，请执行步骤1。	fs.jindofsx.data.cache.enable	数据缓存开关： false（默认值）：禁用数据缓存。 true：启用数据缓存。 说明 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。

4. 使用Spark访问OSS。
   - 创建表
     ```
     create table test_oss (c1 string) location "oss://<Bucket>/<path>";
     ```
   - 插入数据
     ```
     insert into table test_oss values ("testdata");
     ```
   - 查询OSS表
     ```
     select * from test_oss;
     ```

5.4.2.3.5. Spark处理阿里云OSS-HDFS服务上的数据+JindoFSx透明加速

本文为您介绍Spark如何使用JindoSDK处理OSS-HDFS服务（JindoFS服务）中的数据。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

说明 本文以EMR-3.40.0版本为例介绍。

操作步骤

1. （可选）如果想使用缓存功能，请先配置JindoData服务的AccessKey信息，具体操作请参见步骤一：配置AccessKey。
2. 进入HDFS服务的**core-site**页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 在**集群服务**页签，单击HDFS服务区域的**配置**。
   v. 在HDFS服务的**服务配置**区域，单击**core-site**页签。
3. 新增和修改配置项。

   #### 全局配置

   新增配置项的具体操作，请参见添加配置项。修改配置项的具体操作，请参见修改配置项。

内容	参数	描述
配置JindoSDK OSS-HDFS服务实现类	fs.AbstractFileSystem.oss.impl	固定值为com.aliyun.jindodata.oss.OSS。
	fs.oss.impl	固定值为com.aliyun.jindodata.oss.JindoOssFileSystem。
	fs.xengine	固定值为jindofsx。
配置OSS Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	OSS-HDFS服务的Endpoint。例如，cn-***.oss-dls.aliyuncs.com。

内容	参数	描述
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
开启透明缓存加速（可选） **注意** 如果使用缓存功能，请执行步骤1。	fs.jindofsx.data.cache.enable	数据缓存开关： false（默认值）：禁用数据缓存。 true：启用数据缓存。 **说明** 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。

任务级别配置

使用参数在提交Spark任务的时候设置JindoSDK，示例如下。

```
spark-submit --conf spark.hadoop.fs.AbstractFileSystem.oss.impl=com.aliyun.jindodata.oss.OSS --conf spark.hadoop.fs.oss.impl=com.aliyun.jindodata.oss.JindoOssFileSystem --conf spark.hadoop.fs.oss.accessKeyId=xxx  --conf spark.hadoop.fs.oss.accessKeySecret=xxx --conf spark.hadoop.fs.jindofsx.namespace.rpc.address=hostname:port --conf spark.hadoop.fs.jindofsx.data.cache.enable=true
```

4. 使用Spark访问OSS-HDFS服务。
   - 创建表
     ```
     create table test_oss (c1 string) location "oss://<Bucket>/<path>";
     ```
   - 插入数据
     ```
     insert into table test_oss values ("testdata");
     ```
   - 查询OSS表
     ```
     select * from test_oss;
     ```

5.4.2.3.6. Spark处理JindoFSx统一挂载的数据

本文为您介绍Spark如何处理JindoFSx统一挂载的数据。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

说明 本文以EMR-3.40.0版本为例介绍。

操作步骤

1. （可选）如果想使用缓存功能，请先配置JindoData服务的AccessKey信息，具体操作请参见步骤一：配置AccessKey。
2. 进入HDFS服务的**core-site**页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 在**集群服务**页签，单击HDFS服务区域的**配置**。
   v. 在HDFS服务的**服务配置**区域，单击**core-site**页签。
3. 新增和修改配置项。

全局配置

i. 新增和修改配置项。

新增配置项的具体操作，请参见添加配置项。修改配置项的具体操作，请参见修改配置项。

内容	参数	描述
配置实现类	fs.AbstractFileSystem.jindo.impl	固定值为com.aliyun.jindodata.jindo.JINDO。
	fs.jindo.impl	固定值为com.aliyun.jindodata.jindo.JindoFileSystem。
	fs.xengine	固定值为jindofsx。
配置Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	■ OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。 ■ OSS-HDFS服务的Endpoint。例如，cn-***.oss-dls.aliyuncs.com。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 说明 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
缓存加速功能（可选） 注意 如果使用缓存功能，请执行步骤1。	fs.jindofsx.data.cache.enable	数据缓存开关： ■ false（默认值）：禁用数据缓存。 ■ true：启用数据缓存。 说明 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。
	fs.jindofsx.meta.cache.enable	元数据缓存开关： ■ false（默认值）：禁用元数据缓存。 ■ true：启用元数据缓存。
	fs.jindofsx.slice.cache.enable	小文件缓存优化开关： ■ false（默认值）：禁用小文件缓存。 ■ true：启用小文件缓存。
	fs.jindofsx.ram.cache.enable	内存缓存开关： ■ false（默认值）：禁用内存缓存。 ■ true：启用内存缓存。
	fs.jindofsx.short.circuit.enable	短路读开关： ■ true（默认值）：打开。 ■ false：关闭。

任务级别配置

使用参数在提交Spark任务的时候设置JindoSDK，示例如下。

```
spark-submit --conf spark.hadoop.fs.AbstractFileSystem.oss.impl=com.aliyun.jindodata.oss.OSS --conf spark.hadoop.fs
.oss.impl=com.aliyun.jindodata.oss.JindoOssFileSystem --conf spark.hadoop.fs.oss.accessKeyId=xxx  --conf spark.hado
op.fs.oss.accessKeySecret=xxx --conf spark.hadoop.fs.oss.endpoint=oss-cn-xxx-internal.aliyuncs.com --conf spark.had
oop.fs.jindofsx.namespace.rpc.address=hostname:port --conf spark.hadoop.fs.jindofsx.data.cache.enable=true
```

4. 挂载OSS或OSS-HDFS服务目录。

挂载语法如下。

```
jindo admin -mount <path> <realpath>
```

例如，执行以下命令挂载OSS目录。

```
jindo admin -mount /jindooss oss://<yourBucketName>.<yourBucketEndpoint>/
```

> **说明** 执行上述命令后，则 `/jindooss` 目录真正挂载的文件路径是 `oss://<yourBucketName>.<yourBucketEndpoint>/` 。

5. 使用Spark访问OSS。
   - 创建表
     ```
     create table test_oss (c1 string) location "jindo://emr-header-1:8101/jindooss/<path>";
     ```
   - 插入数据
     ```
     insert into table test_oss values ("testdata");
     ```
   - 查询OSS表
     ```
     select * from test_oss;
     ```

5.4.2.3.7. Hive处理阿里云OSS上的数据+JindoFSx透明加速

JindoFSx存储加速系统提供了透明缓存的使用方式，兼容原生OSS存储方式，文件以对象的形式存储在OSS上，每个文件根据实际访问情况会在本地进行缓存，提升访问OSS的效率，同时兼容了原有OSS文件形式，数据访问上能够与其他OSS客户端完全兼容，作业访问OSS的方式无需做任何修改。本文为您介绍Hive如何处理阿里云OSS上的数据。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> **说明** 本文以EMR-3.40.0版本为例介绍。

操作步骤

1. 配置JindoData服务的AccessKey信息，具体操作请参见步骤一：配置AccessKey。
2. 进入HDFS服务的**core-site**页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 在**集群服务**页签，单击HDFS服务区域的**配置**。
   v. 在HDFS服务的**服务配置**区域，单击**core-site**页签。
3. 新增和修改配置项。

   新增配置项的具体操作，请参见添加配置项。修改配置项的具体操作，请参见修改配置项。

内容	参数	描述
配置OSS实现类	fs.AbstractFileSystem.oss.impl	固定值为com.aliyun.jindodata.oss.OSS。
	fs.jindo.impl	固定值为com.aliyun.jindodata.oss.JindoOssFileSystem。
	fs.xengine	固定值为jindofsx。
配置OSS Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。

内容	参数	描述
开启透明缓存加速（可选） 注意 如果使用缓存功能，请执行步骤1。	fs.jindofsx.data.cache.enable	数据缓存开关： false（默认值）：禁用数据缓存。 true：启用数据缓存。 说明 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。

4. 重启Hive服务，具体操作请参见重启服务。
5. 指定OSS存储。
   - 在创建数据库和表时，可以指定OSS路径，把数据库或表的数据默认保存到OSS上，示例如下。

     ```
     CREATE DATABASE db_on_oss1 LOCATION 'oss://bucketname/path/to/db1';
     CREATE TABLE db2.table_on_oss ... LOCATION 'oss://bucketname/path/to/db2/tablepath';
     ```

   - 在Hive Metastore的*hive-site.xml*配置中设置参数hive.metastore.warehouse.dir到OSS路径，并重启Hive Metastore，则后续创建的数据库和这些数据库下的表都会默认存储于OSS。

     ```
     <configuration>
         <property>
             <name>hive.metastore.warehouse.dir</name>
             <value>oss://bucketname/path/to/warehouse</value>
         </property>
     </configuration>
     ```

   - 给已有表添加OSS的分区。

     ```
     ALTER TABLE existed_table ADD PARTITION (dt='2021-03-01', country='cn') LOCATION 'oss://bucketname/path/to/us/part210
     301cn';
     ```

5.4.2.3.8. Hive处理阿里云OSS-HDFS上的数据+JindoFSx透明加速

JindoFSx存储加速系统提供了透明缓存的使用方式，兼容原生OSS存储方式，文件以对象的形式存储在OSS-HDFS上，每个文件根据实际访问情况会在本地进行缓存，提升访问OSS-HDFS的效率，同时兼容了原有OSS文件形式。本文为您介绍Hive如何处理OSS-HDFS服务（JindoFS服务）中的数据。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

说明 本文以EMR-3.40.0版本为例介绍。

操作步骤

1. 配置JindoData服务的AccessKey信息，具体操作请参见步骤一：配置AccessKey。
2. 进入HDFS服务的**core-site**页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 在**集群服务**页签，单击HDFS服务区域的**配置**。
   v. 在HDFS服务的**服务配置**区域，单击**core-site**页签。
3. 新增和修改配置项。

   新增配置项的具体操作，请参见添加配置项。修改配置项的具体操作，请参见修改配置项。

内容	参数	描述
配置实现类	fs.AbstractFileSystem.oss.impl	固定值为com.aliyun.jindodata.oss.OSS。
	fs.oss.impl	固定值为com.aliyun.jindodata.oss.JindoOssFileSystem。
	fs.xengine	固定值为jindofsx。
	fs.oss.accessKeyId	OSS的AccessKey ID。

内容 配置OSS Accesskey	参数	描述
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	OSS-HDFS服务的Endpoint。例如，cn-***.oss-dls.aliyuncs.com。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
开启透明缓存加速（可选） **注意** 如果使用缓存功能，请执行步骤1。	fs.jindofsx.data.cache.enable	数据缓存开关： false（默认值）：禁用数据缓存。 true：启用数据缓存。 **说明** 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。

4. 重启Hive服务，具体操作请参见重启服务。
5. 指定OSS-HDFS服务路径。
   - 在创建数据库和表时，可以指定OSS-HDFS服务的路径，把数据库或表的数据默认保存到OSS-HDFS服务上，示例如下。

     ```
     CREATE DATABASE db_on_oss1 LOCATION 'oss://bucket_name/path/to/db1';
     CREATE TABLE db2.table_on_oss ... LOCATION 'oss://bucket_name/path/to/db2/tablepath';
     ```

   - 在Hive Metastore的*hive-site.xml*配置中设置参数hive.metastore.warehouse.dir到OSS-HDFS服务路径，并重启Hive Metastore，则后续创建的数据库和这些数据库下的表都会默认存储于OSS-HDFS服务。

     ```
     <configuration>
         <property>
             <name>hive.metastore.warehouse.dir</name>
             <value>oss://bucket_name/path/to/warehouse</value>
         </property>
     </configuration>
     ```

   - 给已有表添加OSS的分区。

     ```
     ALTER TABLE existed_table ADD PARTITION (dt='2021-03-01', country='cn') LOCATION 'oss://bucket_name/path/to/us/part210301cn';
     ```

5.4.2.3.9. Hive处理JindoFSx统一挂载的数据

Hive是大数据的常用工具之一，使用Hive可以搭建离线数仓。随着数据量不断增长，传统的基于HDFS存储的数仓可能无法以较低成本满足用户的需求，常见的可以结合对象存储等云存储使用Hive。本文为您介绍Hive如何处理JindoFSx统一挂载的数据。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

说明 本文以EMR-3.40.0版本为例介绍。

操作步骤

1. （可选）如果想使用缓存功能，请先配置JindoData服务的AccessKey信息，具体操作请参见步骤一：配置AccessKey。
2. 进入HDFS服务的**core-site**页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 在**集群服务**页签，单击HDFS服务区域的**配置**。
   v. 在HDFS服务的**服务配置**区域，单击**core-site**页签。
3. 新增和修改配置项。

   新增配置项的具体操作，请参见添加配置项。修改配置项的具体操作，请参见修改配置项。

内容	参数	描述
配置实现类	fs.AbstractFileSystem.jindo.impl	固定值为com.aliyun.jindodata.jindo.JINDO。
	fs.jindo.impl	固定值为com.aliyun.jindodata.jindo.JindoFileSystem。
	fs.xengine	固定值为jindofsx。
配置Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	○ OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。 ○ OSS-HDFS服务的Endpoint。例如，cn-***.oss-dls.aliyuncs.com。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 说明 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
缓存加速功能（可选） 注意 如果使用缓存功能，请执行步骤1。	fs.jindofsx.data.cache.enable	数据缓存开关： ○ false（默认值）：禁用数据缓存。 ○ true：启用数据缓存。 说明 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。
	fs.jindofsx.meta.cache.enable	元数据缓存开关： ○ false（默认值）：禁用元数据缓存。 ○ true：启用元数据缓存。
	fs.jindofsx.slice.cache.enable	小文件缓存优化开关： ○ false（默认值）：禁用小文件缓存。 ○ true：启用小文件缓存。
	fs.jindofsx.ram.cache.enable	内存缓存开关： ○ false（默认值）：禁用内存缓存。 ○ true：启用内存缓存。
	fs.jindofsx.short.circuit.enable	短路读开关： ○ true（默认值）：打开。 ○ false：关闭。

4. 重启Hive服务，具体操作请参见重启服务。
5. 挂载OSS或OSS-HDFS服务目录。

   挂载语法如下。

   ```
   jindo admin -mount <path> <realpath>
   ```

   例如，执行以下命令挂载OSS-HDFS服务目录。

   ```
   jindo admin -mount /jindodls oss://<yourBucketName>.<yourBucketEndpoint>/
   ```

   说明 执行上述命令后，则 `/jindodls` 目录真正挂载的文件路径是 `oss://<yourBucketName>.<yourBucketEndpoint>/` 。
6. 指定存储路径。
   - 在创建数据库和表时，可以指定OSS路径，把数据库或表的数据默认保存到OSS上，示例如下。

     ```
     CREATE DATABASE db_on_oss1 LOCATION 'jindo://headerhost:8101/jindooss/path/to/db1';
     CREATE TABLE db2.table_on_oss ... LOCATION 'jindo://headerhost:8101/jindooss/path/to/db2/tablepath';
     ```

- 在Hive Metastore的*hive-site.xml*配置中设置参数hive.metastore.warehouse.dir到OSS路径，并重启Hive Metastore，则后续创建的数据库和这些数据库下的表都会默认存储于OSS。

```
<configuration>
    <property>
        <name>hive.metastore.warehouse.dir</name>
        <value>oss://bucket_name/path/to/warehouse</value>
    </property>
</configuration>
```

- 给已有表添加OSS的分区。

```
ALTER TABLE existed_table ADD PARTITION (dt='2021-03-01', country='cn') LOCATION 'jindo://headerhost:8101/jindooss/path/to/us/part210301cn';
```

5.4.2.3.10. Presto查询阿里云OSS上数据+JindoFSx透明加速

Presto是一个开源的分布式SQL查询引擎，适用于交互式分析查询。本文为您介绍Presto如何使用JindoSDK查询阿里云OSS数据湖存储。

背景信息

JindoSDK包含一些高级调优参数，配置方式以及配置项请参见JindoSDK高级参数配置。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> **说明** 本文以EMR-3.40.0版本为例介绍。

操作步骤

1. 配置JindoData服务的AccessKey信息，具体操作请参见步骤一：配置AccessKey。
2. 进入HDFS服务的**core-site**页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 在**集群服务**页签，单击HDFS服务区域的**配置**。
   v. 在HDFS服务的**服务配置**区域，单击**core-site**页签。
3. 新增和修改配置项。

   新增配置项的具体操作，请参见添加配置项。修改配置项的具体操作，请参见修改配置项。

内容	参数	描述
配置JindoSDK OSS实现类	fs.AbstractFileSystem.oss.impl	固定值为com.aliyun.jindodata.oss.OSS。
	fs.oss.impl	固定值为com.aliyun.jindodata.oss.JindoOssFileSystem。
	fs.xengine	固定值为jindofsx。
配置OSS Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。

内容	参数	描述
开启缓存	fs.jindofsx.data.cache.enable	数据缓存开关： ○ false（默认值）：禁用数据缓存。 ○ true：启用数据缓存。 说明 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。

4. 重启Presto服务，具体操作请参见重启服务。

使用示例

以下以最常用的Hive Catalog为例，使用Presto创建一个OSS上的Schema，并执行一些简单的SQL示例。

1. 执行以下命令，启动Presto客户端。

```
presto --server emr-header-1:9090 --catalog hive
```

2. 执行以下命令，创建并使用一个Location位于OSS上的Schema。

```
create schema testDB with (location='oss://<bucket>/<schema_dir>');
use testDB;
```

3. 执行以下命令，创建表。

```
create table tbl (key int, val int);
```

4. 执行以下命令，向创建的表中插入数据。

```
insert into tbl values (1,666);
```

5. 执行以下命令，查询表数据。

```
select * from tbl;
```

5.4.2.3.11. Presto使用JindoSDK查询阿里云OSS-HDFS服务上的数据

Presto是一个开源的分布式SQL查询引擎，适用于交互式分析查询。本文为您介绍Presto如何使用JindoSDK查询OSS-HDFS服务（JindoFS服务）中的数据。

背景信息

JindoSDK包含一些高级调优参数，配置方式以及配置项请参见JindoSDK高级参数配置。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> 说明 本文以EMR-3.40.0版本为例介绍。

步骤一：配置AccessKey

1. 进入JindoData服务的**common**页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 单击JindoData服务区域的**配置**。
   v. 在JindoData服务的**服务配置**区域，单击**common**页签。
2. 新增配置。
   i. 单击**新增配置项**。

ii. 在**新增配置项**对话框中，新增以下配置项。

新增配置项的具体操作，请参见添加配置项。

- 全局方式配置（所有Bucket使用同一种方式）

参数	描述
jindofsx.oss.accessKeyId	OSS的AccessKey ID。
jindofsx.oss.accessKeySecret	OSS的AccessKey Secret。
jindofsx.oss.endpoint	OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。

- 按照Bucket配置

参数	描述
jindofsx.oss.bucket.XXX.accessKeyId	`XXX`的Bucket的AccessKey ID。
jindofsx.oss.bucket.XXX.accessKeySecret	`XXX`的Bucket的AccessKey Secret。
jindofsx.oss.bucket.XXX.endpoint	`XXX`的Bucket的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
jindofsx.oss.bucket.YYY.accessKeyId	`YYY`的Bucket的AccessKey ID。
jindofsx.oss.bucket.YYY.accessKeySecret	`YYY`的Bucket的AccessKey Secret。
jindofsx.oss.bucket.YYY.endpoint	`YYY`的Bucket的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。

> **说明** `XXX`和`YYY`为OSS Bucket的名称。

iii. 单击**确定**。

iv. 在**确认修改配置**对话框中，输入执行原因，单击**确定**。

3. 重启服务。

i. 在JindoData服务页面，选择**更多操作 > 重启**。

ii. 在**重启JindoData服务**对话框中，输入执行原因，单击**确定**。

iii. 在**确认**对话框中，单击**确定**。

步骤二：配置JindoSDK

1. 进入HDFS服务的**core-site**页签。

i. 登录EMR on ECS控制台。

ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。

iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。

iv. 在**集群服务**页签，单击HDFS服务区域的**配置**。

v. 在HDFS服务的**服务配置**区域，单击**core-site**页签。

2. 新增和修改配置项。

新增配置项的具体操作，请参见添加配置项。修改配置项的具体操作，请参见修改配置项。

内容	参数	描述
配置实现类	fs.AbstractFileSystem.oss.impl	固定值为com.aliyun.jindodata.oss.OSS。
	fs.oss.impl	固定值为com.aliyun.jindodata.oss.JindoOssFileSystem。
	fs.xengine	固定值为jindofsx。
配置Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	OSS-HDFS服务的Endpoint。例如，cn-***.oss-dls.aliyuncs.com。

内容	参数	描述
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 说明 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
开启缓存加速	fs.jindofsx.data.cache.enable	数据缓存开关： false（默认值）：禁用数据缓存。 true：启用数据缓存。 说明 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。
	fs.jindofsx.meta.cache.enable	元数据缓存开关： false（默认值）：禁用元数据缓存。 true：启用元数据缓存。
	fs.jindofsx.slice.cache.enable	小文件缓存优化开关： false（默认值）：禁用小文件缓存。 true：启用小文件缓存。
	fs.jindofsx.ram.cache.enable	内存缓存开关： false（默认值）：禁用内存缓存。 true：启用内存缓存。
	fs.jindofsx.short.circuit.enable	短路读开关： true（默认值）：打开。 false：关闭。

3. 重启Presto服务，具体操作请参见重启服务。

使用示例

以下以最常用的Hive Catalog为例，使用Presto创建一个OSS上的Schema，并执行一些简单的SQL示例。

1. 执行以下命令，启动Presto客户端。

```
presto --server emr-header-1:9090 --catalog hive
```

2. 执行以下命令，创建并使用一个Location位于OSS上的Schema。

```
create schema testDB with (location='oss://<bucket>/<schema_dir>');
use testDB;
```

3. 执行以下命令，创建表。

```
create table tbl (key int, val int);
```

4. 执行以下命令，向创建的表中插入数据。

```
insert into tbl values (1,666);
```

5. 执行以下命令，查询表数据。

```
select * from tbl;
```

5.4.2.3.12. Presto处理JindoFSx统一挂载的数据

Presto是一个开源的分布式SQL查询引擎，适用于交互式分析查询。本文为您介绍Presto如何处理JindoFSx统一挂载的数据。

背景信息

JindoSDK包含一些高级调优参数，配置方式以及配置项请参见JindoSDK高级参数配置。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> **说明** 本文以EMR-3.40.0版本为例介绍。

操作步骤

1. （可选）如果想使用缓存功能，请先配置JindoData服务的AccessKey信息，具体操作请参见步骤一：配置AccessKey。
2. 进入HDFS服务的**core-site**页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 在**集群服务**页签，单击HDFS服务区域的**配置**。
   v. 在HDFS服务的**服务配置**区域，单击**core-site**页签。
3. 新增和修改配置项。

   新增配置项的具体操作，请参见添加配置项。修改配置项的具体操作，请参见修改配置项。

内容	参数	描述
配置实现类	fs.AbstractFileSystem.jindo.impl	固定值为com.aliyun.jindodata.jindo.JINDO。
	fs.jindo.impl	固定值为com.aliyun.jindodata.jindo.JindoFileSystem。
	fs.xengine	固定值为jindofsx。
配置Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	○ OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。 ○ OSS-HDFS服务的Endpoint。例如，cn-***.oss-dls.aliyuncs.com。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
缓存加速功能（可选） **注意** 如果使用缓存功能，请执行步骤1。	fs.jindofsx.data.cache.enable	数据缓存开关： ○ false（默认值）：禁用数据缓存。 ○ true：启用数据缓存。 **说明** 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。
	fs.jindofsx.meta.cache.enable	元数据缓存开关： ○ false（默认值）：禁用元数据缓存。 ○ true：启用元数据缓存。
	fs.jindofsx.slice.cache.enable	小文件缓存优化开关： ○ false（默认值）：禁用小文件缓存。 ○ true：启用小文件缓存。
	fs.jindofsx.ram.cache.enable	内存缓存开关： ○ false（默认值）：禁用内存缓存。 ○ true：启用内存缓存。
	fs.jindofsx.short.circuit.enable	短路读开关： ○ true（默认值）：打开。 ○ false：关闭。

4. 重启Presto服务，具体操作请参见重启服务。
5. 挂载OSS或OSS-HDFS服务目录。

   挂载语法如下。

   ```
   jindo admin -mount <path> <realpath>
   ```

   例如，执行以下命令挂载OSS-HDFS服务目录。

   ```
   jindo admin -mount /jindodls oss://<yourBucketName>.<yourBucketEndpoint>/
   ```

   > **说明** 执行上述命令后，则 `/jindodls` 目录真正挂载的文件路径是 `oss://<yourBucketName>.<yourBucketEndpoint>/` 。

使用示例

以下以最常用的Hive Catalog为例，使用Presto创建一个OSS上的Schema，并执行一些简单的SQL示例。

1. 执行以下命令，启动Presto客户端。

   ```
   presto --server emr-header-1:9090 --catalog hive
   ```

2. 执行以下命令，创建并使用一个Location位于OSS上的Schema。

   ```
   create schema testDB with (location='oss://<bucket>/<schema_dir>');
   use testDB;
   ```

3. 执行以下命令，创建表。

   ```
   create table tbl (key int, val int);
   ```

4. 执行以下命令，向创建的表中插入数据。

   ```
   insert into tbl values (1,666);
   ```

5. 执行以下命令，查询表数据。

   ```
   select * from tbl;
   ```

5.4.2.3.13. Impala处理阿里云OSS上的数据+JindoFSx透明加速

JindoFSx存储加速系统提供了透明缓存的使用方式，兼容原生OSS存储方式，文件以对象的形式存储在OSS上，每个文件根据实际访问情况会在本地进行缓存，提升访问OSS的效率，同时兼容了原有OSS文件形式，数据访问上能够与其他OSS客户端完全兼容，作业访问OSS的方式无需做任何修改。本文为您介绍Impala如何处理阿里云OSS上的数据。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> **说明** 本文以EMR-3.40.0版本为例介绍。

操作步骤

1. 配置JindoData服务的AccessKey信息，具体操作请参见步骤一：配置AccessKey。
2. 进入HDFS服务的**core-site**页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 在**集群服务**页签，单击HDFS服务区域的**配置**。
   v. 在HDFS服务的**服务配置**区域，单击**core-site**页签。
3. 配置JindoSDK。

   新增配置项的具体操作，请参见添加配置项。修改配置项的具体操作，请参见修改配置项。

内容	参数	描述
配置JindoSDK OSS实现类	fs.AbstractFileSystem.oss.impl	固定值为com.aliyun.jindodata.oss.OSS。
	fs.oss.impl	固定值为com.aliyun.jindodata.oss.JindoOssFileSystem。
	fs.xengine	固定值为jindofsx。

内容	参数	描述
配置OSS Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
开启透明缓存加速（可选） **注意** 如果使用缓存功能，请执行步骤1。	fs.jindofsx.data.cache.enable	数据缓存开关： ○ false（默认值）：禁用数据缓存。 ○ true：启用数据缓存。 **说明** 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。

4. 使用Impala访问OSS。
   - 创建表

```
CREATE EXTERNAL TABLE customer_demographics (
 `cd_demo_sk` INT,
 `cd_gender` STRING,
 `cd_marital_status` STRING,
 `cd_education_status` STRING,
 `cd_purchase_estimate` INT,
 `cd_credit_rating` STRING,
 `cd_dep_count` INT,
 `cd_dep_employed_count` INT,
 `cd_dep_college_count` INT)
STORED AS PARQUET
LOCATION 'oss://bucket.endpoint/dir';
```

说明 示例中的LOCATION需要替换为OSS的实际路径。

   - 查询OSS表

```
select * from customer_demographics;
```

5.4.2.3.14. Impala处理阿里云OSS-HDFS上的数据+JindoFSx透明加速

JindoFSx存储加速系统提供了透明缓存的使用方式，兼容原生OSS存储方式，文件以对象的形式存储在OSS-HDFS上，每个文件根据实际访问情况会在本地进行缓存，提升访问OSS-HDFS的效率，同时兼容了原有OSS文件形式。本文为您介绍Impala如何使用JindoSDK处理阿里云OSS-HDFS（JindoFS服务）上的数据。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

说明 本文以EMR-3.40.0版本为例介绍。

操作步骤

1. 配置JindoData服务的AccessKey信息，具体操作请参见步骤一：配置AccessKey。
2. 进入HDFS服务的**core-site**页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 在**集群服务**页签，单击HDFS服务区域的**配置**。
   v. 在HDFS服务的**服务配置**区域，单击**core-site**页签。

3. 新增和修改配置项。

   新增配置项的具体操作，请参见添加配置项。修改配置项的具体操作，请参见修改配置项。

内容	参数	描述
配置实现类	fs.AbstractFileSystem.oss.impl	固定值为com.aliyun.jindodata.oss.OSS。
	fs.oss.impl	固定值为com.aliyun.jindodata.oss.JindoOssFileSystem。
	fs.xengine	固定值为jindofsx。
配置OSS Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	OSS-HDFS服务的Endpoint。例如，cn-***.oss-dls.aliyuncs.com。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 说明 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
开启透明缓存加速（可选） 注意 如果使用缓存功能，请执行步骤1。	fs.jindofsx.data.cache.enable	数据缓存开关： false（默认值）：禁用数据缓存。 true：启用数据缓存。 说明 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。

4. 使用Impala访问OSS。
   - 创建表

```
CREATE EXTERNAL TABLE customer_demographics (
 `cd_demo_sk` INT,
 `cd_gender` STRING,
 `cd_marital_status` STRING,
 `cd_education_status` STRING,
 `cd_purchase_estimate` INT,
 `cd_credit_rating` STRING,
 `cd_dep_count` INT,
 `cd_dep_employed_count` INT,
 `cd_dep_college_count` INT)
STORED AS PARQUET
LOCATION 'oss://bucket.endpoint/dir';
```

   - 查询OSS表

```
select * from customer_demographics;
```

5.4.2.3.15. Impala处理JindoFSx统一挂载的数据

本文为您介绍Impala如何处理JindoFSx统一挂载的数据。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> 说明 本文以EMR-3.40.0版本为例介绍。

操作步骤

1. （可选）如果想使用缓存功能，请先配置JindoData服务的AccessKey信息，具体操作请参见步骤一：配置AccessKey。
2. 进入HDFS服务的core-site页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。

iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。

iv. 在**集群服务**页签，单击HDFS服务区域的**配置**。

v. 在HDFS服务的**服务配置**区域，单击**core-site**页签。

3. 配置JindoSDK。

新增配置项的具体操作，请参见添加配置项。修改配置项的具体操作，请参见修改配置项。

内容	参数	描述
配置实现类	fs.AbstractFileSystem.jindo.impl	固定值为com.aliyun.jindodata.jindo.JINDO。
	fs.jindo.impl	固定值为com.aliyun.jindodata.jindo.JindoFileSystem。
	fs.xengine	固定值为jindofsx。
配置Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	○ OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。 ○ OSS-HDFS服务的Endpoint。例如，cn-***.oss-dls.aliyuncs.com。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
缓存加速功能（可选） **注意** 如果使用缓存功能，请执行步骤1。	fs.jindofsx.data.cache.enable	数据缓存开关： ○ false（默认值）：禁用数据缓存。 ○ true：启用数据缓存。 **说明** 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。
	fs.jindofsx.meta.cache.enable	元数据缓存开关： ○ false（默认值）：禁用元数据缓存。 ○ true：启用元数据缓存。
	fs.jindofsx.slice.cache.enable	小文件缓存优化开关： ○ false（默认值）：禁用小文件缓存。 ○ true：启用小文件缓存。
	fs.jindofsx.ram.cache.enable	内存缓存开关： ○ false（默认值）：禁用内存缓存。 ○ true：启用内存缓存。
	fs.jindofsx.short.circuit.enable	短路读开关： ○ true（默认值）：打开。 ○ false：关闭。

4. 挂载OSS或OSS-HDFS服务目录。

挂载语法如下。

```
jindo admin -mount <path> <realpath>
```

例如，执行以下命令挂载OSS目录。

```
jindo admin -mount /jindooss oss://<yourBucketName>.<yourBucketEndpoint>/
```

说明 执行上述命令后，则 `/jindooss` 目录真正挂载的文件路径是 `oss://<yourBucketName>.<yourBucketEndpoint>/` 。

5. 使用Impala访问OSS。
   - 创建表

```
CREATE EXTERNAL TABLE customer_demographics (
 `cd_demo_sk` INT,
 `cd_gender` STRING,
 `cd_marital_status` STRING,
 `cd_education_status` STRING,
 `cd_purchase_estimate` INT,
 `cd_credit_rating` STRING,
 `cd_dep_count` INT,
 `cd_dep_employed_count` INT,
 `cd_dep_college_count` INT)
STORED AS PARQUET
LOCATION 'jindo://headerhost:8101/jindooss/dir';
```

   - 查询OSS表

```
select * from customer_demographics;
```

5.4.2.3.16. 切换为Hadoop原生的JobCommitter

E-MapReduce（简称EMR）集群默认使用JindoCommitter加速大数据作业，解决OSS等对象存储在Spark、MapReduce等作业使用原生Hadoop JobCommitter时遇到的性能和一致性等问题。如果您不想使用默认的JindoCommitter，则可以参照本文切换为Hadoop原生的JobCommitter。本文为您介绍如何切换为Hadoop原生的JobCommitter。

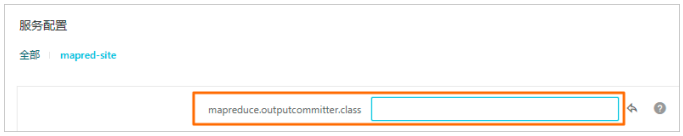
前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> **说明** 本文以EMR-3.40.0版本为例介绍。

步骤一：修改YARN配置

1. 进入YARN服务的**mapred-site**页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 在**集群服务**页签，单击YARN服务区域的**配置**。
   v. 在YARN服务的**服务配置**区域，单击**mapred-site**页签。
2. 修改配置项。

版本	参数	描述
EMR Hadoop 2.x版本	mapreduce.outputcommitter.class	删除参数值，将参数值置为空。 例如，搜索mapreduce.outputcommitter.class配置，删除参数值。 
EMR Hadoop 3.x版本	mapreduce.outputcommitter.factory.class	删除参数值，将参数值置为空。

3. 保存配置。
   i. 单击右上角的**保存**。
   ii. 在**修改信息**对话框中，输入**执行原因**，开启**自动配置更新**。
   iii. 单击**保存**。
4. 重启YARN服务，详情请参见重启服务。

步骤二：修改Spark配置

1. 进入Spark服务的**spark-default**页签。
   i. 在**集群服务**页签，单击Spark服务区域的**配置**。

ii. 在Spark服务的**服务配置**区域，单击**spark-defaults**页签。

2. 修改配置项。

参数	描述
spark.sql.sources.outputCommitterClass	删除参数值，将参数值置为空。
spark.sql.parquet.output.committer.class	固定值为org.apache.parquet.hadoop.ParquetOutputCommitter。

3. （可选）如果您集群的Spark是2.x版本，则还需在**spark-defaults**页签新增以下配置项。
   i. 在Spark**服务配置**区域，单击**新增配置项**。
   ii. 新增参数为spark.sql.hive.commitProtocolClass，参数值为org.apache.spark.sql.execution.datasources.SQLHadoopMapReduceCommitProtocol的配置项。
   iii. 单击**确定**。
4. 保存配置。
   i. 单击右上角的**保存**。
   ii. 在**修改信息**对话框中，输入**执行原因**，开启**自动配置更新**。
   iii. 单击**保存**。
5. 重启Spark服务，详情请参见重启服务。

5.4.3. JindoData 4.0.0版本简介

JindoData 4.x是阿里云E-MapReduce产品SmartData自研组件（SmartData 3.8.0版本）架构升级之后首次发布的版本，重点对接和支持了阿里云OSS存储产品和阿里云OSS-HDFS服务（JindoFS服务）。本文为您介绍JindoData 4.0.0版本支持的功能。

使用限制

- JindoSDK暂不支持OSS上超大文件写入（大于80 GB）。
- JindoSDK暂不支持OSS append方式写入。
- JindoSDK暂不支持OSS客户端加密。
- JindoSDK暂不支持老版本JindoFS Block模式和Cache模式。
- 阿里云OSS-HDFS服务（JindoFS服务）暂不支持老版本JindoFS Block模式系统升级。需要用户通过JindoDistCp迁移工具把数据从老系统迁移到新服务。
- JindoData 4.0.0版本暂未发布JindoFSx缓存系统。

JindoSDK Hadoop

- 为阿里云OSS提供了Java Hadoop SDK，完全兼容Hadoop OSS Connector，性能上大幅领先。
- 支持多种Credential Provider设置方式，包括配置、ECS Role和EMR免密机制。
- 支持写入时归档，包括归档和深冷归档。

JindoShell CLI

- 对Hadoop或HDFS Shell提供额外的命令扩展，为阿里云OSS提供面向Hadoop用户使用的操作方式。
- 支持ls2扩展命令，在标准ls命令的基础上可以额外显示文件或对象在OSS上的存储状态。例如，标准、低频还是归档。
- 支持archive命令，允许指定目录进行转归档操作。
- 支持restore命令，允许指定目录进行解冻操作。

JindoFuse POSIX

为阿里云OSS提供优化后的Fuse客户端，受益于完全Native代码的开发实现，性能在业界大幅领先。

JindoDistCp数据迁移

支持将自建HDFS集群数据迁移到阿里云OSS，针对大文件和大量小文件场景优化。

OSS-HDFS服务

- JindoSDK内置支持访问阿里OSS-HDFS服务（JindoFS服务），提供全面的HDFS接口访问和使用体验。
- 提供Hadoop或HDFS Shell额外的命令扩展，为阿里云OSS-HDFS服务（JindoFS服务）提供面向Hadoop用户使用的操作方式。
- 支持通过HDFS命令和JindoShell扩展命令使用HDFS快照功能，详情请参见管理OSS-HDFS服务快照。
- 支持使用命令导入（UserGroupsMapping），设定用户组信息。
- 支持使用命令设定Hadoop Proxy User规则。

JindoSDK

- 使用JindoSDK访问OSS，详情请参见Hadoop使用JindoSDK访问OSS。
- 使用JindoSDK访问OSS-HDFS服务，详情请参见OSS-HDFS服务快速入门。

6.开发指南
6.1. EMR Studio
6.1.1. EMR Studio概述

EMR Studio是E-MapReduce提供的开源大数据开发套件，包含Apache Zeppelin、Jupyter Notebook和Apache Airflow等开源组件。能够无缝关联EMR集群（EMR on ECS和EMR on ACK）的计算引擎提交任务，并提供了交互式开发、任务调度和任务监控等开源大数据开发使用体验。覆盖了大数据处理ETL、交互式数据分析、机器学习和实时计算等多种应用场景。

EMR Studio核心优势

优势	描述
兼容开源	EMR Studio提供深度优化的开源组件使用体验，100%兼容开源大数据生态。您无需修改任务代码，即可平滑迁移上云。通过EMR Studio数据开发工作台，您可以在开源组件原生UI的基础上无缝衔接开发环节和生产调度环节。 
简化运维	EMR Studio提供开箱即用的大数据开发环境，可以快速响应业务需求。您可以将EMR Studio一键关联至EMR集群（EMR on ECS、EMR on ACK）提交作业，并可以在不关闭Notebook的情况下切换计算集群。EMR Studio自动适配Hive、Spark、Flink、Presto和Impala等多个计算引擎并协同工作。EMR Studio提供了统一的用户管理和认证能力，您可以控制用户访问数据开发控制台。EMR Studio已与阿里云云监控服务集成，您可以在云监控中灵活配置任务调度监控，保障开发环境稳定。
节省成本	您可以根据任务负载灵活变更EMR Studio的硬件资源，压缩调度资源成本。EMR Studio支持使用集群模版功能，您可以设置EMR Studio动态拉起EMR计算集群运行临时任务，当任务结束时自动释放计算集群，能够极大降低计算资源成本。集群模板功能详情，请参见创建集群模板。
便捷集成	EMR Studio采用半托管的部署形态，您可以直接登录集群灵活操作和部署软件，可以将EMR Studio集成至已有系统。EMR Studio作为一款云上产品，支持与数据湖构建（DLF）和对象存储（OSS）等云上产品对接，构建云原生大数据产品架构。您可以在创建EMR Studio时指定OSS bucket路径，EMR Studio将自动备份作业代码和作业日志，并可以通过该路径提交Airflow DAG脚本。

相关文档

- 如果您想了解如何创建EMR Studio，详情请参见创建EMR Studio集群。
- 如果您想快速使用EMR Studio，体验EMR Studio交互式开发和工作流调度功能，详情请参见快速入门。
- 如果您想了解EMR Studio组件的相关配置和示例教程，便于您完成作业，可以参见以下内容：
  - JupyterHub
    - 管理JupyterHub
    - 使用Python3 Kernel运行EMR PySpark
  - Zeppelin
    - Zeppelin概述
    - 交互式开发教程
      - Spark
      - Flink
      - Hive
      - Presto
      - Shell
      - TPCH和TPCDS

- Airflow
  - 基础使用
    - Airflow常用配置说明
    - 管理DAG
    - 配置Airflow报警事件
    - 代码示例
  - 最佳实践
    - 定期调度Zeppelin中的作业
    - 定期调度Jupyter中的作业
    - 动态启动计算集群运行工作流调度

问题反馈

如果您在使用EMR Studio集群过程中有任何疑问或问题，请提交工单或联系我们的技术人员协助处理，同时也欢迎您使用钉钉搜索钉钉群号31675206加入钉钉群进行反馈或交流。

6.1.2. 创建EMR Studio集群

本文为您介绍如何在E-MapReduce（简称EMR）控制台上创建EMR Studio集群。

前提条件

已完成RAM授权，详细信息请参见角色授权。

> **说明** 首次创建EMR Studio集群时会弹出授权该角色的窗口，请使用阿里云账号对系统角色AliyunECSInstanceForEMRStudioRole进行授权。

使用限制

EMR Studio集群仅支持绑定到同一个VPC内的EMR集群，不支持跨VPC。

操作步骤

1. 进入创建集群页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
      - 地域：创建的集群将会在对应的地域内，一旦创建不能修改。
      - 资源组：默认显示账号全部资源。
   iii. 单击**创建集群**，进行创建。
2. 在创建集群页面，完成集群相关配置。

   创建集群时，您需要对集群进行软件配置、硬件配置和基础配置。

   > **注意** 集群创建完成后，除了集群名称以外，其他配置均无法修改，所以在创建时请仔细确认各项配置。

   i. 软件配置。

配置项	描述
集群类型	选择EMR Studio。
产品版本	默认最新的软件版本。
必选服务	默认的服务组件，后期可以在管理页面中启停服务。
高级设置	**软件自定义配置**：可指定JSON文件对集群中的基础软件（例如Hadoop、Spark和Hive等）进行配置，详细使用方法请参见软件配置。默认不开启。

ii. 硬件配置。

区域	配置项	描述
付费类型	**付费类型**	默认包年包月。当前支持的付费类型如下： ■ **按量付费**：一种后付费模式，即先使用再付费。按量付费是根据实际使用的小时数来支付费用，每小时计费一次，适合短期的测试任务或是灵活的动态任务。 ■ **包年包月**：一种预付费模式，即先付费再使用。
网络配置	**可用区**	可用区为在同一地域下的不同物理区域，可用区之间内网互通。通常使用默认的可用区即可。
	网络类型	默认专有网络。
	VPC	选择在该地域的VPC。如果没有可用的VPC，单击**创建VPC/子网（交换机）**前往新建。 **注意** 因为EMR Studio集群仅支持关联同一个VPC内的EMR计算集群，所以创建EMR Studio集群时需要选择与EMR计算集群相同的VPC。
	交换机	选择在对应VPC下可用区的交换机，如果在这个可用区没有可用的交换机，则需要新创建一个。
	安全组名称	选择已有的安全组。安全组详情请参见安全组概述。 您也可以单击**新建安全组**，然后直接输入安全组名称来新建一个安全组。 **注意** 禁止使用ECS上创建的企业安全组。
实例	**选型配置**	EMR Studio最小模型为1个Master，Core数量可以为0。Core数量会影响Airflow的运行模式，不影响其他组件。如果Core数量为0，则Airflow的运行模式就是Local模式（LocalExecutor），如果Core数量大于0，则Airflow的运行模式是分布式模式（CeleroyExecutor）。建议您根据业务创建选择集群规模： ■ **Master实例**：主要负责Master组件的部署，推荐机型ecs.c7.2xlarge。您可以根据实际负载调整实例规格。 ■ **系统盘配置**：根据需要选择SSD云盘、ESSD云盘或者高效云盘。 ■ **系统盘大小**：根据需要调整磁盘容量，推荐至少200 GB。 ■ **数据盘配置**：根据需要选择SSD云盘、ESSD云盘或者高效云盘。 ■ **数据盘大小**：根据需要调整磁盘容量，推荐至少300 GB。 ■ **Master数量**：默认1台。 ■ **Core实例**：主要负责集群所有数据的存储，推荐机型ecs.c7.2xlarge。您可以根据实际负载调整实例规格。 ■ **系统盘配置**：根据需要选择SSD云盘、ESSD云盘或者高效云盘。 ■ **系统盘大小**：根据需要调整磁盘容量，推荐至少200 GB。 ■ **数据盘配置**：根据需要选择SSD云盘、ESSD云盘或者高效云盘。 ■ **数据盘大小**：根据需要调整磁盘容量，推荐至少300 GB。 ■ **Core数量**：默认1台，可以根据业务场景调整。 ■ 仅使用EMR Studio Notebook组件，无调度场景，Core数量可以为0。 ■ 需要使用Airflow进行调度，Core数量至少为1。

iii. 基础配置。

区域	配置项	描述
基础信息	**集群名称**	集群的名字，长度限制为1~64个字符，仅可使用中文、字母、数字、短划线（-）和下划线（_）。
	元数据选择	配置Airflow元数据。 ■ **集群内置MySQL**：表示元数据存储在集群本地环境的MySQL数据库中。 ■ **使用自建RDS**：表示使用自建的阿里云RDS作为元数据库，更多信息请参见配置独立RDS MySQL。
	数据开发存储	EMR Studio集群的数据都会存在OSS上，即使您的EMR Studio集群销毁了，您可以通过重新创建集群来恢复原有集群的状态（包括您的Notebook以及Airflow调度的作业），并且代码和配置都不会丢失。 EMR Studio集群会在您所选OSS路径下创建*logs*、*dags*和*notebook*三个文件夹： ■ *logs*：在*/airflow/*目录下，用于存储Airflow调度的日志信息。 ■ *dags*：在*/airflow/*目录下，用于存储Airflow DAG脚本。 ■ *notebook*：在*/zeppelin/*目录下，用于存储Notebook信息。
	挂载公网	集群是否挂载弹性公网IP地址，建议在创建时开启挂载公网。未开启或是关闭挂载公网，将无法使用EMR控制台访问链接与端口功能查看开源组件Web UI。 **说明** 如果创建集群时，未开启挂载公网，您可以参见弹性公网IP中申请EIP的内容处理，或技术支持处理。
	密钥对	关于密钥对的使用详情，请参见SSH密钥对。
	密码	设置Master节点的登录密码，密码规则：8~30个字符，且必须同时包含大写字母、小写字母、数字和特殊字符。 特殊字符包括：感叹号（!）、at（@）、井号（#）、美元符号（$）、百分号（%）、乘方（^）、and（&）和星号（*）。
高级设置	**添加用户**	添加访问开源大数据软件Web UI的账号。
	权限设置	通过RAM角色为在集群上运行的应用程序提供调用其他阿里云服务所需的必要权限，无需调整，使用默认即可。 ■ **服务角色**：用户将权限授予EMR服务，允许EMR代表用户调用其他阿里云的服务，例如ECS和OSS。 ■ **ECS应用角色**：当用户的程序在EMR计算节点上运行时，可不填写阿里云AccessKey来访问相关的云服务（例如OSS），EMR会自动申请一个临时AccessKey来授权本次访问。**ECS应用角色**用于控制这个AccessKey的权限。
	数据盘加密	默认不开启。 打开**加密开关**，即启动对集群节点ECS中所有属性为云盘的数据盘进行加密的功能。默认使用服务密钥为用户的数据进行加密，也支持使用用户自选密钥为用户的数据进行加密。 **注意** 不支持加密本地盘。
	引导操作	可选配置，您可以在集群启动Hadoop前执行您自定义的脚本，详情请参见引导操作。
	标签	可选配置，您可以在创建集群时绑定标签，也可以在集群创建完成后，在集群详情页绑定标签，详情请参见设置标签。
	资源组	可选配置。详情请参见使用资源组。

说明 页面右边会显示您所创建集群的配置清单以及集群费用。根据不同的付费类型，展示不同的价格信息。

3. 当所有的信息确认正确有效后，选中服务条款，单击**创建**。

注意

- 按量付费集群：立刻开始创建。

  集群创建完成后，集群的状态变为**空闲**。
- 包年包月集群：先生成订单，在支付完成订单以后集群才会开始创建。

6.1.3. 快速入门

本文为您介绍如何通过阿里云E-MapReduce（简称EMR）控制台，快速创建EMR Studio集群并开展交互式开发和工作流调度工作。

背景信息

如果您想了解更多Zeppelin、Jupyter和Airflow的信息，请参见以下内容：

- Zeppelin概述
- 管理JupyterHub
- 定期调度Zeppelin中的作业

前提条件

- 已申请体验EMR Studio的资格，如果还未申请，请提交工单。
- 已创建EMR Studio集群，详情请参见创建EMR Studio集群。

  > **说明** 如果您是第一次创建EMR Studio集群，在创建EMR Studio集群时，控制台会弹出系统角色授权窗口，请您使用主账号对系统角色AliyunECSInstanceForEMRStudioRole进行授权。角色授权详情请参见角色授权。

- 安全组规则已开启8000、8081和8443端口。

  添加安全组规则，详情请参见添加安全组规则。
- 已添加用户，详情请参见添加用户。

使用限制

EMR Studio仅支持与同一VPC下的EMR计算集群进行关联。

操作流程

1. 步骤一：进入数据开发工作台
2. 步骤二：关联计算集群
3. 步骤三：使用Zeppelin交互式开发作业
4. 步骤四：编写Airflow Python脚本
5. 步骤五：使用Airflow调度Notebook作业

步骤一：进入数据开发工作台

1. 进入详情页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
2. 添加Linux用户。

   您可以通过用户管理功能添加用户，详情请参见添加用户。
3. 在左侧导航栏中，选择**访问链接与端口**。
4. 单击**Studio Workspace UI**所在行的链接。

   输入步骤2中添加的用户名和密码，即可正常访问Web UI页面。

   > **说明** 请稍等几秒即可看到Studio Workspace UI的信息。

步骤二：关联计算集群

> **注意** EMR Studio仅支持与同一VPC下的EMR计算集群进行关联。

关联集群后，EMR Studio集群可以访问对应EMR集群上的资源并提交Job。

1. 在**关联集群**页面，单击**可关联集群**页签。

   > **说明** 仅显示同一个VPC下的EMR集群。支持关联Hadoop集群、Dataflow集群和DataScience集群三种集群类型。

2. 选择待关联集群的集群类型。
3. 单击待关联集群操作列的**关联集群**。

4. 在**关联集群**对话框中，单击**绑定**。

   待**已关联集群**页签，显示关联的集群信息时，表示关联成功。

   > **说明** 绑定集群过程大约需要1~2分钟，请您耐心等待。

步骤三：使用Zeppelin交互式开发作业

关联集群后，您可以在Zeppelin Notebook中进行交互式开发。EMR数据开发集群自带教程，本文以Airflow调度教程1为例介绍。

1. 在左侧导航栏中，单击**Zeppelin**。
2. 在**Zeppelin**页面，选择**阿里云EMR数据开发教程 > Airflow教程 > Airflow调度教程1**。

   页面展示如下图所示。

区域	描述
①	Markdown语言（以%md开头），主要是为了添加一些说明文字。
②	Shell脚本语言（以%sh开头），您可以运行Shell脚本，生成数据。
③	Hive（以%hive开头），您可以执行任意的Hive SQL语句。
④	单击 图标，可以切换关联的集群。
⑤	单击 图标，运行当前段落。 **说明** 部分解释器（Interpreter）在第一次使用的时候会去下载依赖，如果提示信息类似 `Interpreter Setting 'hive' is not ready` ，说明下载过程还没结束，请稍后重试。

步骤四：编写Airflow Python脚本

Airflow的调度需要手动编写Python脚本来构建DAG，EMR Studio自动将指定OSS路径内的Python脚本同步至Airflow DAGs，因此，您可以在编辑和上传完DAG脚本之后，进入数据开发工作台，在左侧导航栏中，单击**Airflow**，即可进入DAGs页面查看您创建的DAG。

EMR Studio自带调度教程，您可以在Zeppelin页面，选择**阿里云EMR数据开发教程 > Airflow教程 > Airflow调度教程1**查看。Airflow的基本用法，请参见Apache Airflow。

> **说明** EMR Studio自带用于调度Zeppelin Notebook的Operator（ZeppelinOperator），并提供Note和Paragraph两种级别调度方式。更多信息，请参见定期调度Zeppelin中的作业。

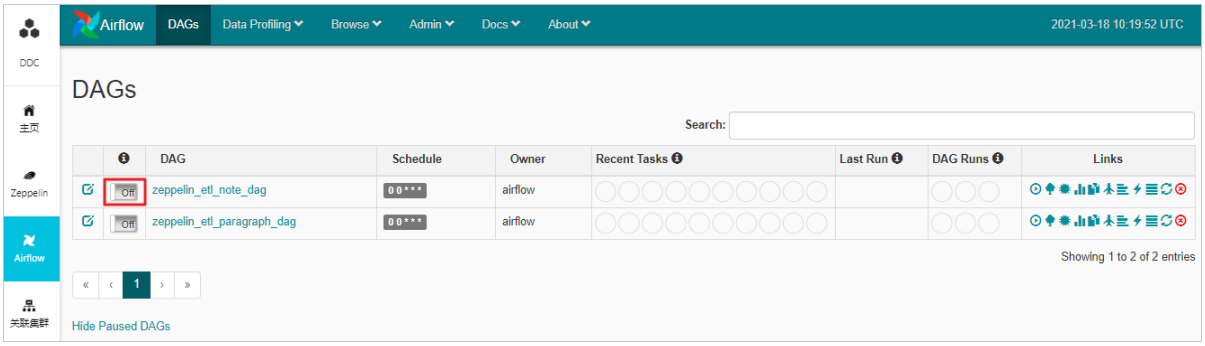
本文以Paragraph级别调度为例，为每个Paragraph构建一个Task，然后串联起来构成一个链式DAG。代码示例如下。

```
from airflow import DAG
from datetime import datetime, timedelta
from airflow.contrib.operators.zeppelin_operator import ZeppelinOperator
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2018, 1, 1),
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 0,
    'retry_delay': timedelta(minutes=5),
}
with DAG('zeppelin_example_paragraph_dag',
         max_active_runs=5,
         schedule_interval='0 0 * * *',
         default_args=default_args) as dag:
    execution_date = "{{ ds }}"
    raw_data_task = ZeppelinOperator(
        task_id='raw_data_task',
        conn_id='zeppelin_default',
        note_id='2FXADMCY9',
        paragraph_id='paragraph_1613703810025_1489985581',
        params= {'dt' : execution_date}
    )
    csv_data_task = ZeppelinOperator(
        task_id='csv_data_task',
        conn_id='zeppelin_default',
        note_id='2FXADMCY9',
        paragraph_id='paragraph_1613705325984_828524031',
        params= {'dt' : execution_date}
    )
    parquet_data_task = ZeppelinOperator(
        task_id='parquet_data_task',
        conn_id='zeppelin_default',
        note_id='2FXADMCY9',
        paragraph_id='paragraph_1613709950691_1420422787',
        params= {'dt' : execution_date}
    )
    query_task = ZeppelinOperator(
        task_id='query_task',
        conn_id='zeppelin_default',
        note_id='2FXADMCY9',
        paragraph_id='paragraph_1613703819960_454383326',
        params= {'dt' : execution_date}
    )
    raw_data_task >> csv_data_task >> parquet_data_task >> query_task
```

步骤五：使用Airflow调度Notebook作业

- 启用DAG调度

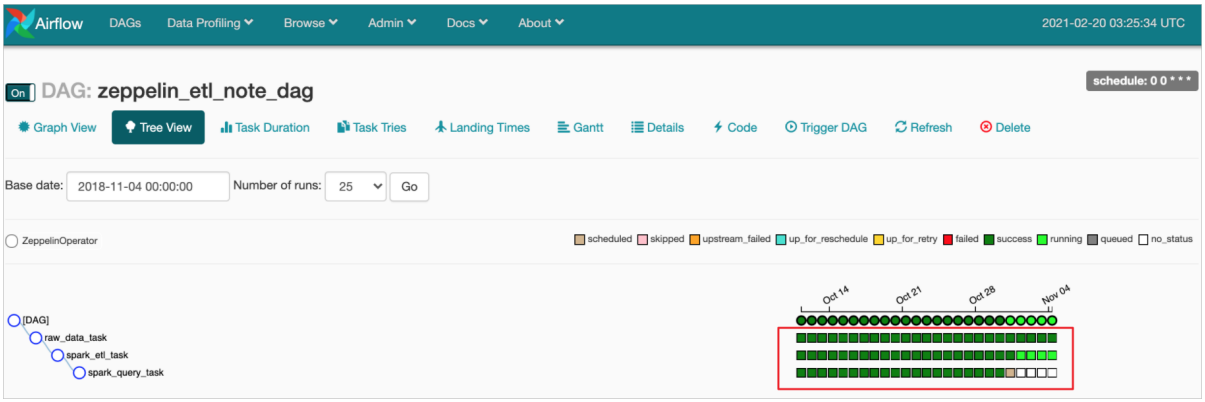
  在**DAGs**页面，打开**off**开关，即可启用DAG调度。



- 查看调度任务的详细信息

  在**DAGs**页面，单击待查看调度任务的DAG，进入**Tree View**页面。

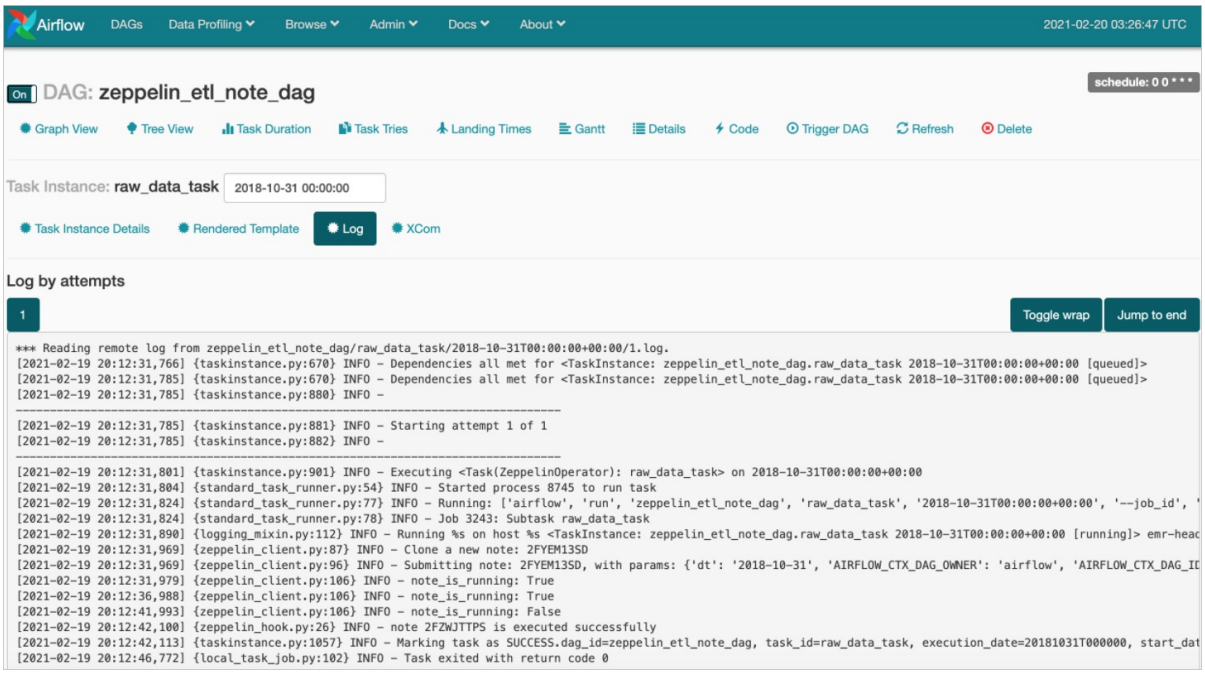
  在**Tree View**页面，您可以查看对应DAG运行调度任务的详细信息。



- Task的运行情况

  在**Tree View**页面，单击 图标，即可查看对应Task的运行情况。

  详细信息如下图所示。



6.1.4. Airflow

6.1.4.1. 基础使用

6.1.4.1.1. Airflow常用配置说明

阿里云E-MapReduce（简称EMR）的EMRStudio集群中，您可以修改Airflow.cfg文件的配置。本文为您介绍Airflow中常用的配置。

Airflow所有配置信息的详情，请参见Configuration Reference。

参数		描述
core	default_timezone	默认时区设置，遵循IANA时区字符。 默认值为Asia/Shanghai。
	parallelism	Airflow全局可以并行运行的最大任务数。 默认值为32。
	default_task_retries	默认状态下每个任务的重试次数，可以在DAG或者Task中重新设置以覆盖默认值。 默认值为0。
scheduler	catchup_by_default	设置此参数为False可以使Scheduler不执行catchup操作，即Airflow不会自行对当前日期和DAG的start_date之间做backfill操作，但是在命令行执行backfill时依然可以生效。此参数也可以在定义DAG时针对每个DAG进行单独配置。 默认值为True。
	scheduler_zombie_task_threshold	本地任务周期性会向数据库发送心跳，如果在此设定值内没有发送心跳，则Scheduler会把该任务标记为失败，并且重新调度该任务。 默认值为300。单位为秒。
celery	worker_concurrency	启动celery worker时的默认并发度。该参数设置了一个worker节点分配的Task数量。请根据worker上的资源和任务本身需要设置该数值。
	worker_autoscale	启动celery worker时的最大和最小并发度（始终保持最少的进程，但是如果有需要的话可以增加到最多进程数）。请根据worker上的资源和任务本身需要设置这两个数值。 如果worker_autoscale选项有设定数值，则参数worker_concurrency会被忽略。例如，设置worker_autoscale的值为16,12时，则会忽略worker_concurrency参数。

6.1.4.1.2. 管理DAG

本文为您介绍如何在在Airflow Web UI页面或OSS控制台管理DAG。

背景信息

本文通过以下两种方式，为您介绍如何管理DAG，您可以根据您的实际情况，选择相应的方式。

- 如果是生产环境，建议您使用方式一：在OSS控制台编辑DAG。
- 如果是开发环境或测试环境，建议您使用方式二：在Airflow Web UI页面管理DAG。

前提条件

- 已创建EMR Studio集群。

  创建集群详情，请参见创建集群。
- 已绑定计算集群。

> 注意
> - 绑定集群页签下，仅显示同一个VPC下的EMR集群。
> - 仅支持绑定Hadoop集群。

方式一：在OSS控制台编辑DAG

警告 无论是通过Code Editor插件还是OSS更改DAG，在Airflow扫描到更新后都会在调度中生效，请谨慎修改！

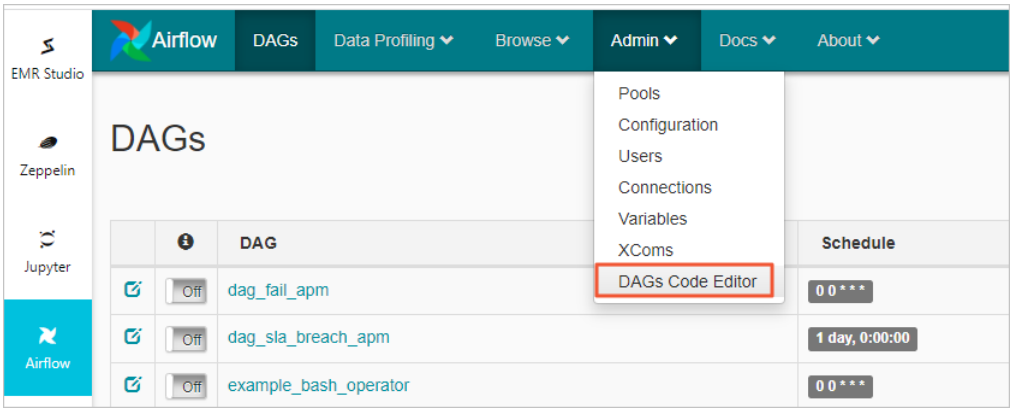
1. 登录 OSS管理控制台。
2. 进入您在创建集群时选择的数据开发存储的*dags*路径下。
3. 您可以对已有的DAG进行以下操作。

操作	描述
上传文件	详细步骤，请参见上传文件。
修改文件	请在您本地修改或新建文件，然后上传文件至OSS控制台。
重命名文件	详细步骤，请参见重命名文件。
新建目录	详细步骤，请参见创建目录。
下载文件	详细步骤，请参见下载文件。
删除文件	详细步骤，请参见删除Object。
删除目录	详细步骤，请参见删除目录。

方式二：在Airflow Web UI页面管理DAG

警告 在Code Editor中修改提交会直接作用在OSS上的对应文件上，不建议在生产环境中使用Code Editor，避免操作失误造成损失。可以在开发或测试中使用Code Editor，以提高效率。

1. 在Airflow Web UI页面，在上方选择**Admin > DAGs Code Editor**。



2. 您可以对已有的DAG进行以下操作。

操作	描述
新建文件	i. 在**Files**页面，单击右上角的**New**。 ii. 输入文件内容。 iii. 单击下方的**Save**。 iv. 在**Save Flie**对话框中，输入文件名，单击**Save**。
修改文件	i. 在**Files**页面，单击目标文件的文件名。 ii. 输入文件内容。 iii. 单击下方的**Save**。
重命名文件	i. 在**Files**页面，单击目标文件的文件名。 ii. 单击下方的**Save as...**。 iii. 在**Save Flie**对话框中，输入文件名，单击**Save**。
新建文件目录	请在OSS控制台创建目录，详细步骤，请参见创建目录。

操作	描述
下载文件	在Files页面，单击目标文件所在行的图标，即可下载目标文件。
删除文件	i. 在Files页面，单击目标文件的文件名。 ii. 单击下方的Delete。 iii. 在Confirm Delete对话框中，单击Delete。 iv. 在Success对话框中，单击OK。

6.1.4.1.3. 配置Airflow报警事件

EMR Studio支持将告警事件推送至云监控，您可以在云监控（CloudMonitor）控制台通过配置告警规则来实现告警通知和管理。

使用限制

确保和EMR Studio集群使用同一个账号。

操作流程

1. 创建报警联系人。

   同一个报警联系人，可以加入多个报警联系组。
2. 创建报警联系组。

   报警联系组是一组报警联系人，可以包含一个或多个报警联系人。
3. 添加报警联系人到报警联系组。
4. 创建事件报警规则。

创建报警联系人

1. 登录云监控控制台。
2. 在左侧导航栏，选择**报警服务 > 报警联系人**。
3. 在**报警联系人**页签，单击**新建联系人**。
4. 在**设置报警联系人**页面，填写报警联系人的姓名、手机号码、邮箱、旺旺和钉钉机器人，**报警通知信息语言**保持默认值**自动**。

   > **说明** **自动**表示云监控根据当前阿里云账号注册时的语言，自动适配报警通知信息的语言。
5. 信息验证无误后，单击**确认**。
6. （可选）报警联系人邮箱和手机号码激活。

   如果您设置了报警联系人的邮箱和手机号码，默认处于**等待激活**状态。报警联系人需要根据邮件和短消息中的激活链接，在24小时内进行激活，否则无法收到报警通知。激活后，您可以在报警联系人列表中看到目标报警联系人的手机号码和邮箱。

创建报警联系组

1. 登录云监控控制台。
2. 在左侧导航栏，选择**报警服务 > 报警联系人**。
3. 单击**报警联系组**页签。
4. 在**报警联系组**页签，单击**新建联系组**。
5. 在**新建联系组**页面，填写报警联系组的组名，并选择报警联系人。
6. 单击**确认**。

添加报警联系人到报警联系组

1. 登录云监控控制台。
2. 在左侧导航栏，选择**报警服务 > 报警联系人**。
3. 在**报警联系人**页签，选中目标报警联系人。
4. 单击下方的**添加到报警联系组**。
5. 在**确认信息**对话框中，选择目标报警联系组。

6. 单击**确定**。

创建事件报警规则

1. 登录云监控控制台。
2. 在左侧导航栏，选择**报警服务 > 报警规则**。
3. 在**报警规则列表**页面，单击**事件报警**页签。
4. 在**事件报警**页面，单击**创建事件报警**。
5. 在**创建/修改事件报警**页面，设置事件报警规则和报警方式。
   i. 在**基本信息**区域，填写**时间报警规则**。
   ii. 在**事件报警规则**区域，**事件类型**保持默认值为**系统事件**，并设置相关参数。

参数	描述
报警规则名称	事件报警规则的名称。
产品类型	事件报警规则的云服务类型。 在**产品类型**下拉列表中选择**E-MapReduce**。
事件类型	事件报警规则的事件类型。 在**产品类型**下拉列表中选择**EMRStudio**。
事件等级	事件报警规则的事件等级。 在**事件等级**下拉列表中选择告警规则，例如，选择严重和警告。
事件名称	事件报警规则的事件名称。
资源范围	事件报警规则作用的资源范围。取值： ■ **全部资源**：当**资源范围**选择**全部资源**时，任何资源发生相关事件，都会按照配置发送通知。 ■ **应用分组**：当**资源范围**选择**应用分组**时，只有指定应用分组内的资源发生相关事件，才会发送通知。
联系人组	事件报警规则的报警联系人组。 在**联系人组**下拉列表中，选择创建报警联系组中新建的报警联系组。
通知方式	事件报警的级别和通知方式。取值： ■ Warning（短信+邮箱+钉钉机器人） ■ Info（邮箱+钉钉机器人）
消息服务队列	事件报警投递到消息服务的指定队列。
函数计算	事件报警投递到函数计算的指定函数。
URL回调	设置URL回调地址和请求方法。填写公网可访问的URL，云监控会将报警信息通过POST或GET请求推送到该地URL，目前仅支持HTTP协议。关于如何设置URL回调，请参见使用系统事件报警回调。
日志服务	事件报警投递到日志服务的指定日志库。

6. 单击**确定**。

相关文档

在EMR Studio配置DAG告警示例的详情，请参见给DAG配置告警。

6.1.4.1.4. 代码示例

本文通过示例为您介绍如何使用OSS Sensor、WebHDFS Sensor、Spark Operator、Hive Operator、Bash Operator和给DAG配置告警。

背景信息

本文为您介绍以下代码示例：

- 使用OSS Senser
- 使用WebHDFS Sensor

- 使用Spark Operator
- 使用Hive Operator
- 使用Bash Operator
- 使用ZeppelinOperator
- 使用JupyterOperator
- 给DAG配置告警

前提条件

- 已创建EMR Studio集群。

  创建集群详情，请参见创建集群。
- 已绑定计算集群。

> 注意
>
> - 绑定集群页签下，仅显示同一个VPC下的EMR集群。
> - 仅支持绑定Hadoop集群。

使用OSS Senser

- 使用场景：任务B需要从OSS中读取文件*data.txt*，但文件*data.txt*是通过其他程序上传的，并且不确定具体上传时间。执行任务B前需要判断OSS中是否已经存在文件*data.txt*，此时您可以使用OSS Sensor，建立任务B依赖于Sensor A的依赖关系，如果Sensor A检测到OSS中存在*data.txt*文件，则执行任务B。
- 示例：本示例中Sensor1先检测OSS中的目录*airflow/dags/Aliyun_Example_Dags/*下是否存在*zeppelin_etl_note.py*文件，如果存在，则会继续执行下一个任务Sensor2，然后检测OSS中的*airflow/dags/Aliyun_Example_Dags/*目录下是否有文件名与*zeppelin_**_not?.py*相匹配的文件，如果存在，则会继续执行最后一个任务。如果Sensor检测到相应的文件在OSS中不存在，则会按照设定的参数，每间隔一段时间检测一次，判断目标文件有没有加入到OSS中。

```
from datetime import timedelta
# The DAG object; we'll need this to instantiate a DAG
from airflow import DAG
# Operators; we need this to operate!
from airflow.operators.bash_operator import BashOperator
from airflow.utils.dates import days_ago
from airflow.contrib.sensors.aliyun_oss_sensor import AliyunOssKeySensor
# These args will get passed on to each operator
# You can override them on a per-task basis during operator initialization
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': days_ago(2),
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
    # 'queue': 'bash_queue',
    # 'pool': 'backfill',
    # 'priority_weight': 10,
    # 'end_date': datetime(2016, 1, 1),
    # 'wait_for_downstream': False,
    # 'dag': dag,
    # 'sla': timedelta(hours=2),
    # 'execution_timeout': timedelta(seconds=300),
    # 'on_failure_callback': some_function,
    # 'on_success_callback': some_other_function,
    # 'on_retry_callback': another_function,
    # 'sla_miss_callback': yet_another_function,
    # 'trigger_rule': 'all_success'
}
dag = DAG(
    'example_oss_sensor',
    default_args=default_args,
    description='a dag used to test oss sensor',
    schedule_interval=timedelta(days=1),
)
sensor1 = AliyunOssKeySensor(
    task_id='detect_target_key_1',
    # bucket_name是目标文件所在OSS的名称。
    bucket_name='chufeng-oss-ddc-hz',
    # bucket_key是目标文件在OSS中的完整路径。
    bucket_key='airflow/dags/Aliyun_Example_Dags/zeppelin_etl_note.py',
    dag=dag,
)
# wildcard_match设置为True使用通配符匹配(wildcard match)，可以在OSS Bucket中对bucket_key进行模糊搜索。
# 星号（*）匹配单个或多个字符，问号（?）匹配单个字符。
sensor2 = AliyunOssKeySensor(
    task_id='detect_target_key_2',
    bucket_name='chufeng-oss-ddc-hz',
    # 星号（*）匹配单个或多个字符，问号（?）匹配单个字符。
    bucket_key='airflow/dags/Aliyun_Example_Dags/zeppelin_**_not?.py',
    # 默认值为False，如果进行通配符匹配（wildcard match），则需要设为True。
    wildcard_match=True,
    dag=dag,
)
task = BashOperator(
    task_id='print_date',
    bash_command='date',
    dag=dag,
)
sensor1 >> sensor2 >> task
```

使用WebHDFS Sensor

- 使用场景：任务B需要从HDFS中读取文件*a.txt*，但文件*a.txt*是通过其他程序上传的，并且不确定具体上传时间。执行任务B前需要判断HDFS中是否已经存在文件*a.txt*，此时您可以使用WebHDFS Sensor，建立任务B依赖于Sensor A的依赖关系。如果Sensor A检测到HDFS中存在*a.txt*文件，则执行任务B。
- 示例：本示例中Sensor1先检测HDFS中是否存在*a.txt*文件，如果存在，则会继续执行下一个任务；如果不存在，则会按照设定的参数，每间隔一段时间检测一次，判断目标文件有没有加入到HDFS中。

```
from datetime import timedelta
# The DAG object; we'll need this to instantiate a DAG
from airflow import DAG
# Operators; we need this to operate!
from airflow.operators.bash_operator import BashOperator
from airflow.sensors.hdfs_sensor import HdfsSensor
from airflow.sensors.web_hdfs_sensor import WebHdfsSensor
from airflow.utils.dates import days_ago
# These args will get passed on to each operator
# You can override them on a per-task basis during operator initialization
# 需要关联Hadoop集群，否则可能会执行失败。
# 优先使用Sensor中指定的cluster_id，如果Sensor中未指定，则使用在DAG的default_args中指定的cluster_id。
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': days_ago(2),
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
    'cluster_id': "C-8A9CAA9E4440****",
    # 'queue': 'bash_queue',
    # 'pool': 'backfill',
    # 'priority_weight': 10,
    # 'end_date': datetime(2016, 1, 1),
    # 'wait_for_downstream': False,
    # 'dag': dag,
    # 'sla': timedelta(hours=2),
    # 'execution_timeout': timedelta(seconds=300),
    # 'on_failure_callback': some_function,
    # 'on_success_callback': some_other_function,
    # 'on_retry_callback': another_function,
    # 'sla_miss_callback': yet_another_function,
    # 'trigger_rule': 'all_success'
}
dag = DAG(
    'example_web_hdfs_sensor',
    default_args=default_args,
    description='a dag used to test webhdfs sensor',
    schedule_interval=timedelta(days=1),
)
sensor1 = WebHdfsSensor(
    # filepath
    filepath="/a.txt",
    # cluster_id="C-8A9CAA9E4440****",
    task_id='detect_target_key_1',
    dag=dag,
)
task = BashOperator(
    task_id='print_date',
    bash_command='date',
    dag=dag,
)
sensor1 >> task
```

说明 Airflow所有Sensor中都有两个默认参数poke_interval和timeout，均以秒为单位。poke_interval默认值是60（1分钟），timeout默认值是60*60*24*7（1周）。如果需要改变poke_interval，则建议设定值不要低于60，以免频繁请求造成效率下降。

使用Spark Operator

注意

- Spark Operator必须指定cluster_id。

  优先使用Operator中指定的cluster_id，如果未指定，则使用DAG中default_args参数指定的cluster_id。

- Spark Operator所需文件建议上传至挂载的OSS路径，并在代码中通过变量oss_mount_folder使用。

  Operator可能会执行在EMR Studio集群的任意worker节点上，因此需要提交的JAR或文件等，建议上传至创建集群时所设置的数据开发存储的OSS路径下，可以避免同步数据的操作。该路径的Bucket会挂载至EMR Studio集群中所有主机的*/mnt/jfs/*下。上传完成后，可以在Operator中通过oss_mount_folder使用，例如，实例代码。oss_mount_folder为EMR Studio Airflow中默认已提供的变量，值为数据开发存储的OSS路径所映射的主机的本地路径。

- 使用场景：Spark相关的两个Operator，SparkSubmitOperator用于提交JAR或Python文件，SparkSqlOperator用于提交spark-sql。
- 示例：首选需要通过spark-submit运行SparkPi样例，然后通过Spark-sql创建表并插入数据，最后使用自定义函数UDF。

```
from datetime import timedelta
from airflow.contrib.operators.spark_sql_operator import SparkSqlOperator
from airflow.contrib.operators.spark_submit_operator import SparkSubmitOperator
from airflow.models import DAG
from airflow.models import Variable
from airflow.utils.dates import days_ago
args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': days_ago(1),
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 0,
    'cluster_id': "C-8A9CAA9E4440****",
}
dag = DAG(
    dag_id="example_spark_operator",
    default_args=args,
    schedule_interval='0 0 * * *',
    dagrun_timeout=timedelta(minutes=60),
    tags=['example'],
)
oss_mount_folder = Variable.get('oss_mount_folder',"")
run_first = SparkSubmitOperator(
    task_id='run_first',
    dag=dag,
    application=oss_mount_folder + '/example/spark-examples_2.11-2.4.7.jar',
    java_class='org.apache.spark.examples.SparkPi',
    name='airflow-spark-submit',
    conf={'spark.memory.storageFraction':0.6},
    executor_cores= 1,
    executor_memory='1g',
    driver_memory='1g',
    # cluster_id="C-6394A97DE6D7****",
    # conn_id='spark_default',
    # files=None,
    # py_files=None,
    # archives=None,
    # driver_class_path=None,
    # jars=None,
    # packages=None,
    # exclude_packages=None,
    # repositories=None,
    # total_executor_cores=None,
    # keytab=None,
    # principal=None,
    # proxy_user=None,
    # num_executors=None,
    # status_poll_interval=1,
    # application_args=None,
    # env_vars=None,
    # verbose=False,
    # spark_binary=None,
    # cluster_id=None,
)
run_second = SparkSqlOperator(
    task_id='run_second',
```

```
    dag=dag,
    sql='''
    create database if not exists airflow;
    use airflow;
    drop table if exists test_sparksql;
    create table test_sparksql(name string);
    insert into test_sparksql values('studio');
    ''',
    name='airflow-spark-sql-1',
    conf='spark.sql.shuffle.partitions=200'
    # conn_id='spark_default',
    # total_executor_cores=None,
    # executor_cores=None,
    # executor_memory=None,
    # keytab=None,
    # principal=None,
    # master='yarn',
    # num_executors=None,
    # verbose=True,
    # yarn_queue='default',
    # cluster_id=None,
)
run_third = SparkSqlOperator(
    task_id='run_third',
    dag=dag,
    sql='''
    use airflow;
    add jar oss://emr-studio-example/hive-udf-1.0-SNAPSHOT.jar;
    create temporary function simpleudf AS 'com.aliyun.emr.hive.udf.SimpleUDFExample';
    show functions like '*udf';
    select simpleudf(name) from test_sparksql;
    ''',
    name='airflow-spark-sql-2',
    conf='spark.sql.shuffle.partitions=200'
)
run_first >> run_second >> run_third
```

使用Hive Operator

> 注意
>
> - Hive Operator必须指定cluster_id。
>
> 优先使用Operator中指定的cluster_id，如果未指定，则使用DAG中default_args参数指定的cluster_id。
> - Operator所需文件建议上传至挂载的OSS路径。

- 使用场景：Hive Operator可用于提交Hive SQL。
- 示例：先创建表并插入数据，然后使用自定义函数UDF。

  *example_hive_operator.py*文件详情如下。

```
from datetime import timedelta
from airflow.models import DAG
from airflow.operators.hive_operator import HiveOperator
from airflow.utils.dates import days_ago
args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': days_ago(1),
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 0,
    'cluster_id': "C-8A9CAA9E4440****",
}
dag = DAG(
    dag_id="example_hive_operator",
    default_args=args,
    schedule_interval='0 0 * * *',
    dagrun_timeout=timedelta(minutes=60),
    tags=['example'],
)
run_first = HiveOperator(
    task_id='run_first',
    dag=dag,
    hql='''
    create database if not exists airflow;
    use airflow;
    drop table if exists test_hive;
    create table test_hive(name string);
    insert into test_hive values('studio');
    ''',
    hiveconfs={'hive.execution.engine': 'tez'},
    mapred_job_name='airflow-hive-sql-1',
    # hive_cli_conn_id="hiveserver2_default",
    # cluster_id="C-8A9CAA9E4440****", 可以另指定集群，不指定时默认使用DAG的集群。
    # mapred_queue=None,
    # mapred_queue_priority=None,
    # hiveconf_jinja_translate=False,
    # script_begin_tag=None,
    # run_as_owner=False,
)
run_second = HiveOperator(
    task_id='run_second',
    dag=dag,
    hql='''
    use airflow;
    add jar oss://emr-studio-example/hive-udf-1.0-SNAPSHOT.jar;
    create temporary function simpleudf AS 'com.aliyun.emr.hive.udf.SimpleUDFExample';
    show functions like '*udf';
    select simpleudf(name) from test_hive;
    ''',
    hiveconfs={'hive.execution.engine': 'tez'},
    mapred_job_name='airflow-hive-sql-2',
)
run_first >> run_second
```

使用Bash Operator

> 注意
>
> - Bash Operator如果调用Hadoop和HDFS等命令，则必须指定cluster_id。
>
>   优先使用Operator中指定的cluster_id，如果未指定，则使用DAG中default_args参数指定的cluster_id。
> - Operator所需文件建议上传至挂载的OSS路径，并在代码中通过变量oss_mount_folder使用。
>
>   Operator可能会执行在EMR Studio集群的任意worker节点上，因此需要提交的JAR或文件等，建议上传至创建集群时所设置的数据开发存储的OSS路径下，可以避免同步数据的操作。该路径的Bucket会挂载至EMR Studio集群中所有主机的*/mnt/jfs/*下。上传完成后，可以在Operator中通过oss_mount_folder使用，例如，实例代码。oss_mount_folder为EMR Studio Airflow中默认已提供的变量，值为数据开发存储的OSS路径所映射的主机的本地路径。

- 使用场景：Bash Operator可用于提交bash命令。
- 示例：首先构造数据集上传至HDFS，然后提交MapReduce作业，最后查看HDFS上的结果。

*example_bash_operator.py*代码详情如下。

```
from builtins import range
from datetime import timedelta
from airflow.models import DAG
from airflow.models import Variable
from airflow.operators.bash_operator import BashOperator
from airflow.utils.dates import days_ago
args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': days_ago(1),
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 0,
    'cluster_id': "C-8A9CAA9E4440****",
}
dag = DAG(
    dag_id='example_bash_operator',
    default_args=args,
    schedule_interval='0 0 * * *',
    dagrun_timeout=timedelta(minutes=60),
    tags=['example'],
)
oss_mount_folder = Variable.get('oss_mount_folder')
run_first = BashOperator(
    task_id='run_first',
    bash_command=r'''
    hdfs dfs -mkdir -p /tmp/wordcount/input &&
    hdfs dfs -rm -r -f /tmp/wordcount/output &&
    echo -e "1201\tGopal\t45000\tTechnical manager
1202\tManisha\t45000\tProof reader
1203\tMasthanvali\t40000\tTechnical writer
1204\tKiran\t40000\tHr Admin
1205\tKranthi\t30000\tOp Admin" > /tmp/sample.txt &&
    hdfs dfs -put -f /tmp/sample.txt /tmp/wordcount/input
    ''',
    dag=dag,
)
run_second = BashOperator(
    bash_command='hadoop jar {0}/example/hadoop-mapreduce-examples-3.2.1.jar wordcount /tmp/wordcount/input /tmp/wordcoun
t/output'.format(oss_mount_folder),
    task_id='run_second',
    dag=dag,
)
run_third = BashOperator(
    bash_command='hdfs dfs -cat /tmp/wordcount/output/part-r-00000',
    task_id='run_third',
    dag=dag,
)
run_first >> run_second >> run_third
```

给DAG配置告警

告警示例如下：

- DAG失败告警示例（dag_fail_apm）

  根据DAG失败告警级别的不同从airflow.contrib.operators.aliyun_apm_operator导入不同的callback告警函数，并传给DAG中的on_failure_callback函数。对于严重的失败事件，采用函数apm_alert_callback_dagrun_fail_critical。对于不严重的、警告类型的失败事件，采用函数apm_alert_callback_dagrun_fail_warning。

```
from airflow import DAG
from datetime import datetime, timedelta
from airflow.utils.dates import days_ago
from airflow.operators.bash_operator import BashOperator
# 导入告警callback函数。
from airflow.contrib.operators.aliyun_apm_operator import apm_alert_callback_dagrun_fail_critical
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': days_ago(2),
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 0,
    'retry_delay': timedelta(minutes=5),
    # 将告警callback函数传入DAG。
    'on_failure_callback': apm_alert_callback_dagrun_fail_critical,
}
with DAG('dag_fail_apm',
         max_active_runs=5,
         schedule_interval='0 0 * * *',
         default_args=default_args) as dag:
    intentionally_failed_task = BashOperator(
        task_id='intentionally_failed_task',
        bash_command='echooooooooo make a mistake intentionally'
    )
    intentionally_failed_task
```

- DAG SLA告警示例（dag_sla_breach_apm）

  根据DAG SLA触发告警级别的不同从airflow.contrib.operators.aliyun_apm_operator导入不同的callback告警函数，并传给DAG中的sla_miss_callback函数。对于严重的SLA触发事件，采用apm_alert_callback_miss_sla_critical；对于不严重的、警告类型的SLA触发事件，采用apm_alert_callback_miss_sla_warning。

```
from datetime import timedelta
# 导入SLA触发callback告警函数。
from airflow.contrib.operators.aliyun_apm_operator import apm_alert_callback_miss_sla_warning
# The DAG object; we'll need this to instantiate a DAG
from airflow import DAG
# Operators; we need this to operate!
from airflow.operators.bash_operator import BashOperator
from airflow.utils.dates import days_ago
# These args will get passed on to each operator
# You can override them on a per-task basis during operator initialization
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': days_ago(3),
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retry_delay': timedelta(seconds=20)
    # 'queue': 'bash_queue',
    # 'pool': 'backfill',
    # 'priority_weight': 10,
    # 'end_date': datetime(2016, 1, 1),
    # 'wait_for_downstream': False,
    # 'dag': dag,
    # 'sla': timedelta(hours=2),
    # 'execution_timeout': timedelta(seconds=300),
    # 'on_failure_callback': some_function,
    # 'on_success_callback': some_other_function,
    # 'on_retry_callback': another_function,
    # 'trigger_rule': 'all_success'
}
dag = DAG(
    'dag_sla_breach_apm',
    default_args=default_args,
    description='breach sla intentionally',
    schedule_interval=timedelta(days=1),
    sla_miss_callback=apm_alert_callback_miss_sla_warning
)
t2 = BashOperator(
    task_id='oversleep',
    depends_on_past=False,
    bash_command='sleep 10',
    # SLA设定1s，当命令sleep 10s时，必定触发SLA。
    sla=timedelta(seconds=1),
    dag=dag
)
t2
```

6.1.4.2. 最佳实践

6.1.4.2.1. 定期调度Zeppelin中的作业

当您在Zeppelin里完成作业开发后，您就可以通过Airflow定期调度作业。本文为您介绍如何使用Airflow调度作业。

前提条件

- 已创建EMR Studio集群。

  创建集群详情，请参见创建集群。
- 安全组规则已开启8000、8081和8443端口。

  添加安全组规则，详情请参见添加安全组规则。

背景信息

ZeppelinOperator支持以下两种调度方式：

- Note级别调度

  一个Note可以作为Airflow的一个单独Task来调度，多个Note可以组合成一个复杂的DAG。
- Paragraph级别调度

一个段落（Paragraph）可以作为Airflow的一个单独Task来调度，多个Paragraph可以组合成一个复杂的DAG。

ZeppelinOperator需要指定以下参数：

- task_id：必填参数，需要保证在DAG范围内唯一。
- conn_id：必填参数，默认值为zeppelin_default，表示连接的Zeppelin的地址信息。
- note_id：必填参数，您要运行的Note的ID。

  您可以在Note页面右上角找到这个Note的ID。
- paragraph_id：可选参数，当您运行整个Note时，不需要指定paragraph_id。当您运行单个Paragraph时，需要指定paragraph_id。

  您可以在Paragraph页面右上角找到这个Paragraph的ID。
- params：可选参数，当您的Note或Paragraph需要传递参数时，需要指定params参数。

Airflow支持以下方式传递参数：

- 通过Jinja Templating传递参数。

  Jinja Templating详情，请参见Concepts。
- 通过宏的方式来传递参数。

  常用的变量可以通过宏的方式来传递参数，例如，本文示例中的`{{ ds }}`就是传递执行日期，关于Airflow支持的宏，请参见Macros reference。

ZeppelinOperator两种调度方式示例：

- Note级别调度

  本示例总共有3个Task，分别为：

  i. **生成数据**：对应示例中的raw_data_task。
  ii. **把CSV格式转为Parquet格式**：对应示例中的spark_etl_task。
  iii. **查询Parquet表**：对应示例中的spark_query_task。

  每个Task都写在单独的Note里，因此您只需要在每个ZeppelinOperator里指定note_id。

```
from airflow import DAG
from datetime import datetime, timedelta
from airflow.contrib.operators.zeppelin_operator import ZeppelinOperator
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2018, 1, 1),
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 0,
    'retry_delay': timedelta(minutes=5),
}
with DAG('zeppelin_etl_note',
         max_active_runs=5,
         schedule_interval='0 0 * * *',
         default_args=default_args) as dag:
    execution_date = "{{ ds }}"
    raw_data_task = ZeppelinOperator(
        task_id='raw_data_task',
        conn_id='zeppelin_default',
        note_id='2FZWJTTPS',
        params= {'dt' : execution_date}
    )
    spark_etl_task = ZeppelinOperator(
        task_id='spark_etl_task',
        conn_id='zeppelin_default',
        note_id='2FX3GJW67',
        params= {'dt' : execution_date}
    )
    spark_query_task = ZeppelinOperator(
        task_id='spark_query_task',
        conn_id='zeppelin_default',
        note_id='2FZ8H4JPV',
        params= {'dt' : execution_date}
    )
    raw_data_task >> spark_etl_task >> spark_query_task
```

- Paragraph级别调度

本示例总共有4个Task，分别为：

i. **生成数据**：对应示例中的raw_data_task。
ii. **Load csv数据到Hive表**：对应示例中的csv_data_task。
iii. **把CSV格式转为Parquet格式**：对应示例中的parquet_data_task。
iv. **查询Parquet表**：对应示例中的query_ta。

每个Task都是一个Paragraph，因此您需要指定note_id和paragraph_id。本示例中还指定了参数params，把当前执行时间传递进去，用到了宏 `{{ ds }}` 来表示运行时间YYYY-MM-DD。

```
from airflow import DAG
from datetime import datetime, timedelta
from airflow.contrib.operators.zeppelin_operator import ZeppelinOperator
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2018, 1, 1),
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 0,
    'retry_delay': timedelta(minutes=5),
}
with DAG('zeppelin_example_paragraph_dag',
         max_active_runs=5,
         schedule_interval='0 0 * * *',
         default_args=default_args) as dag:
    execution_date = "{{ ds }}"
    raw_data_task = ZeppelinOperator(
        task_id='raw_data_task',
        conn_id='zeppelin_default',
        note_id='2FXADMCY9',
        paragraph_id='paragraph_1613703810025_1489985581',
        params= {'dt' : execution_date}
    )
    csv_data_task = ZeppelinOperator(
        task_id='csv_data_task',
        conn_id='zeppelin_default',
        note_id='2FXADMCY9',
        paragraph_id='paragraph_1613705325984_828524031',
        params= {'dt' : execution_date}
    )
    parquet_data_task = ZeppelinOperator(
        task_id='parquet_data_task',
        conn_id='zeppelin_default',
        note_id='2FXADMCY9',
        paragraph_id='paragraph_1613709950691_1420422787',
        params= {'dt' : execution_date}
    )
    query_task = ZeppelinOperator(
        task_id='query_task',
        conn_id='zeppelin_default',
        note_id='2FXADMCY9',
        paragraph_id='paragraph_1613703819960_454383326',
        params= {'dt' : execution_date}
    )
    raw_data_task >> csv_data_task >> parquet_data_task >> query_ta
```

操作流程

1. 步骤一：编写Airflow DAG脚本
2. 步骤二：上传Airflow DAG脚本
3. 步骤三：启用DAG脚本
4. 步骤四：查看DAG运行状态

步骤一：编写Airflow DAG脚本

EMR Studio集群暂不支持在线编写Airflow DAG脚本，您需要在本地编写Airflow DAG脚本。在Airflow里构建DAG的详情，请参见Airflow官网。

EMR Studio集群新增了ZeppelinOperator，您可以使用ZeppelinOperator调用Zeppelin Notebook。使用ZeppelinOperator可以最大程度地保证开发环境和生产环境的一致性，避免由于开发阶段和生产阶段环境不一致导致的问题。ZeppelinOperator使用的详细信息请参见背景信息。

步骤二：上传Airflow DAG脚本

本文通过OSS控制台上传Airflow DAG脚本。上传Airflow DAG脚本的路径为您创建集群时指定的OSS路径。

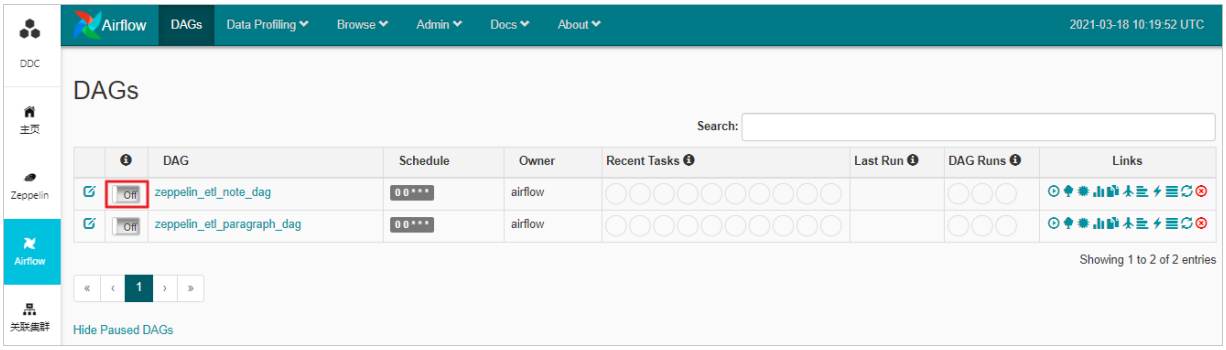
1. 登录 OSS管理控制台。
2. 上传Airflow DAG脚本，详情请参见上传文件。

   上传您在步骤一：编写Airflow DAG脚本中编写的Airflow DAG脚本。

   例如，如果您创建集群时设置的路径为*oss://bucket_1/your_folder*，则您的Airflow DAG脚本存放的路径就是*oss://bucket_1/your_folder/airflow/dags*。

步骤三：启用DAG脚本

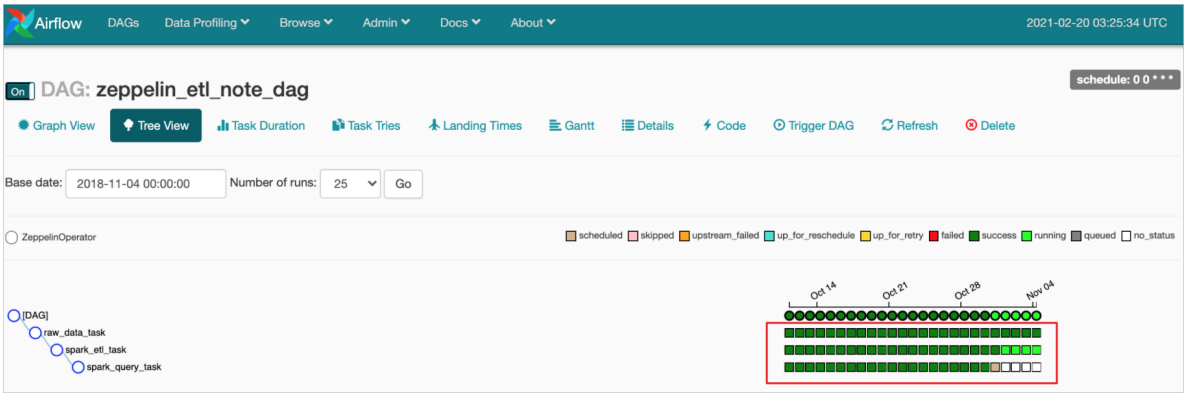
在**DAGs**页面，打开待启用DAG脚本所在行的**off**开关，即可启用DAG调度。



> **说明** 上传Airflow DAG脚本至OSS后，在DAGs页面并不能立刻看到上传的DAG，大约需要1~2分钟，请您耐心等待。

步骤四：查看DAG运行状态

1. 在**DAGs**页面，单击待查看调度任务的DAG。
2. 在**Tree View**页面，您可以查看对应DAG运行调度任务的详细信息。



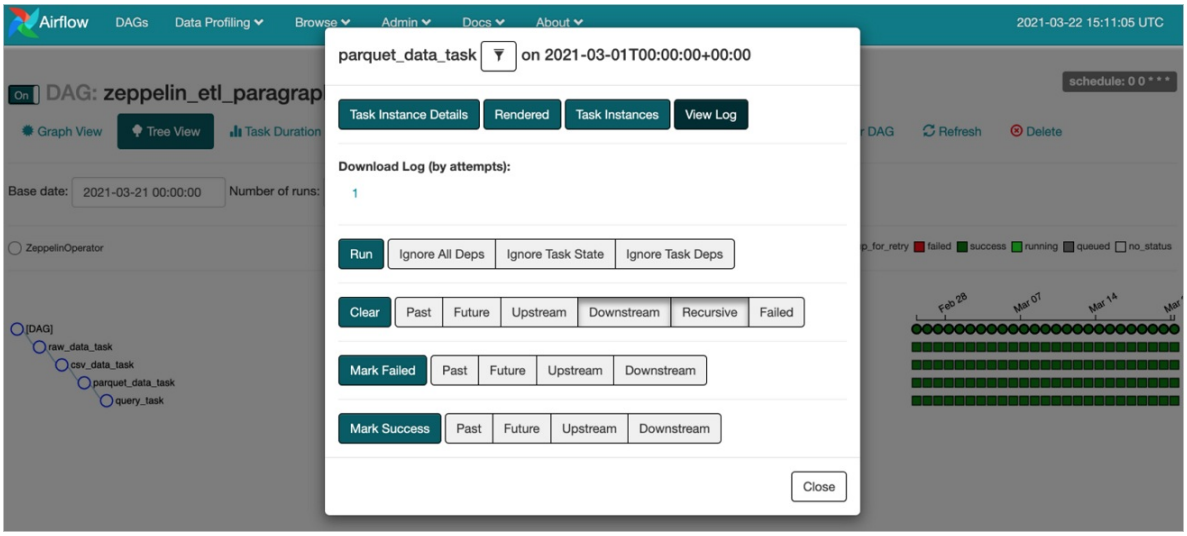
任务状态如下表所示。

状态	描述
scheduled	已开始调度
skipped	跳过本次调度
upstream_failed	上游任务失败
up_for_reschedule	已重新调度
up_for_retry	重试
failed	调度失败
success	调度成功
running	调度正在运行中

状态	描述
queued	加入调度队列
no_status	没有状态

3. 在Tree View页面，单击 图标。

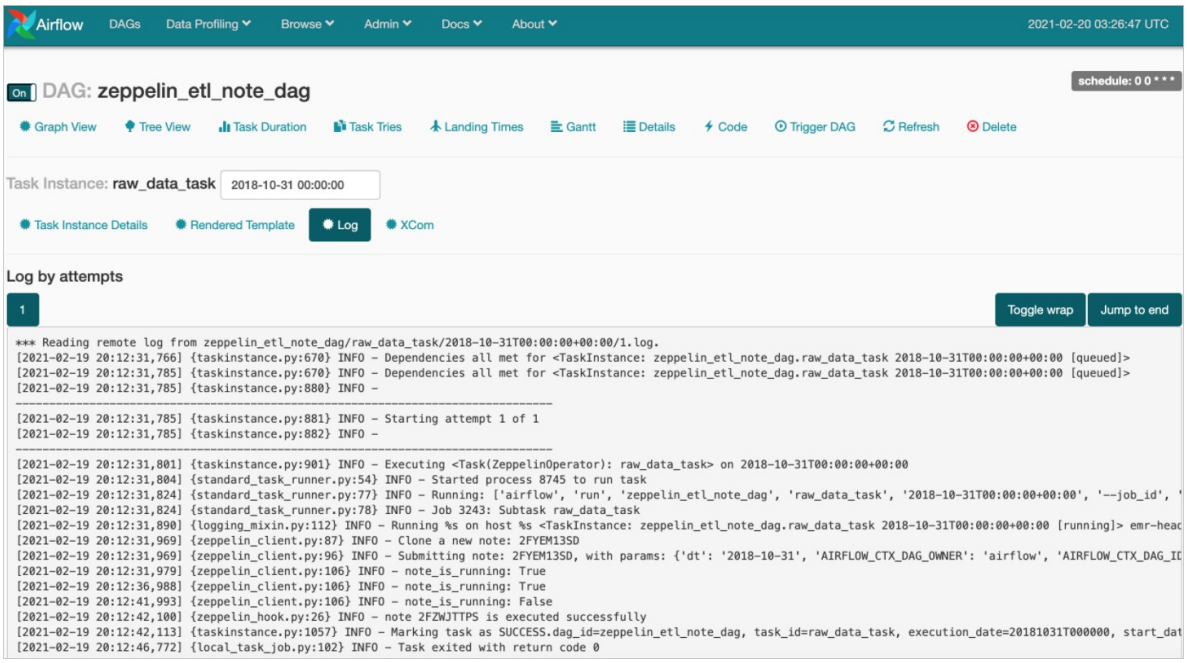
   弹出相应Task的对话框。



4. 单击Task对话框中的**View log**。

   即可查看Task详情的运行情况。

   详细信息如下图所示。



6.1.4.2.2. 定期调度Jupyter中的作业

当您在Jupyter里完成作业开发后，您就可以通过Airflow定期调度作业。本文为您介绍如何将Jupyter中编写的Notebook，生成调度任务并定时执行。

前提条件

- 已创建EMR Studio集群。

创建集群详情，请参见创建集群。

- 安全组规则已开启8000、8081和8443端口。

  添加安全组规则，详情请参见添加安全组规则。

- 已绑定计算集群。

  > 注意
  > - 绑定集群页签下，仅显示同一个VPC下的EMR集群。
  > - 仅支持绑定Hadoop集群。

- 已添加用户，详情请参见添加用户。
- 已完成Notebook的编辑。您可以直接下载Notebook示例文件airflow_nb_example.ipynb至本地，然后上传至您的Jupyter中查看。

操作流程

1. 步骤一：编写Airflow DAG脚本
2. 步骤二：上传Airflow DAG脚本
3. 步骤三：启用DAG脚本

步骤一：编写Airflow DAG脚本

示例代码如下。

```
from airflow import DAG
from datetime import datetime, timedelta
from airflow.contrib.operators.zeppelin_operator import ZeppelinOperator
from jupyter_operator import JupyterNotebookEmrOperator
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2021, 8, 1), # 第一次开始执行的时间。
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 0, # 失败重试次数。
    'retry_delay': timedelta(minutes=5), # 失败重试间隔。
}
with DAG('jupyter_airflow_example',
         max_active_runs=1,
         schedule_interval='0 0 * * *', # 执行周期，依次是分、时、周、月和年，此处表示每天0：00执行。
         default_args=default_args) as dag:
    execution_date = "{{ ds }}"
    op = JupyterNotebookEmrOperator(
        task_id='<yourTaskId>',
        vcores=<yourVcore>,
        memory='<yourMemory>',
        job_name='<yourJobname>',
        cluster_id='<yourClusterIP>',
        notebook_path='<yourNotebookpath>',
        username='<yourName>',
        venv='<yourVENV>',
        env={'VENV': '<yourENV>', 'SPARK_HOME': '/usr/lib/spark-current', 'HIVE_CONF_DIR': '/etc/ecm/hive-conf', 'PYSPARK_P
YTHON': './env/bin/python'},
        params={'<yourparams>': execution_date},
    )
```

其中，涉及的参数信息如下。

参数	描述
<yourTaskId>	Operator通用Task ID，在DAG文件内唯一，字符串类型。例如，jupyter_task。
<yourVcore>	指定运行容器的虚拟核数，整数类型。例如，1。
<yourMemory>	指定运行容器的内存，字符串类型。例如，4 G。
<yourJobname>	YARN作业名，字符串类型。例如，papermill。
<yourClusterIP>	您关联的Hadoop集群的ID。

参数	描述
<yourName>	登录JupyterHub的用户名。
<yourNotebookpath>	Notebook在OSS中的存储路径，只需填写相对路径即可。本文示例为*spark_magic_example.ipynb*。
<yourVENV>	运行Jupyter时选择的虚拟环境，只需要填写虚拟环境名称即可，不带*.tar.gz*文件名后缀，字符串类型。例如，*emr-jupyterenv*。
<yourENV>	指定容器环境变量，dict类型。
<yourparams>	Notebook参数，需要在Notebook的单元格上添加parameters标签。 标签详情，请参见Parameterize。

步骤二：上传Airflow DAG脚本

本文通过OSS控制台上传Airflow DAG脚本。上传Airflow DAG脚本的路径为您创建集群时指定的OSS路径。

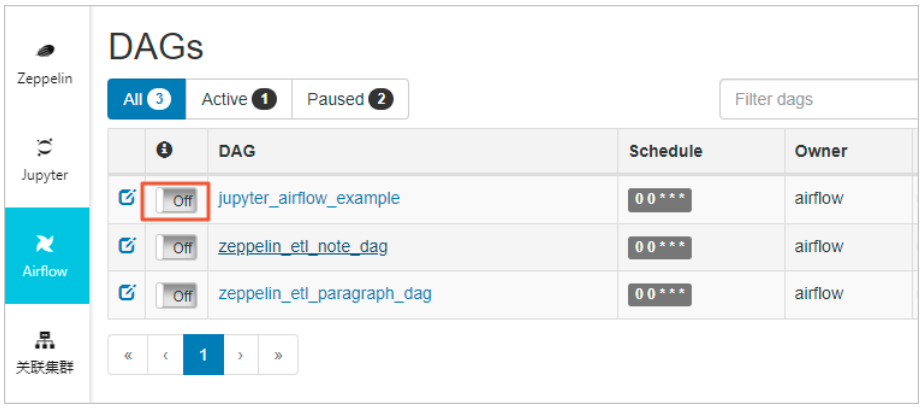
1. 登录 OSS管理控制台。
2. 上传Airflow DAG脚本，详情请参见上传文件。

   上传您在步骤一：编写Airflow DAG脚本中编写的Airflow DAG脚本。

   上传Airflow DAG脚本的路径与您创建集群时设置的路径有关。例如，如果您创建集群时设置的路径为*oss://bucket_1/your_folder*，则您的Airflow DAG脚本存放的路径就是*oss://bucket_1/your_folder/airflow/dags*。

步骤三：启用DAG脚本

在**DAGs**页面，打开待启用DAG脚本所在行的**off**开关，即启用了调度任务。



> **说明** 上传Airflow DAG脚本至OSS后，在DAGs页面并不能立刻看到调度任务，大约需要1~2分钟，请您耐心等待。

6.1.4.2.3. 动态启动计算集群运行工作流调度

本文为您介绍如何通过EMR集群的模板功能为EMR Studio动态拉起计算集群来运行工作流，该计算集群会在调度完成后自动释放。

前提条件

- 已创建EMR Studio集群。

  创建集群详情，请参见创建集群。
- 安全组规则已开启8000、8081和8443端口。

  添加安全组规则，详情请参见添加安全组规则。

操作步骤

1. 创建EMR集群模板，详情请参见创建集群模板。
2. 编写Airflow DAG脚本。

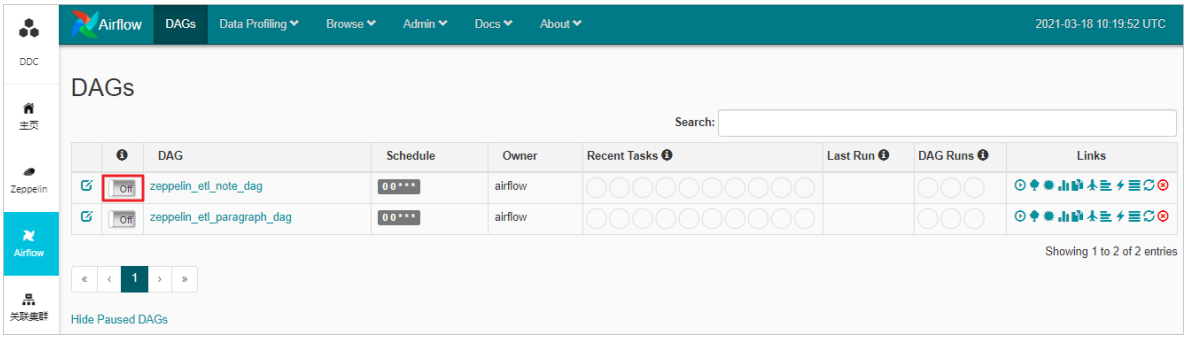
   配置zeppelin_etl_note_dag脚本，示例代码如下。

```
from airflow import DAG
from datetime import datetime, timedelta
from airflow.contrib.operators.zeppelin_operator import ZeppelinOperator
from airflow.utils.dates import days_ago
from airflow.contrib.operators.aliyun_emr_operator import TearDownClusterOperator
from airflow.contrib.operators.aliyun_emr_operator import CreateClusterFromTemplateOperator
default_args = {
        'owner': 'airflow',
        'depends_on_past': False,
        'start_date': datetime(2019, 4, 9),
        'email_on_failure': False,
        'email_on_retry': False,
        'retries': 0,
        'retry_delay': timedelta(minutes=500),
}
with DAG('zeppelin_etl_note_dag_new',
                 max_active_runs=5,
                 schedule_interval='0 0 * * *',
                 default_args=default_args) as dag:
    execution_date = "{{ ds }}"
    head = CreateClusterFromTemplateOperator(
        task_id='create_cluster',
        template_id ='CT-37377AD06E3B****',
    )
   raw_data_task = ZeppelinOperator(
        task_id='raw_data_task',
        conn_id='zeppelin_default',
        note_id='2FZWJTTPS',
        create_cluster_task_id='create_cluster',
        params= {'dt' : execution_date}
        )
    spark_etl_task = ZeppelinOperator(
        task_id='spark_etl_task',
        conn_id='zeppelin_default',
        note_id='2FX3GJW67',
        create_cluster_task_id='create_cluster',
        params= {'dt' : execution_date}
    )
    spark_query_task = ZeppelinOperator(
        task_id='spark_query_task',
        conn_id='zeppelin_default',
        note_id='2FZ8H4JPV',
        create_cluster_task_id='create_cluster',
        params= {'dt' : execution_date}
    )
    tail = TearDownClusterOperator(
        task_id='tear_down_cluster',
        create_cluster_task_id='create_cluster',
    )
    head >> raw_data_task >> spark_etl_task >> spark_query_task >> tail
```

> **说明** template_id的值为您步骤1中创建模板集群的ID，并且ZeppelinOperator和TearDownClusterOperator中的create_cluster_task_id需要和CreateClusterFromTemplateOperator中的task_id保持一致。

3. 上传Airflow DAG脚本。
   i. 登录 OSS管理控制台。
   ii. 上传Airflow DAG脚本，详情请参见上传文件。
4. 启用DAG脚本。

在**DAGs**页面，打开待启用DAG脚本所在行的**off**开关，即可启用DAG调度。



说明 上传Airflow DAG脚本至OSS后，在DAGs页面并不能立刻看到上传的DAG，大约需要1~2分钟，请您耐心等待。

5. 查看DAG运行状态。
   i. 在**DAGs**页面，单击待查看调度任务的DAG。
   ii. 在**Tree View**页面，您可以查看对应DAG运行调度任务的详细信息。
   iii. 在**Tree View**页面，单击 图标。
   iv. 单击Task对话框中的**View log**。
   即可查看Task详情的运行情况。

```
[2021-04-28 11:55:07,924] {aliyun_emr_operator.py:114} INFO - cluster C-A813BA399[illegible] is created
[2021-04-28 11:55:08,386] {aliyun_emr_operator.py:121} INFO - bind cluster response:{'status': 'OK', 'successResponse': True}
[2021-04-28 11:55:18,401] {aliyun_emr_operator.py:135} INFO - binding status: PROCESSING
[2021-04-28 11:55:28,416] {aliyun_emr_operator.py:135} INFO - binding status: PROCESSING
[2021-04-28 11:55:38,432] {aliyun_emr_operator.py:135} INFO - binding status: PROCESSING
[2021-04-28 11:55:48,447] {aliyun_emr_operator.py:135} INFO - binding status: PROCESSING
[2021-04-28 11:55:58,672] {aliyun_emr_operator.py:135} INFO - binding status: PROCESSING
[2021-04-28 11:56:08,687] {aliyun_emr_operator.py:135} INFO - binding status: PROCESSING
[2021-04-28 11:56:18,704] {aliyun_emr_operator.py:135} INFO - binding status: PROCESSING
[2021-04-28 11:56:28,719] {aliyun_emr_operator.py:135} INFO - binding status: PROCESSING
[2021-04-28 11:56:38,735] {aliyun_emr_operator.py:135} INFO - binding status: PROCESSING
[2021-04-28 11:56:48,751] {aliyun_emr_operator.py:135} INFO - binding status: PROCESSING
[2021-04-28 11:56:58,767] {aliyun_emr_operator.py:135} INFO - binding status: PROCESSING
[2021-04-28 11:57:08,782] {aliyun_emr_operator.py:135} INFO - binding status: PROCESSING
[2021-04-28 11:57:18,798] {aliyun_emr_operator.py:135} INFO - binding status: PROCESSING
[2021-04-28 11:57:28,833] {taskinstance.py:1057} INFO - Marking task as SUCCESS.dag_id=zeppelin_etl_note_dag_new, task_id=create_cluster_dl, execution_date=20190605T000000, start_date
[2021-04-28 11:57:30,677] {local_task_job.py:102} INFO - Task exited with return code 0
```

说明 通过Log信息您可以看到已关联的集群信息，调度完成后集群会被释放掉。

6.1.4.3. 常见问题

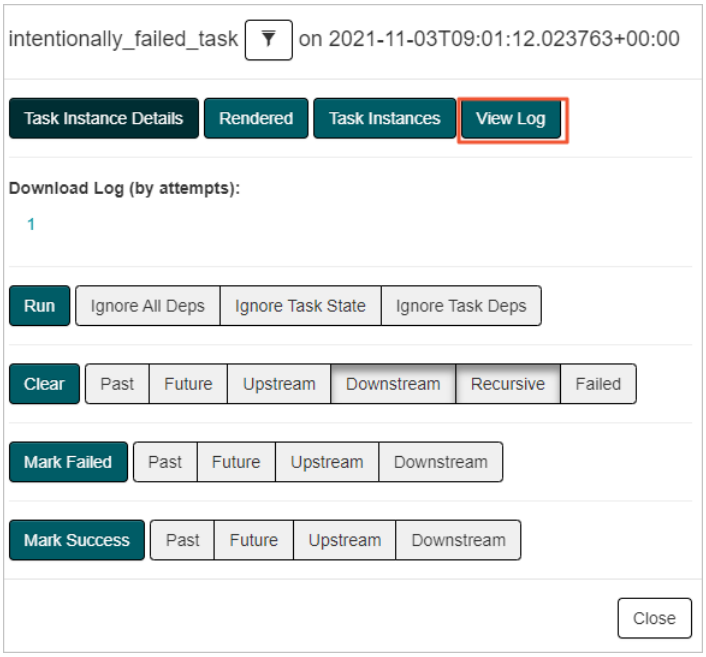
本文汇总了Airflow使用时的常见问题。

- 如何排查调度失败的任务？
- 如何修改Airflow的配置？
- 如何手动重启失败的DAG？
- 如何在DAG中使用集群模板？
- 如何确定更新的DAG已生效？

如何排查调度失败的任务？

您可以按照以下步骤排查：

1. 在Airflow的**DAGs**页面，单击失败的DAG。
2. 在**Tree View**页面，单击失败任务的 图标。
   弹出相应Task的对话框。
3. 单击Task对话框中的**View log**。



您可以在Log页面，根据失败作业的日志信息处理。

如何修改Airflow的配置？

您可以在E-MapReduce控制台的Airflow服务页面，修改配置项，详细信息请参见管理组件参数。

如何手动重启失败的DAG？

您可以按照以下步骤操作：

1. 在Airflow的**DAGs**页面，单击目标调度任务的DAG。
2. 在**Tree View**页面，单击 图标。

   弹出相应Task的对话框。
3. 单击Task对话框中的**Clear**。

4. 单击**OK**。

   Scheduler会当作此任务未运行过，然后触发此任务重新运行。

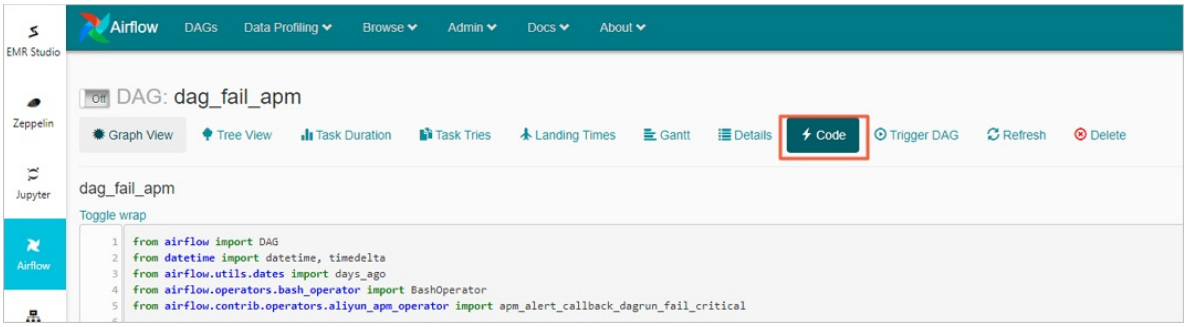
如何在DAG中使用集群模板？

详细操作，请参见动态启动计算集群运行工作流调度。

如何确定更新的DAG已生效？

无论是上传或者修改DAG后，都需要等待一段时间才能生效。每隔一段时间Airflow会扫描并更新DAG。您可以通过以下操作查看：

1. 在Airflow的**DAGs**页面，单击目标DAG。
2. 单击上方的**Code**。



可以在此页面查看代码是否已更新，已更新则说明更新已生效。如果超过五分钟还未更新，请技术支持。

6.1.5. Zeppelin

6.1.5.1. Zeppelin概述

本文介绍阿里云E-MapReduce如何访问Zeppelin。您可以通过访问Zeppelin，进行大数据可视化分析。

前提条件

- 已创建集群，并选择了Zeppelin服务，详情请参见创建集群。
- 在集群安全组中打开8080端口，详情请参见访问链接与端口。
- 已添加本文示例所需的服务，例如，Presto、Flink和Impala。

  添加服务详情请参见添加服务。

访问Zeppelin

1. 进入集群详情页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
2. 在左侧导航栏，单击**访问链接与端口**。
3. 单击Zeppelin所在行的链接。

   您可以直接访问Web UI页面。

示例

以下内容只适用于EMR-3.33.0及之后版本和EMR-4.6.0及之后版本：

- 如何使用Spark
- 如何使用Flink
- 如何使用Presto
- 如何使用Impala
- 如何使用Hive

如何使用Spark

1. 在Zeppelin页面，单击**Create new note**。
2. 在**Create new note**对话框，输入**Note Name**，选择**Default Interpreter**为**spark**。
3. 单击**create**。
4. 在Zeppelin的Notebook页面，您可以执行以下命令。

   您无需配置，EMR里的Zeppelin中已经配置了Spark Interpreter。Spark默认执行模式是Yarn-cluster。支持以下三种代码方式：
   - Spark Scala

     `%spark` 表示执行Spark Scala代码。

```
%spark
val df = spark.read.options(Map("inferSchema"->"true","delimiter"->";","header"->"true"))
.csv("file:///usr/lib/spark-current/examples/src/main/resources/people.csv")
z.show(df)
df.registerTempTable("people")
```

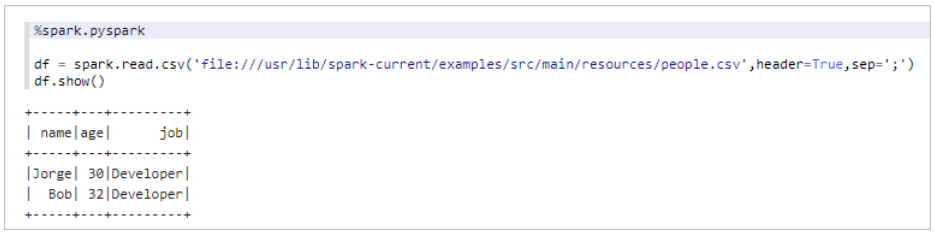
返回信息如下所示。

- PySpark

  `%spark.pyspark` 表示执行PySpark代码。

```
%spark.pyspark
df = spark.read.csv('file:///usr/lib/spark-current/examples/src/main/resources/people.csv',header=True,sep=';')
df.show()
```

返回信息如下所示。

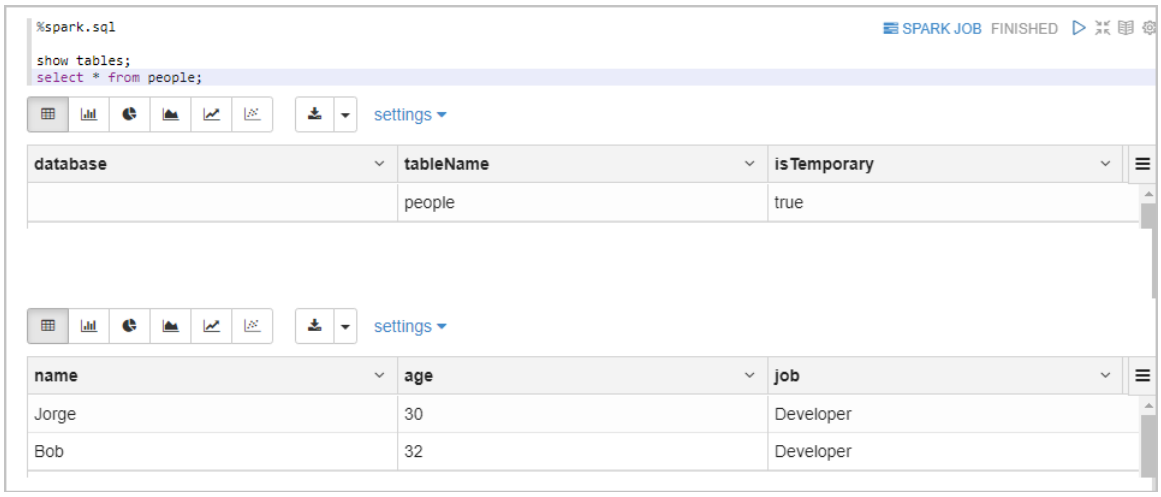


- Spark SQL

  `%spark.sql` 表示执行Spark SQL代码。

```
%spark.sql
show tables;
select * from people;
```

返回信息如下所示。



如何使用Flink

1. 在Zeppelin页面，单击**Create new note**。
2. 在**Create new note**对话框，输入**Note Name**，选择**Default Interpreter**为**flink**。
3. 单击**create**。
4. 在Zeppelin的Notebook页面，您可以执行以下命令。

   您无需配置，EMR里的Zeppelin已经为您配置了Flink Interpreter。支持以下三种代码方式：

- Flink Scala

  `%flink` 表示执行Flink Scala代码。

```
%flink
val data = benv.fromElements("hello world","hello flink","hello hadoop")
data.flatMap(line => line.split("\\s"))
                    .map(w => (w,1))
                    .groupBy(0)
                    .sum(1)
                    .print()
```

  返回信息如下所示。

```
%flink

val data = benv.fromElements("hello world","hello flink","hello hadoop")
data.flatMap(line => line.split("\\s"))
                    .map(w => (w,1))
                    .groupBy(0)
                    .sum(1)
                    .print()

data: org.apache.flink.api.scala.DataSet[String] = org.apache.flink.api.scala.DataSet@4dbe3bf4
(flink,1)
(hadoop,1)
(hello,3)
(world,1)
```

- PyFlink

  `%flink.pyflink` 表示执行PyFlink代码。

PyFlink Table Api

```
%flink.ipyflink

t = st_env.from_path("cdn_access_log")\
   .select("uuid, "
           "ip_to_country(client_ip) as country, "
           "response_size, request_time")\
  .group_by("country")\
  .select(
           "country, count(uuid) as access_count, "
           "sum(response_size) as total_download,  "
           "sum(response_size) * 1.0 / sum(request_time) as download_speed")
   #.insert_into("cdn_access_statistic")

# z.show will display the result in zeppelin front end in table format, you can uncomment the above insert statement and
    the below st_env.execute in order to write the data to mysql table.
z.show(t, stream_type="update")
#st_env.execute("pyflink_cdn_access_job")
```

- Flink SQL

  `%flink.ssql` 表示执行Flink SQL代码。

  在运行下面的示例前，需要先运行下面的代码以构建一个模拟用户日志的数据。

```
%flink
import org.apache.flink.streaming.api.functions.source.SourceFunction
import org.apache.flink.table.api.TableEnvironment
import org.apache.flink.streaming.api.TimeCharacteristic
import org.apache.flink.streaming.api.checkpoint.ListCheckpointed
import org.apache.flink.runtime.state.filesystem.FsStateBackend
import java.util.Collections
import scala.collection.JavaConversions._
senv.setStreamTimeCharacteristic(TimeCharacteristic.EventTime)
senv.enableCheckpointing(5000)
senv.setStateBackend(new FsStateBackend("file:///tmp/flink/checkpoints"));
val data = senv.addSource(new SourceFunction[(Long, String)] with ListCheckpointed[java.lang.Long] {
  val pages = Seq("home", "search", "search", "product", "product", "product")
  var count: Long = 0
  var running : Boolean = true
  // startTime is 2020/1/1
  var startTime: Long = new java.util.Date(2020 - 1900,0,1).getTime
  var sleepInterval = 100
  override def run(ctx: SourceFunction.SourceContext[(Long, String)]): Unit = {
    val lock = ctx.getCheckpointLock
    while (count < 1000000 && running) {
      lock.synchronized({
        ctx.collect((startTime + count * sleepInterval, pages(count.toInt % pages.size)))
        count += 1
        Thread.sleep(sleepInterval)
      })
    }
  }
  override def cancel(): Unit = {
    running = false
  }
  override def snapshotState(checkpointId: Long, timestamp: Long): java.util.List[java.lang.Long] = {
    Collections.singletonList(count)
  }
  override def restoreState(state: java.util.List[java.lang.Long]): Unit = {
    state.foreach(s => count = s)
  }
}).assignAscendingTimestamps(_._1)
stenv.registerDataStream("log", data, 'time, 'url, 'rowtime.rowtime)
```

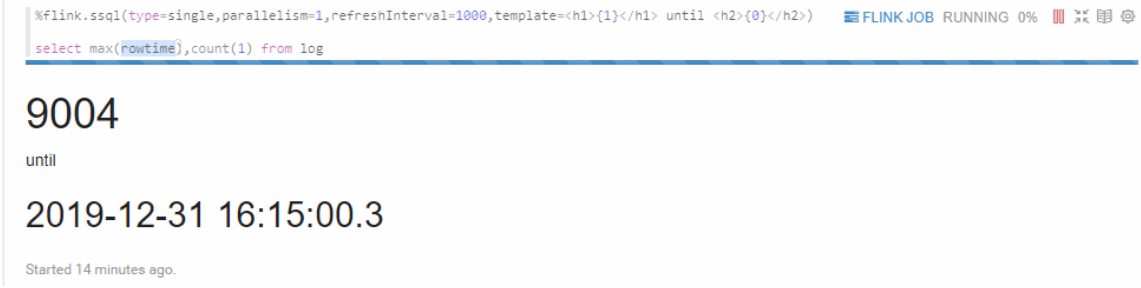
Zeppelin支持所有类型的Flink SQL语句，包括DDL和DML等。您还可以在Zeppelin可视化流式数据，Flink支持以下三种流式数据的可视化：

- Single模式

  Single模式适合当输出结果是一行的情况，不适用图形化的方式展现。例如下面的 `Select` 语句。这条SQL语句只有一行数据，但这行数据会持续不断的更新。这种模式的数据输出格式是HTML形式，您可以用 `template` 来指定输出模板， `{i}` 是第i列的placeholder。

  ```
  %flink.ssql(type=single,parallelism=1,refreshInterval=1000,template=<h1>{1}</h1> until <h2>{0}</h2>)
  select max(rowtime),count(1) from log
  ```

  返回信息如下所示。

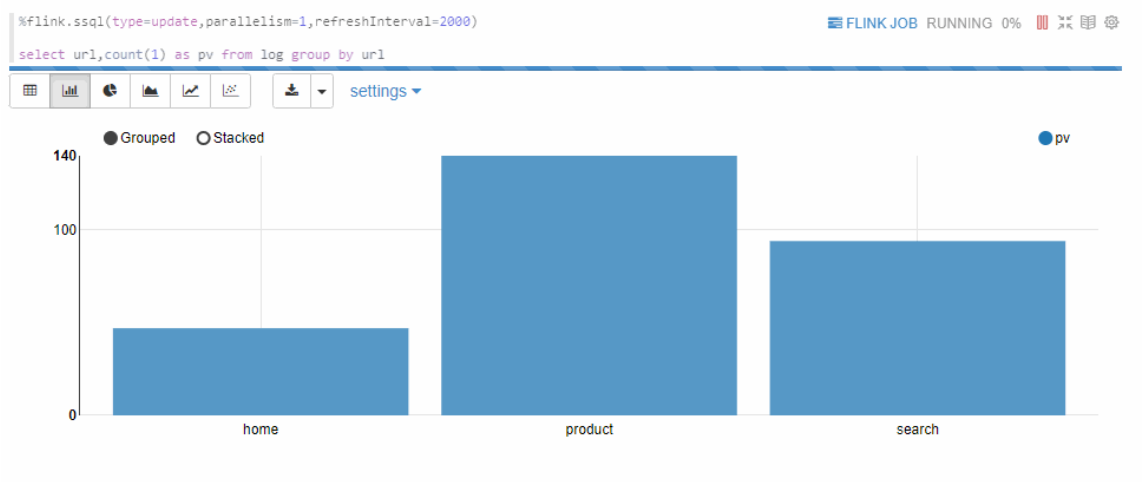
  

- Update模式

  Update模式适合多行输出的情况。例如下面的 `select group by` 语句。此模式会定期更新这多行数据，输出是Zeppelin的table格式，因此可以用Zeppelin自带的可视化控件。

```
%flink.ssql(type=update,parallelism=1,refreshInterval=2000)
select url,count(1) as pv from log group by url
```

  返回信息如下所示。

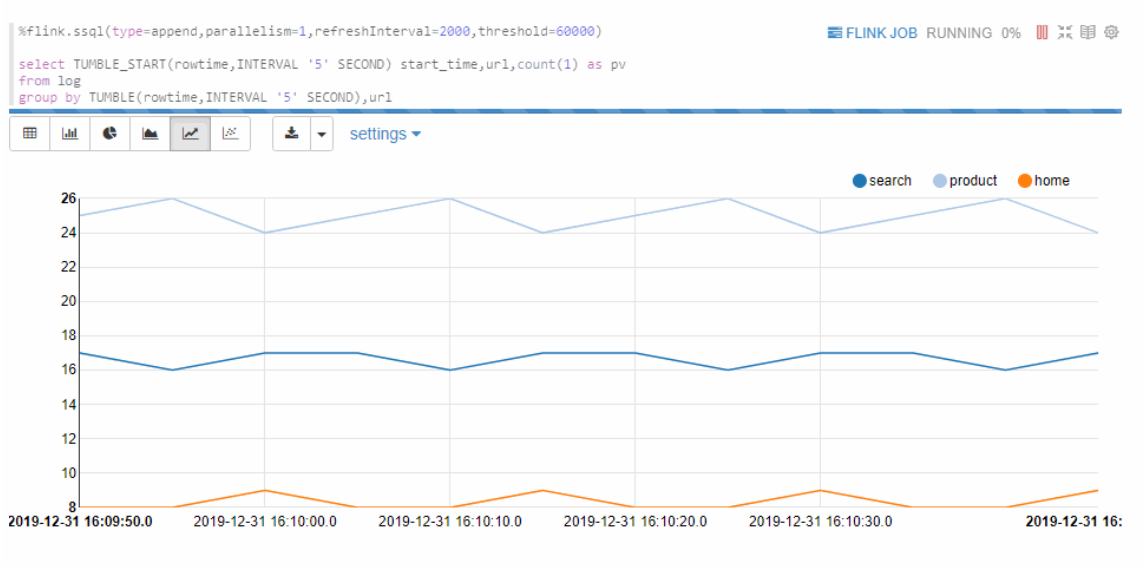


- Append模式

  Append模式适合不断有新数据输出，但不会覆盖原有数据，只会不断append的情况。例如下面的基于窗口的 `group by` 语句。Append模式要求第一列数据类型是TIMESTAMP，本示例的 `start_time` 是TIMESTAMP类型。

```
%flink.ssql(type=append,parallelism=1,refreshInterval=2000,threshold=60000)
select TUMBLE_START(rowtime,INTERVAL '5' SECOND) start_time,url,count(1) as pv from log
group by TUMBLE(rowtime,INTERVAL '5' SECOND),url
```

  返回信息如下所示。



  如果没有呈现如上所示的图表，可能是您的图表配置不对，请按照如下图所示配置图表，然后运行段落。

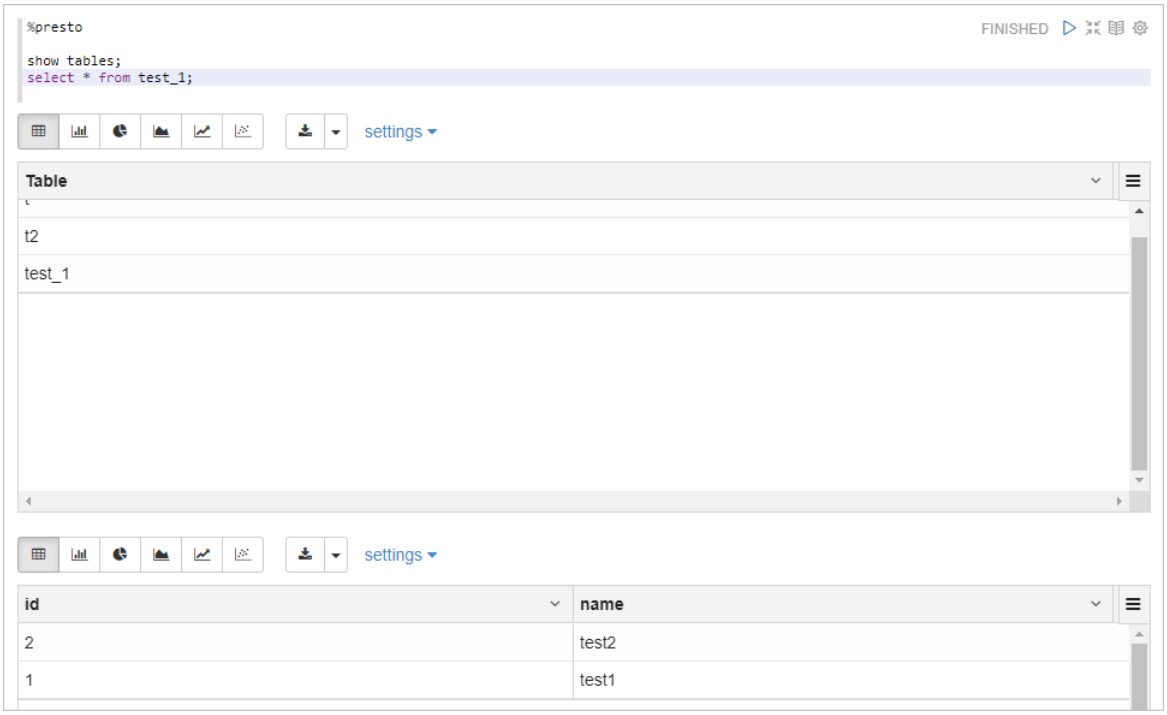
如何使用Presto

1. 在Zeppelin页面，单击**Create new note**。
2. 在**Create new note**对话框，输入**Note Name**，选择**Default Interpreter**为**presto**。
3. 单击**create**。
4. 在Zeppelin的Notebook页面，您可以执行以下命令查看表信息。

   `%presto` 表示执行Presto SQL代码，您无需配置，Zeppelin会自动连接到EMR集群的Presto服务。

```
%presto
show tables;
select * from test_1;
```

返回信息如下所示。



如何使用Impala

1. 在Zeppelin页面，单击**Create new note**。
2. 在**Create new note**对话框，输入**Note Name**，选择**Default Interpreter**为**impala**。
3. 单击**create**。
4. 在Zeppelin的Notebook页面，您可以执行以下命令查看表信息。

`%impala` 表示执行Impala SQL代码，您无需配置，Zeppelin会自动连接到EMR集群的Impala服务。

```
%impala
drop table if exists test_1;
create table test_1(id int,name string);
insert into  test_1 values(1,'test1');
insert into  test_1 values(2,'test2');
select * from test_1;
```

返回信息如下所示。

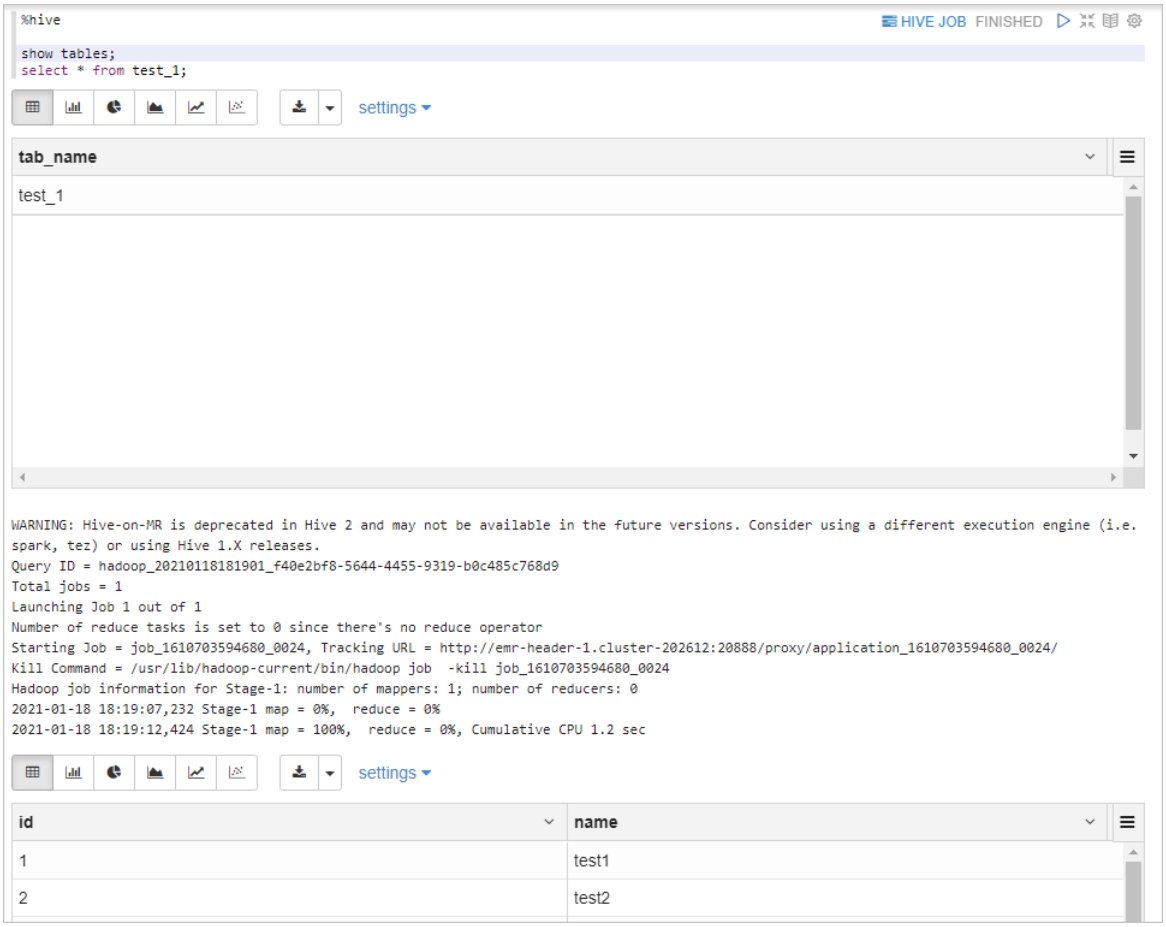
如何使用Hive

1. 在Zeppelin页面，单击**Create new note**。
2. 在**Create new note**对话框，输入**Note Name**，选择**Default Interpreter**为**hive**。
3. 单击**create**。
4. 在Zeppelin的Notebook页面，您可以执行以下命令查看表信息。

   `%hive` 表示执行Hive SQL代码，您无需配置，Zeppelin会自动连接到EMR集群的Hive Thrift Server服务。

   ```
   %hive
   show tables;
   select * from test_1;
   ```

   返回信息如下所示。



6.1.5.2. 交互式开发教程

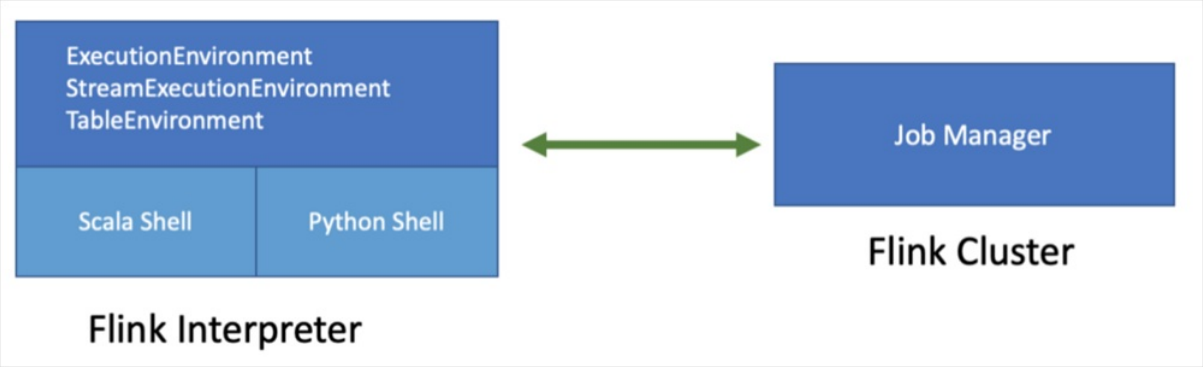
6.1.5.2.1. Flink

Flink是流式计算引擎。本文为您介绍如何在Zeppelin中使用Flink。

背景信息

Zeppelin支持Flink的3种主流语言，包括Scala、PyFlink和SQL。Zeppelin中所有语言共用一个Flink Application，即共享一个ExecutionEnvironment和StreamExecutionEnvironment。例如，您在Scala里注册的table和UDF是可以被其他语言使用的。

Flink解释器的基本架构如下图所示：



Zeppelin支持Flink，您可以在Zeppelin里使用Flink的所有功能。

Zeppelin的SQL开发环境和Flink自带的SQL-Client类似，但提供了更多的特性，具体内容如下：

- 同时支持Batch SQL和Streaming SQL

  Zeppelin中同时支持Batch SQL和Streaming SQL，%flink.ssql用来执行Streaming SQL，%flink.bsql用来执行Batch SQL。

- 支持多条语句

  您可以在Zeppelin的每一个Paragraph中写多条SQL语句，每条SQL语句以分号（;）间隔。

- 支持Comment

  您可以在SQL中添加Comment，`--` 开头的表示单行Comment，`/* */` 包围的表示多行Comment。

- 支持Job并行度配置

  您可以设置每个Paragraph的parallelism来控制Flink SQL Job的并行度。

```
%flink.ssql(parallelism=2)


insert into sink_kafka
select status, direction, cast(event_ts/1000000000 as timestamp(3))
from source_kafka where status <> 'foo'
```

- 支持Multiple insert

  例如，当您有多条INSERT语句读同一个Source时，结果会写到不同的Sink，默认情况下每条SQL语句都会独立运行一个Flink Job，如果您想合并多条语句到同一个Flink Job的话，您就需要设置runAsOne为true。

```
%flink.ssql(runAsOne=true)


insert into sink_kafka
select status, direction, cast(event_ts/1000000000 as timestamp(3))
from source_kafka where status <> 'foo';

insert into sink_kafka2
select status, direction, cast(event_ts/1000000000 as timestamp(3))
from source_kafka where status = 'foo';
```

- JobName的设置

  对于INSERT语句的Flink Job，您可以通过设置jobName的方式来指定Job名称。

  > 注意 只有INSERT语句才支持设置jobName，SELECT语句不支持。此方式只适用于单条INSERT语句，不支持Multiple insert。

```
%flink.ssql(jobName="test job")


insert into sink_kafka
select status, direction, cast(event_ts/1000000000 as timestamp(3))
from source_kafka where status <> 'foo';
```

注意事项

Zeppelin的Flink解释器已经为您创建好了所有Environment变量，您无需创建。

Scala（%flink）

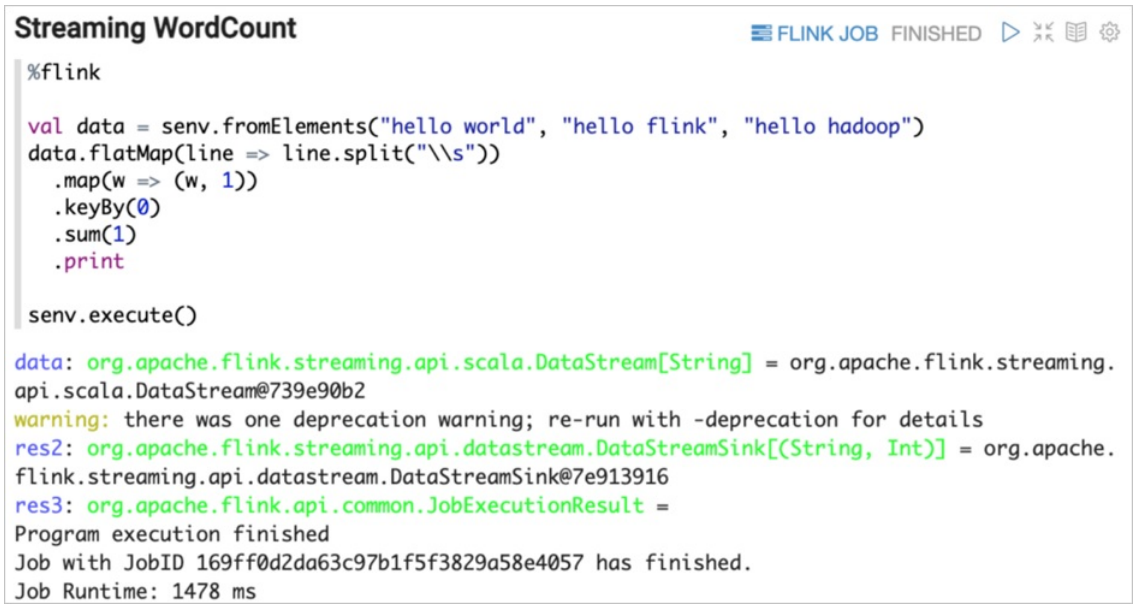
Flink on Zeppelin支持的默认语言是Scala（%flink），也是整个Flink Interpreter内部实现的入口。Flink Interpreter内部会创建Flink Scala Shell，在Flink Scala里会创建Flink的各种Environment。您在Zeppelin上写的Scala代码就会提交到这个Flink Scala Shell里去执行。

Zeppelin已经为您创建了以下7个变量：

- senv（StreamExecutionEnvironment）
- benv（ExecutionEnvironment）
- stenv（StreamTableEnvironment for blink planner）
- btenv（BatchTableEnvironment for blink planner）
- stenv_2（StreamTableEnvironment for flink planner）
- btenv_2（BatchTableEnvironment for flink planner）
- z（ZeppelinContext）

示例1：使用Scala写的Batch WordCount。

示例2：使用Scala写的Streaming WordCount。



PyFlink (%flink.pyflink)

PyFlink是Flink on Zeppelin上Python语言的入口，Flink Interpreter内部会创建Python Shell。Python Shell内部会创建Flink的各种Environment，但是PyFlink里的各种Environment变量对应的Java变量都是Scala Shell创建的。您在Zeppelin上写的Python代码会提交到这个Python Shell里去执行。

Zeppelin已经为您创建了以下7个变量：

- s_env（StreamExecutionEnvironment）

- b_env（ExecutionEnvironment）
- st_env（StreamTableEnvironment for blink planner）
- bt_env（BatchTableEnvironment for blink planner）
- st_env_2（StreamTableEnvironment for flink planner）
- bt_env_2（BatchTableEnvironment for flink planner）
- z（ZeppelinContext）

SQL (%flink.ssql和%flink.bsql)

Zeppelin支持以下两种不用场景的SQL:

- %flink.ssql：Streaming SQL，使用StreamTableEnvironment来执行SQL。
- %flink.bsql：Batch SQL， 使用BatchTableEnvironment来执行SQL。

Flink支持的SQL语句类型如下:

- DDL（Data Definition Language）
- DML（Data Manipulation Language）
- DQL（Data Query Language）
- Flink定制语句

  例如，SET和HELP语句。

Streaming SQL结果可视化

Zeppelin对于Select语句的结果以流式的方式可视化。

Zeppelin可视化类型包含Single模式、Update模式和Append模式。

- Single模式

  Single模式适合当输出结果是一行的情况，不适合用图形化的方式展现，使用文本的方式更适合。

  例如，下面的Select语句，这条SQL语句只有一行数据，但这行数据会持续不断的更新。

```
%flink.ssql(type=single, parallelism=1, refreshInterval=3000, template=<h1>{1}</h1> until <h2>{0}</h2>)
select max(event_ts), count(1) from sink_kafka
```

  此模式的数据输出格式是HTML形式，您可以用template来指定输出模板，{i}是第i列的placeholder。

  结果如下图所示。

- Update模式

  Update模式适合多行输出的情况，此模式会定期更新数据，输出格式是Zeppelin的table格式。例如，SELECT GROUP BY语句。

```
%flink.ssql(type=update, refreshInterval=2000, parallelism=1)
select status, count(1) as pv from sink_kafka group by status
```

结果如下图所示。

- Append模式

  Append模式适合时间序列数据，不断有新数据输出，但不会覆盖原有数据，只会不断Append的情况。

  例如，基于窗口的GROUP BY语句。Append模式要求第一列数据类型是TIMESTAMP，代码中的start_time就是TIMESTAMP类型。

```
%flink.ssql(type=append, parallelism=1, refreshInterval=2000, threshold=60000)
select TUMBLE_START(event_ts, INTERVAL '5' SECOND) as start_time, status, count(1) from sink_kafka
group by TUMBLE(event_ts, INTERVAL '5' SECOND), status
```

结果如下图所示。

UDF

在Zeppelin您可以通过以下4种方式使用或者定义UDF：

- 在Zeppelin中直接写Scala UDF。

  此方式适合写比较简单的UDF。您只需定义UDF的Class，然后在btenv或stenv中注册定义的UDF。完成注册后，您可以在%flink.bsql或%flink.ssql中使用，因为%flink.bsql或%flink.ssql共享同一个Catalog。

```
%flink
class ScalaUpper extends ScalarFunction {
  def eval(str: String) = str.toUpperCase
}
btenv.registerFunction("scala_upper", new ScalaUpper())
```

- 在Zeppelin中直接写PyFlink UDF。

  此方式适合写比较简单的UDF。在Zeppelin中定义PyFlink UDF与Scala UDF类似。您只需定义UDF的Class，然后在btenv或stenv中注册定义的UDF。完成注册后，您可以在%flink.bsql或%flink.ssql中使用，因为%flink.bsql或%flink.ssql共享同一个Catalog。

```
%flink.pyflink
class PythonUpper(ScalarFunction):
  def eval(self, s):
    return s.upper()
bt_env.register_function("python_upper", udf(PythonUpper(), DataTypes.STRING(), DataTypes.STRING()))
```

- 使用SQL创建UDF。

  部分简单的UDF可以在Zeppelin里直接写，但是如果是一些复杂的UDF，建议在IDE里写，然后在Zeppelin中创建注册。例如：

```
%flink.ssql
CREATE FUNCTION myupper AS 'org.apache.zeppelin.flink.udf.JavaUpper';
```

  因为使用此方式前提是这个UDF对应的JAR包必须要在CLASSPATH里，所以您需要先配置flink.execution.jars，把这个UDF的JAR包放到CLASSPATH里。

```
flink.execution.jars /Users/jzhang/github/flink-udf/target/flink-udf-1.0-SNAPSHOT.jar
```

- 使用flink.udf.jars来指定含有UDF的JAR包。

  如果您有多个UDF，并且UDF之间的逻辑比较复杂，您也不想注册UDF，此时您可以使用flink.udf.jars。

  i. 在IDE里创建Flink UDF项目，编写UDF。

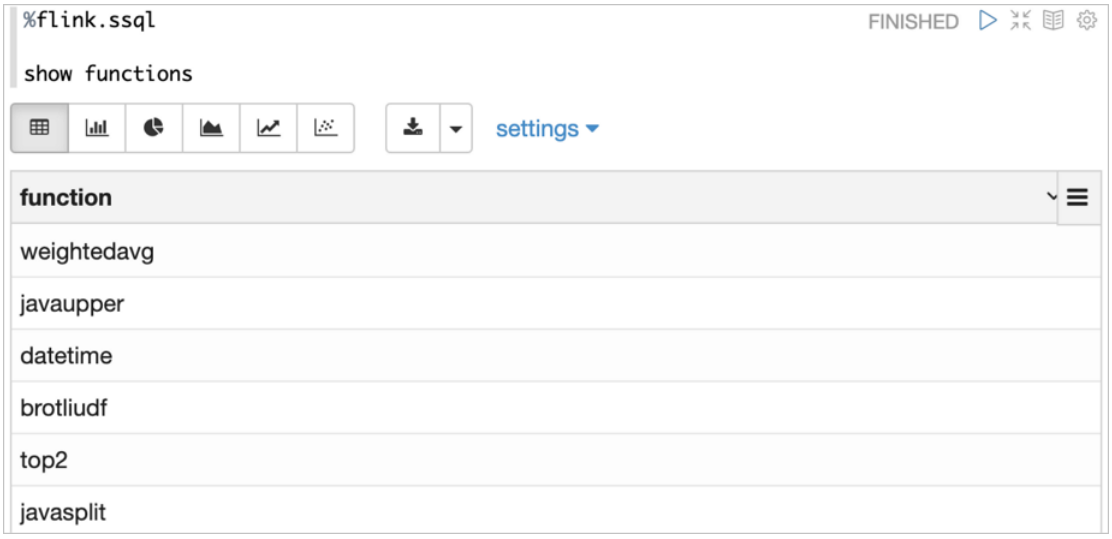
     示例详情，请参见Flink UDF。

  ii. 使用flink.udf.jars指定Flink UDF项目的JAR包。

```
flink.udf.jars /Users/jzhang/github/flink-udf/target/flink-udf-1.0-SNAPSHOT.jar
```

  Zeppelin会扫描JAR包，然后检测出所有的UDF，并且自动注册UDF，UDF的名字就是Class名字。

  例如，show functions的结果如下。



  > **说明** 默认情况下Zeppelin会扫描JAR包里的所有的Class，如果JAR包过大可能会导致性能问题。此时您可以设置flink.udf.jars.packages来指定扫描的Package，以减少扫描的Class数目。

第三方依赖

Flink作业经常会有第三方依赖。Flink通常使用以下2种配置方式添加依赖：

- flink.excuetion.packages

  此方式和在*pom.xml*里添加依赖是类似的，会下载相应的Package和所有Transitive依赖到CLASSPATH里。

例如，添加Kafka Connector依赖。

```
flink.execution.packages    org.apache.flink:flink-connector-kafka_2.11:1.10.0,org.apache.flink:flink-connector-kafka-base_2.11:1.10.0,org.apache.flink:flink-json:1.10.0
```

Package的格式为artifactGroup:artifactId:version，当有多个Package时，每个Package用逗号（,）分隔。

> **说明** 此方式需要Zeppelin机器可以访问外网，如果不能访问外网的话，建议您使用flink.execution.jars方式。

- flink.execution.jars

  如果您的Zeppelin机器不能访问外网或者您的依赖没有发布在Maven Repository里，则您可以将依赖的JAR包放到Zeppelin机器上，然后通过flink.execution.jars来指定这些JAR包，多个JAR包时用逗号（,）分隔。

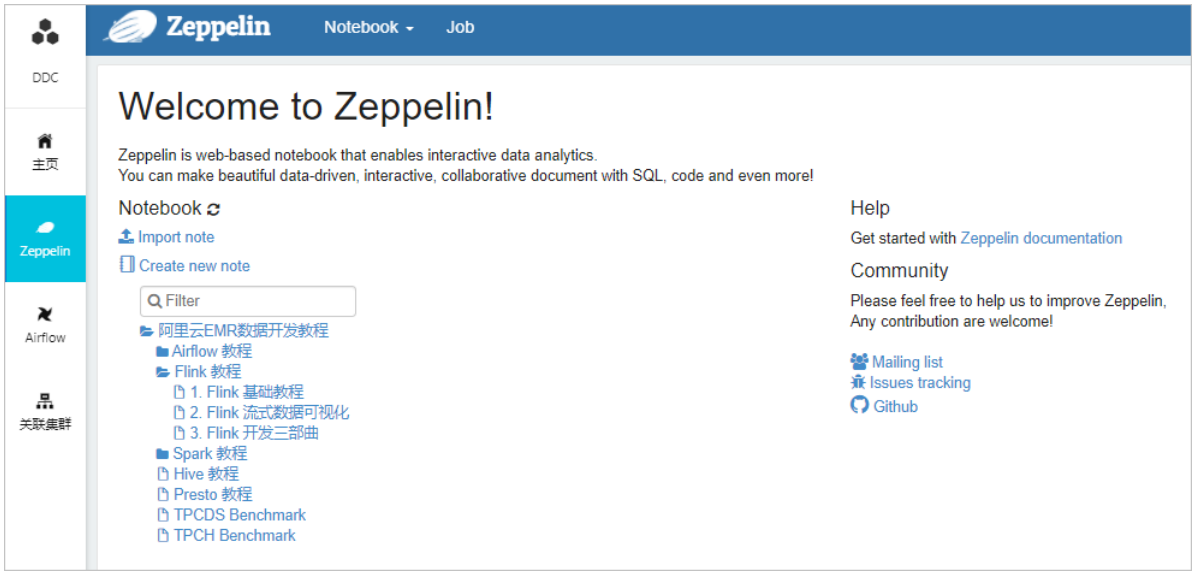
  例如，添加Kafka Connector依赖。

```
flink.execution.jars /Users/jzhang/github/flink-kafka/target/flink-kafka-1.0-SNAPSHOT.jar
```

> **说明** 示例中的*flink-kafka-1.0-SNAPSHOT.jar*是通过pom.xml文件构建的。

内置教程

EMR数据开发集群自带了很多Flink教程，详细信息请在下图页面查看。



6.1.5.2.2. Hive

Zeppelin的Hive解释器是使用JDBC连接HiveServer2。本文为您介绍如何在Zeppelin中使用Hive。

背景信息

EMR数据开发的Zeppelin在以下两方面做了增强：

- 在多个EMR集群中动态切换。
- 您无需配置，所有配置都是自动完成。

Zeppelin的Hive解释器提供以下主要功能：

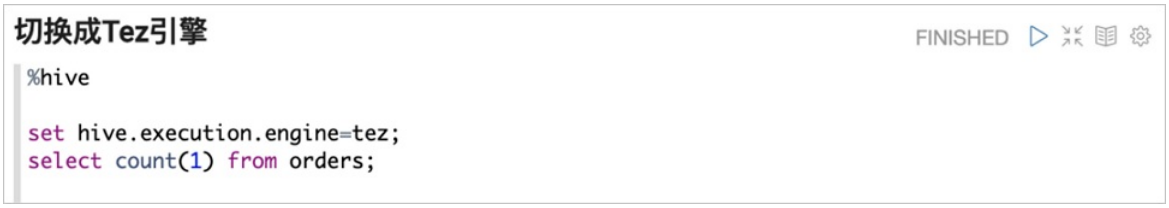
- 支持任何Hive SQL语句（包括DDL和DML等）。

  Hive解释器与Beeline支持的语言及功能一样。
- 支持一个段落编写多条SQL语句，每条SQL语句以分号（;）结尾，按照从上到下的顺序执行。
- 支持注释功能。
- 支持并发执行多个段落。
- 支持参数功能。
- 支持图表展示SQL结果。

示例

- 多行SQL

如下图所示，在一个段落里执行2句SQL语句（以分号间隔），第一句切换执行引擎，第二句运行Select语句。



- 注释

  以 `--` 开头的均为SQL注释。

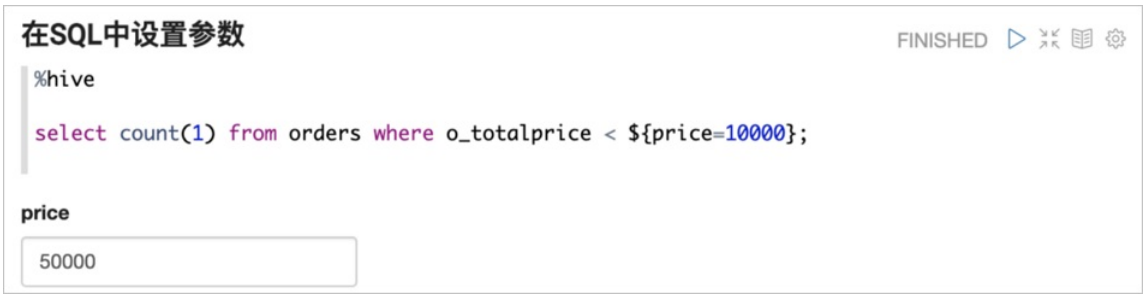
- 参数功能

  Zeppelin支持${var=value} 形式的差数。var是变量名，value是默认值。

  例如，通常不使用变量时的查询SQL语句如下：

```
select count(1) from orders where o_totalprice < 10000;
```

  在Zeppelin中使用${var=value} 形式的参数查询语句时，设置变量名为price，默认值是10000，此时页面上会显示一个文本框，您可以在文本框里填写任意值，按回车键，即使用文本框里的值替换变量price并运行SQL语句。例如，填写的值为50000，运行的SQL语句如下图所示：

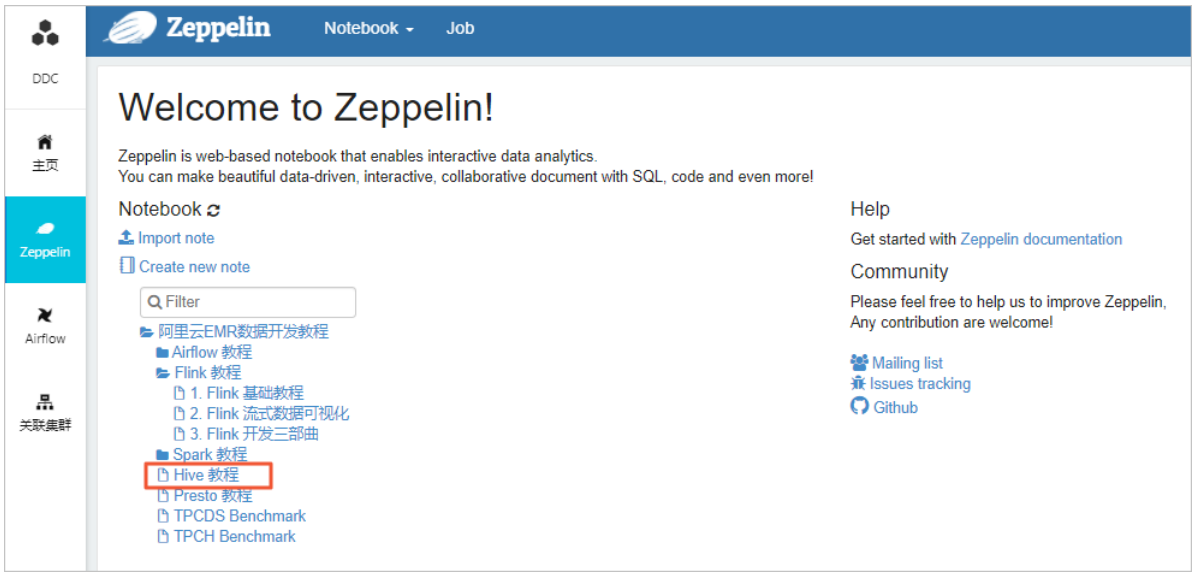


- 图表展示SQL结果

Zeppelin内置了很多图表，您可以通过图表方式展示SQL结果。如下图所示，通过柱状图展示一个GROUP BY语句的结果。

内置教程

EMR数据开发集群自带了很多开发教程，详细信息请在如下图页面查看。



6.1.5.2.3. Spark

Spark是一个通用的大数据计算引擎。本文为您介绍如何在Zeppelin中使用Spark。

背景信息

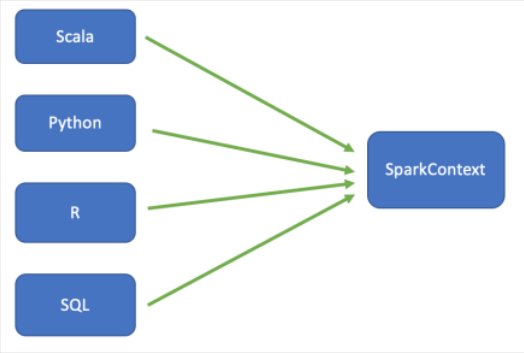
Zeppelin支持2个大类的 Spark 作业类型：代码类型作业以及JAR包类型作业

作业类型	描述	适合场景
代码类型作业	用传统的notebook方式，在Zeppelin里写代码，然后在Zeppelin里执行这些代码	逻辑简单的Spark作业
JAR包类型作业	在IDE里写代码，并打包成JAR包，在Zeppelin里提交JAR包到Hadoop集群运行Spark作业	逻辑复杂，有比较多的依赖，需要依赖IDE来编写代码

。

通常对于逻辑简单的作业，建议您使用代码类型，直接在Zeppelin里直接写代码，提交Spark作业。这样做的好处是您可以以交互式的方式来开发Spark作业，开发效率会更高，还可以用到Zeppelin的其他高级功能，比如可视化，Dynamic forms 等等。 Zeppelin支持Spark的4种主流语言，包括Scala、PySpark、R和SQL。Zeppelin中所有语言在同一个Spark Application里，即共享一个SparkContext和SparkSession。例如，您在Scala里注册的table和UDF是可以被其他语言使用的。

Spark解释器的基本架构如下图所示：



Zeppelin支持Spark，您可以在Zeppelin里使用Spark的所有功能。

本文通过以下方面为您介绍Spark:

- Scala（%spark）
- PySpark（%spark.pyspark）
- SparkR（%spark.r）
- SQL（%spark.sql）
- Spark-submit 提交作业
- 配置Spark
- 配置Driver，Executor
- 配置 Spark App 的Name和queue
- 第三方依赖
- Spark Application的执行用户
- 配置 PySpark 环境
- 内置教程

Scala（%spark）

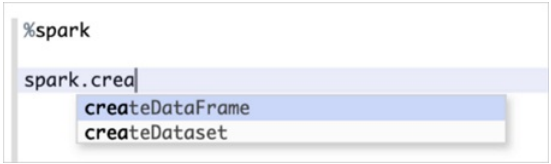
以%spark开头的就是Scala代码的段落（Paragraph）。因为Zeppelin已经为您内置了Scala的SparkContext （sc）和SparkSession（spark）变量，所以您无需再创建SparkContext或者SparkSession。

代码示例如下：

```
%spark
val sum = sc.range(1,10).sum()
println("Sum = " + sum)
```

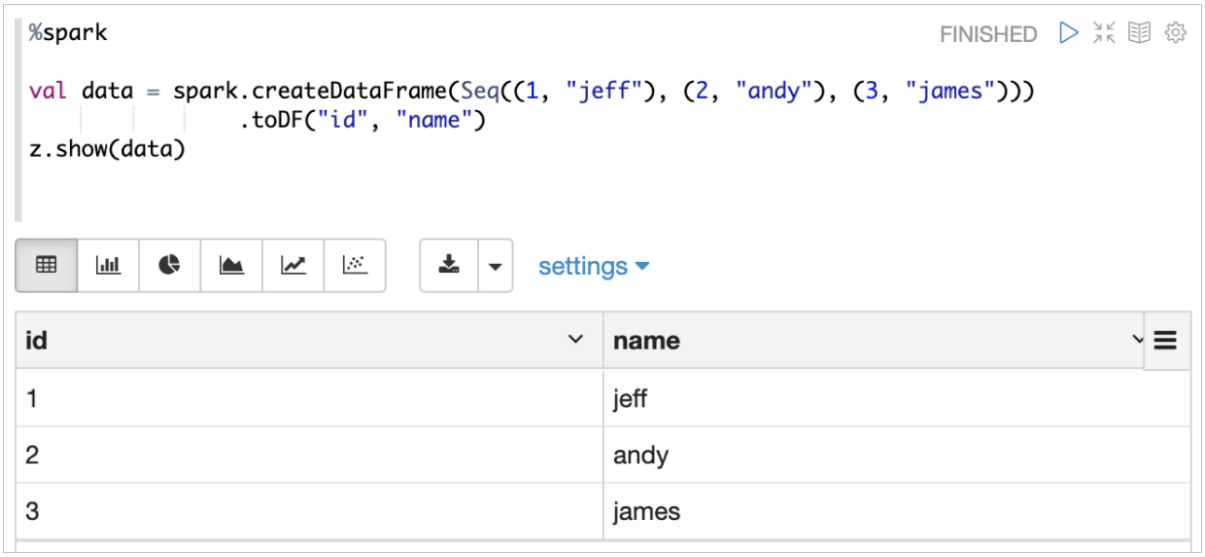
- Code Completion

  Zeppelin里的Scala Shell是支持Code Completion的，按Tab键即可显示当前环境下的候选方法名，如下图所示：



- ZeppelinContext

  ZeppelinContext（变量名为z）是在Spark Scala Shell环境下创建的一个提供一些高级用法的Class。比较实用的方法是z.show。使用z.show展示DataFrame示例如下所示：



- Scala 段落执行顺序

  同一时间只能有一个Scala段落执行，Zeppelin是按照FIFO (First in First out) 的顺序来执行Scala段落。所以如果您的一个Scala段落里正在一个Spark作业的话，另外一个Scala段落只能等待（处于PENDING状态）。

更多 Spark scala 的例子，请参考Zeppelin自带的教程 Spark SQL 教程（Scala）

PySpark（%spark.pyspark）

以%spark.pyspark开头的就是PySpark代码的段落（Paragraph）。因为Zeppelin已经为您内置了PySpark的SparkContext （sc）和SparkSession（spark）变量，所以您无需再创建SparkContext或者SparkSession。

代码示例如下：

```
%spark.pyspark
sum = sc.range(1,10).sum()
print("Sum = " + str(sum))
```

- 配置PySpark

  使用PySpark非常简单，只需要在Spark的interpreter里设置 PYSPARK_PYTHON 和 PYSPARK_DRIVER_PYTHON 为python的可执行文件的路径（默认是用PATH里的python），因为有时候您的系统里可能安装了多个python版本，所以需要设置这两个配置（其实这两个配置是标准的Spark配置选项，不是zeppelin引入的配置选项）

- Python 段落执行顺序

  同一时间只能有一个Python段落执行，Zeppelin是按照FIFO (First in First out) 的顺序来执行Python段落。所以如果您的一个Python段落里正在运行一段Python代码的话，另外一个Python段落只能等待（处于PENDING状态）。

SparkR（%spark.r）

如果您需要使用SparkR，那么请确保您的EMR集群里安装了R语言以及knitr包（需要在每个NodeManager节点上安装，因为数据开发中默认配置的是Yarn-Cluster模式，Driver有可能运行在任意一个NodeManager节点上）。

安装命令如下：

1. 执行如下命令安装R语言。

   ```
   sudo yum install epel-release
   sudo yum install R
   ```

2. 在R里面安装knitr包。

   ```
   install.packages("knitr")
   ```

示例如下：

1. 使用SparkR创建一个DataFrame，并注册为一张table。

```
%spark.r                                                    SPARK JOB FINISHED

localNames <- data.frame(name=c("John", "Smith", "Sarah"), budget=c(19, 53, 18))
names <- createDataFrame(sqlContext, localNames)
printSchema(names)
registerTempTable(names, "names")


root
 |- name: string (nullable = true)
 |- budget: double (nullable = true)
```

2. 使用Spark SQL查询注册的table。

```
%spark.sql                                                  SPARK JOB FINISHED

select * from names
```

settings

name	budget
John	19.0
Smith	53.0
Sarah	18.0

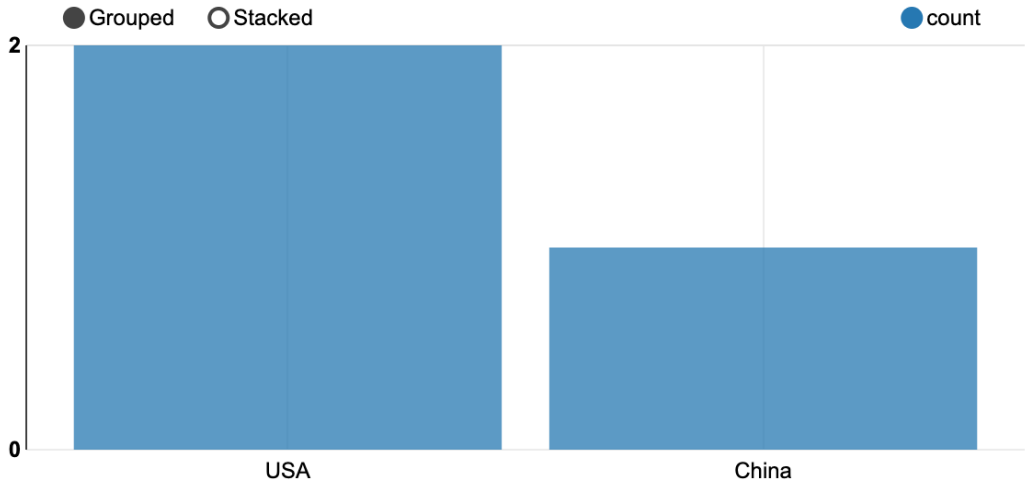
SQL（%spark.sql）

以%spark.sql开头的就是Spark SQL的段落（Paragraph）。您可以运行所有Spark支持的SQL语句，通过Zeppelin可视化展示，如下图所示：

```
%spark.sql                                                  SPARK JOB FINISHED

select country, count(1) as count from people group by country
```

settings



Zeppelin的Spark SQL解释器和其他Spark解释器（PySpark、SparkR和Spark解释器）共享SparkContext和SparkSession，即用其他Spark解释器注册的表也可以使用Spark SQL解释器进行访问。例如：

```
%spark
case class People(name: String, age: Int)
var df = spark.createDataFrame(List(People("jeff", 23), People("andy", 20)))
df.createOrReplaceTempView("people")
```

```
%spark.sql
select * from people
```

- 支持多语句

  %spark.sql支持多条SQL语句，每条SQL语句以分号(;)隔开。

- 支持SQL注释

  您可以在SQL中穿插 comment，有2种comment

  - 以 -- 开头的单行comment
  - /* */ 包围的多行comment

- 支持并发

  目前默认SQL也是串行执行的，没有启用并发。

  您可以在Spark解释器配置中将 zeppelin.spark.concurrentSQL 设为true即可启用SQL并发支持，另外还可以通过zeppelin.spark.concurrentSQL.max控制最大允许的并发数

另外，由于Spark SQL本身的特性，Spark 支持大多数Hive SQL语法。Spark集成Hive后，通常场景下，您可以使用Spark SQL解释器访问Hive表来进行更高效的分析计算，数据开发里的Spark解释器默认已经开启了Hive。

Spark-submit 提交作业

如果您的Spark作业逻辑比较复杂，需要借助于IDE，那么您可以在本地构建JAR包，然后上传到oss，在Zeppelin里用 %spark-submit解释器来提交JAR包类型作业，和spark-submit命令一样，您可以添加任何Spark配置，比如下图：

```
%spark-submit

spark-submit --class org.apache.spark.examples.SparkPi --master yarn --deploy-mode cluster oss://jzhang-test-hz/spark-examples_2.11-2.4.6.jar
```

FINISHED

```
21/07/23 15:23:07 WARN [main] NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
21/07/23 15:23:07 WARN [main] DependencyUtils: Skip remote jar oss://jzhang-test-hz/spark-examples_2.11-2.4.6.jar.
21/07/23 15:23:07 INFO [main] RMProxy: Connecting to ResourceManager at emr-header-1.cluster-235564/192.168.0.225:8032
21/07/23 15:23:07 INFO [main] Client: Requesting a new application from cluster with 2 NodeManagers
21/07/23 15:23:08 WARN [main] DomainSocketFactory: The short-circuit local reads feature cannot be used because libhadoop cannot be loaded.
21/07/23 15:23:08 INFO [main] Configuration: resource-types.xml not found
21/07/23 15:23:08 INFO [main] ResourceUtils: Unable to find 'resource-types.xml'.
21/07/23 15:23:08 INFO [main] Client: Verifying our application has not requested more than the maximum memory capability of the cluster (26144 MB per container)
21/07/23 15:23:08 INFO [main] Client: Will allocate AM container, with 3072 MB memory including 1024 MB overhead
21/07/23 15:23:08 INFO [main] Client: Setting up container launch context for our AM
21/07/23 15:23:08 INFO [main] Client: Setting up the launch environment for our AM container
21/07/23 15:23:08 INFO [main] Client: Preparing resources for our AM container
21/07/23 15:23:08 WARN [main] Client: Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading libraries under SPARK_HOME.
21/07/23 15:23:11 INFO [main] Client: Uploading resource file:/tmp/spark-7a61b796-fe94-4ee5-9881-b22c05bd2a3f/__spark_libs__2445726863891054189.zip -> hdfs://emr-header-1.cluster-2
35564:9000/user/hadoop/.sparkStaging/application_1625562988292_0071/__spark_libs__2445726863891054189.zip
21/07/23 15:23:11 INFO [Thread-7] SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
21/07/23 15:23:11 INFO [DataStreamer for file /user/hadoop/.sparkStaging/application_1625562988292_0071/__spark_libs__2445726863891054189.zip] SaslDataTransferClient: SASL encrypti
on trust check: localHostTrusted = false, remoteHostTrusted = false
```

配置Spark

在阿里云EMR的数据开发里，Spark解释器配置的是Isolated Per Note模式，也就是说每个Note都有一个独立的Spark Application（每个Note对应一个Yarn App，Zeppelin里使用的是Yarn-Cluster模式）。对于每一个Spark Application，您可以在%spark.conf里配置所有Spark相关的配置选项，具体配置选项，请参见Spark Configuration。

代码示例如下：

spark.conf 例子

```
%spark.conf

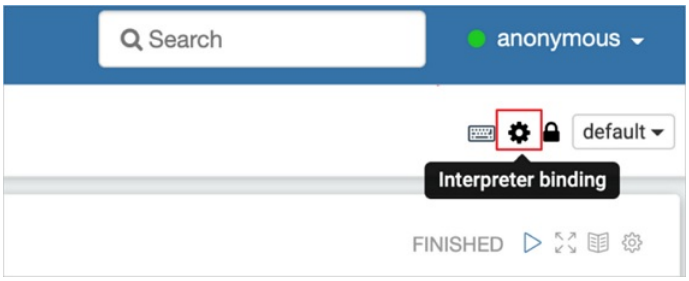

spark.app.name SparkExample
spark.driver.memory 1g
spark.executor.memory 2g

# 配置以下属性来设置Spark dynamic executor allocation, http://spark.apache.org/docs/2.4.7/configuration.html#dynamic-allocation
# spark.dynamicAllocation.enabled true
# spark.dynamicAllocation.initialExecutors 1
# spark.dynamicAllocation.maxExecutors 10
# spark.shuffle.service.enabled true
```

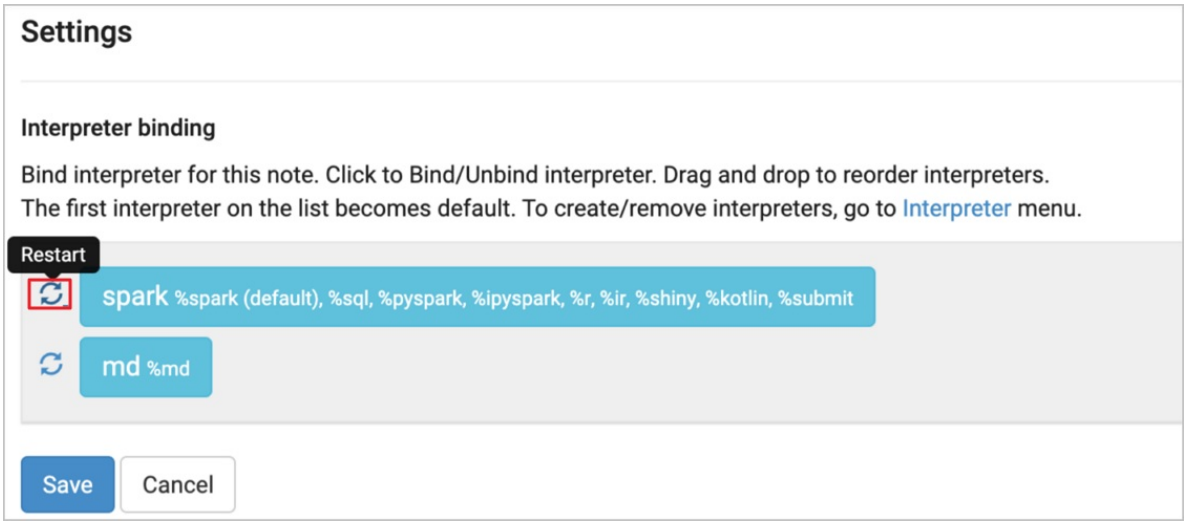
> 注意 因为Spark只支持在启动Spark Application之前配置属性，所以%spark.conf必须在运行Spark代码之前运行，一般都把%spark.conf 放在第一个段落。

如果您在启动Spark Application之后想修改配置（例如，修改Driver Memory），则您必须重启当前Note的Spark解释器。重启步骤如下：

1. 单击 图标。



2. 单击 图标。



3. 重新运行%spark.conf段落。

EMR数据开发中的Spark解释器默认配置如下：

- Yarn-Cluster模式。
- 启用Hive。
- Driver内存1 GB，Executor内存1 GB。
- 启用Dynamic Executor Allocation。

  初始化Executor 1个，最大Executor 10个，更多配置信息，请参见Spark Configuration。

配置Driver，Executor

Spark driver 和 executor 是最常需要做的配置更改， 您可以在 %spark.conf 里配置标准的spark配置信息，比如下面的例子把driver 和 executor 的内存调为4g，executor设为5个

```
%spark.conf
spark.driver.memory 4g
spark.executor.memory 4g
spark.executor.instances 5
```

配置 Spark App 的Name和queue

和配置Spark driver 和 executor 一样， 您可以在 %spark.conf 里配置Spark Application的yarn app name 和 queue。

```
%spark.conf
spark.app.name MySparkApp
spark.yarn.queue queue_1
```

第三方依赖

Spark作业经常会有第三方依赖。标准的Spark通常使用以下3种配置：

- spark.jars

  spark.jars可以用来指定JAR文件，多个JAR包可以用逗号（,）隔开。您可以把JAR包放在OSS上，也可以放在目标EMR集群的HDFS上。建议您放在OSS上，以便于您的JAR包可以被多个EMR集群共享，即使切换集群也不用更改配置。

如下图示例，段落①使用%spark.conf指定OSS上的JAR，这个JAR里写了一个Java UDF，段落②使用PySpark注册Java UDF（您也可以用Scala来注册UDF），然后在SQL中使用Java UDF。

- spark.jars.packages

  spark.jars.packages可以用Package的形式来指定依赖包，Spark会动态下载这些Package到ClassPath里，多个Package以逗号（,）分隔。

  如下图示例，段落①指定delta包，段落②使用这个delta包。

- spark.files

  spark.files用来指定File文件，多个File文件以逗号（,）分隔，您也可以使用fOSS上的文件。

Spark Application的执行用户

EMR Studio 里的 Zeppelin 启用了用户认证，您需要用LDAP里的用户登入，Spark Application的执行用户就是这个登入用户。

配置 PySpark 环境

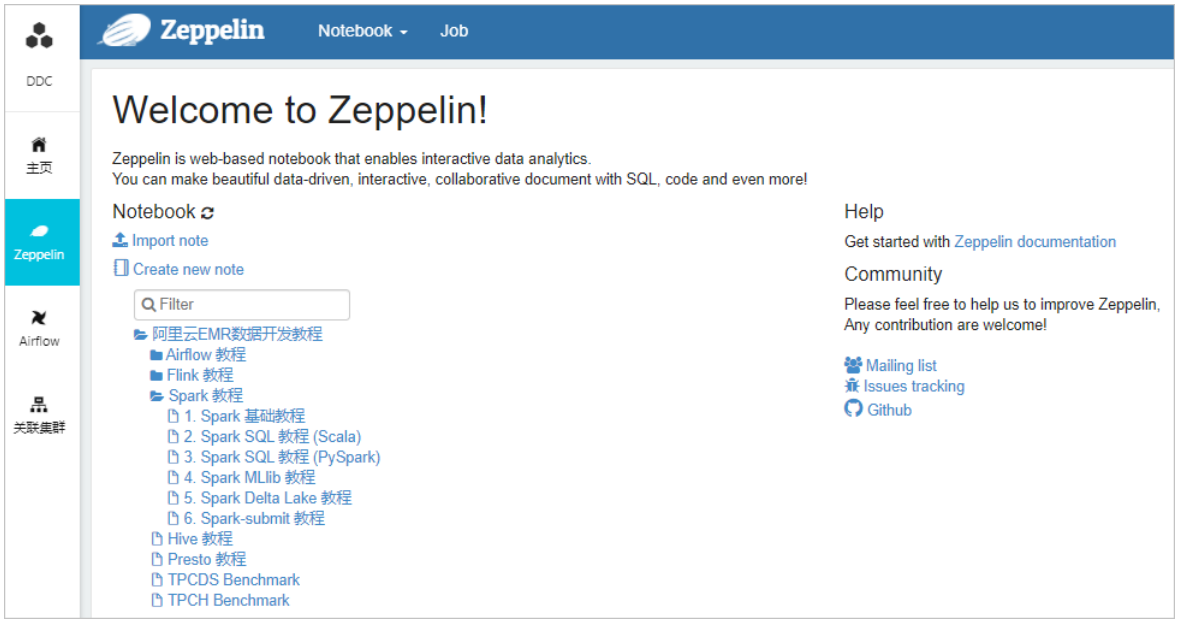
PySpark的环境配置对一般用户来都比较麻烦，您需要在集群所有节点上安装同一个版本的Python，如果要安装第三方库，也要在每个节点安装，而且这种方式没法做到Python环境的隔离。在EMR Studio里我们提供了一个中使用conda创建Python虚拟环境的方法来解决这个问题，具体步骤如下

- 使用conda 创建Python环境
- 使用conda pack 打包Python环境
- 上传Python环境tar包到oss
- 配置Spark Application使用这个Python环境

具体使用方法请参考Zeppelin自带的教程： 02. PySpark 基础教程

内置教程

EMR数据开发集群自带了很多Spark教程，详细信息请在如下图页面查看。



6.1.5.2.4. Presto

Zeppelin的Presto解释器是使用JDBC连接Presto。本文为您介绍如何在Zeppelin中使用Presto。

背景信息

EMR数据开发的Zeppelin在以下两方面做了增强：

- 在多个EMR集群中动态切换。
- 无需配置，所有配置都是自动完成。

Zeppelin的Presto解释器提供以下主要功能：

- 支持任何Presto SQL语句（包括DDL和DML等）。
- 支持多行SQL，每条SQL语句以分号（;）结尾。
- 支持注释功能。
- 支持并发执行多条SQL。
- 支持参数功能。
- 支持图表展示SQL结果。

示例

- 多行SQL

  如下图所示，在一个段落里执行2句SQL语句（以分号间隔），第一句切换database，第二句运行Select语句。

- 注释

以 -- 开头的均为SQL注释。

```
%presto

-- 注释
use tpch_text_5;
select count(1) from orders;

Query executed successfully. Affected rows : 0
```

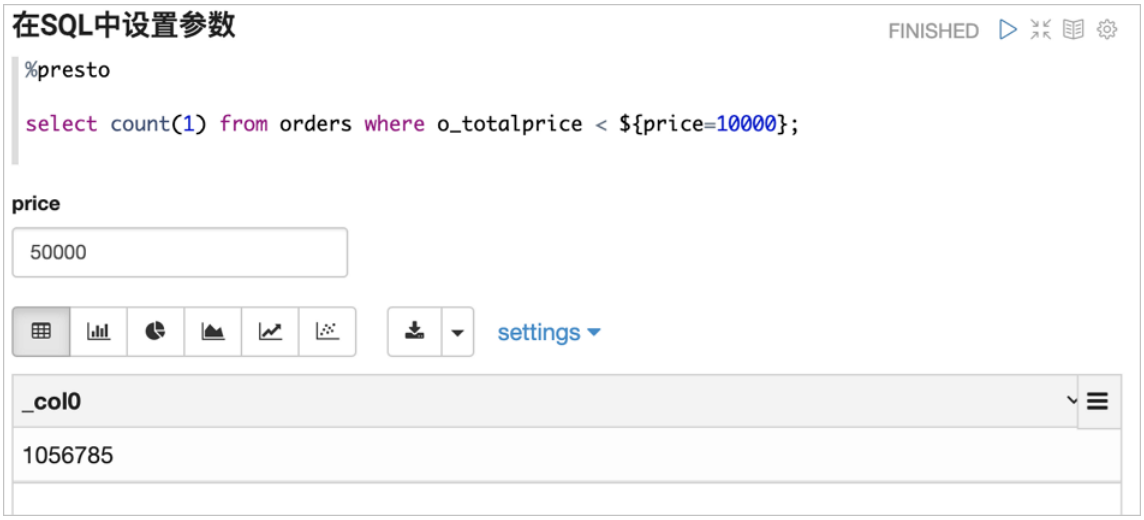
- 参数功能

  Zeppelin支持${var=value} 形式的参数。var是变量名，value是默认值。

  例如，通常不使用变量时的查询SQL语句如下：

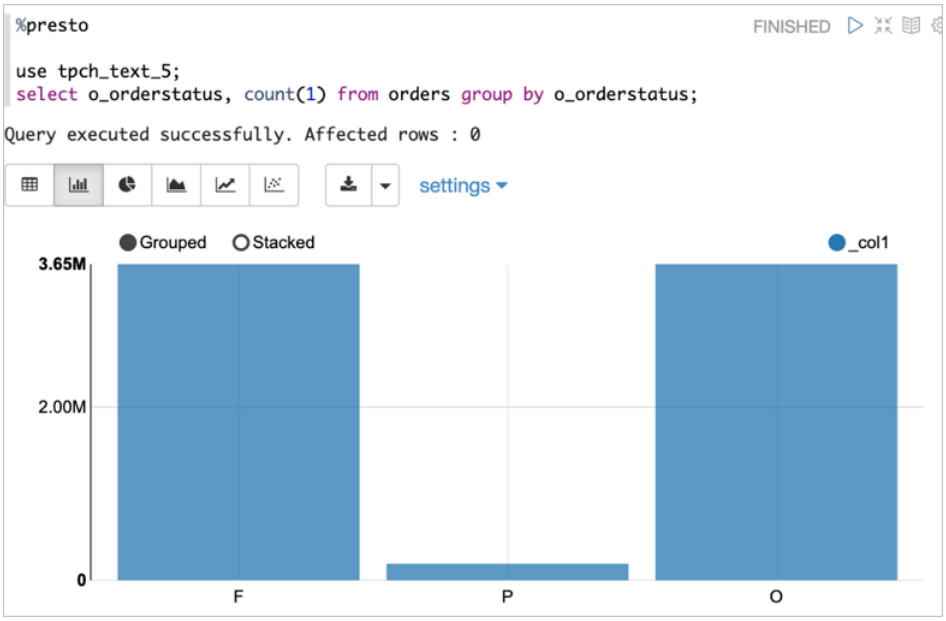
```
select count(1) from orders where o_totalprice < 10000;
```

  在Zeppelin中使用${var=value} 形式的参数查询语句时，设置变量名为price，默认值是10000，此时页面上会显示一个文本框，您可以在文本框里填写任意值，按回车键，即使用文本框里的值替换变量price并运行SQL语句。例如，填写的值为50000，运行的SQL语句如下图所示：



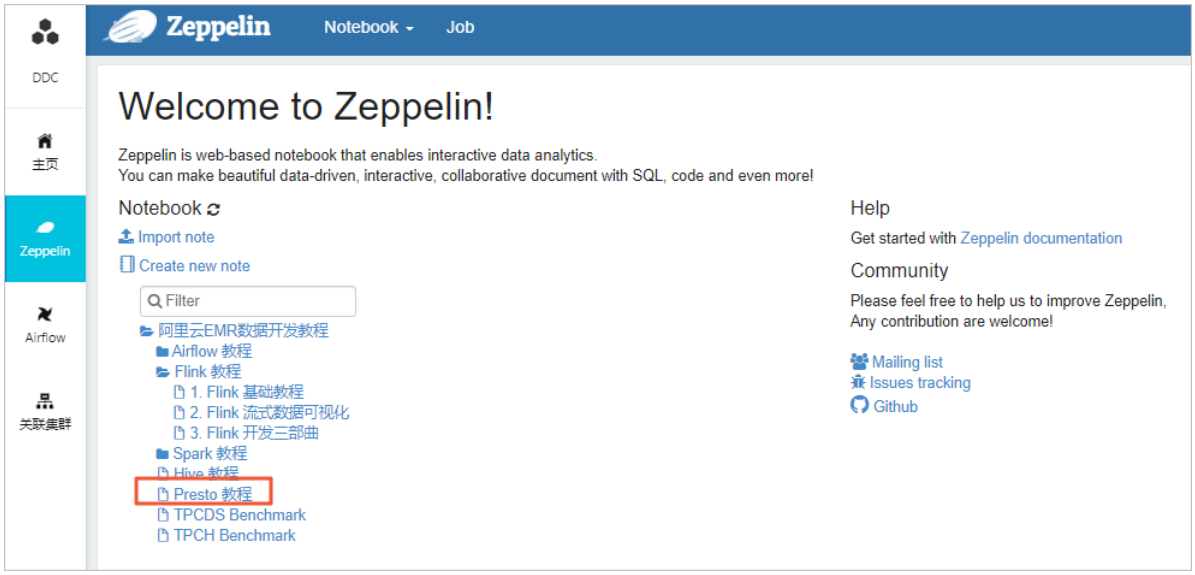
- 图表展示SQL结果

  Zeppelin内置了很多图表，您可以通过图表方式展示SQL结果。如下图所示，通过柱状图展示一个GROUP BY语句的结果。



内置教程

EMR数据开发集群自带了很多开发教程，详细信息请在如下图页面查看。



6.1.5.2.5. Shell

Zeppelin支持Shell脚本（以%sh开头）。与开源Zeppelin相比，E-MapReduce（简称EMR）数据开发集群中的Shell解释器支持在不同EMR集群环境里切换。本文通过示例为您介绍如何在Zeppelin中使用Shell。

使用示例

- 运行hadoop命令

  执行如下命令会显示当前EMR集群根目录下的所有文件，切换到不同的EMR集群，会显示不同集群下的情况。

  ```
  hadoop fs -ls /
  ```

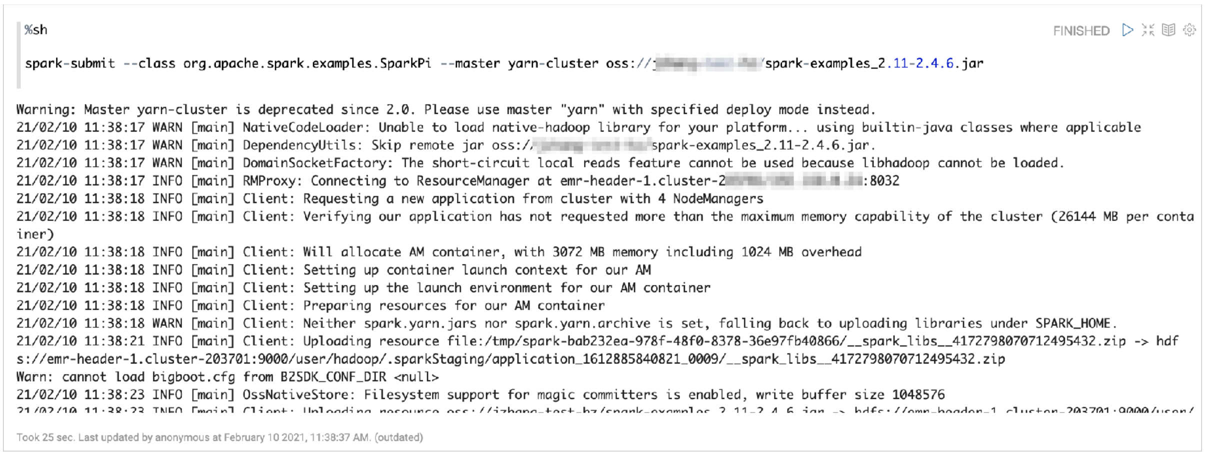
  返回信息如下图所示：

  ```
  %sh                                                    FINISHED

  hadoop fs -ls /

  Found 7 items
  drwxr-xr-x   - hadoop    hadoop          0 2021-01-27 14:25 /apps
  drwxrwxrwx   - flowagent hadoop          0 2021-01-27 14:25 /emr-flow
  drwxr-x--x   - root      hadoop          0 2021-01-27 14:25 /emr-sparksql-udf
  drwxr-x--x   - hadoop    hadoop          0 2021-02-09 12:06 /flink
  drwxr-x--x   - hadoop    hadoop          0 2021-02-23 10:40 /spark-history
  drwxrwxrwt   - hive      hadoop          0 2021-02-23 10:40 /tmp
  drwxr-x--t   - hadoop    hadoop          0 2021-02-12 08:28 /user
  ```

- 运行Spark-Submit命令提交Spark作业。示例如下图所示：


%sh
FINISHED
spark-submit --class org.apache.spark.examples.SparkPi --master yarn-cluster oss://spark-examples_2.11-2.4.6.jar
Warning: Master yarn-cluster is deprecated since 2.0. Please use master "yarn" with specified deploy mode instead.
21/02/10 11:38:17 WARN [main] NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
21/02/10 11:38:17 WARN [main] DependencyUtils: Skip remote jar oss:/spark-examples_2.11-2.4.6.jar.
21/02/10 11:38:17 WARN [main] DomainSocketFactory: The short-circuit local reads feature cannot be used because libhadoop cannot be loaded.
21/02/10 11:38:17 INFO [main] RMProxy: Connecting to ResourceManager at emr-header-1.cluster-2:8032
21/02/10 11:38:18 INFO [main] Client: Requesting a new application from cluster with 4 NodeManagers
21/02/10 11:38:18 INFO [main] Client: Verifying our application has not requested more than the maximum memory capability of the cluster (26144 MB per container)
21/02/10 11:38:18 INFO [main] Client: Will allocate AM container, with 3072 MB memory including 1024 MB overhead
21/02/10 11:38:18 INFO [main] Client: Setting up container launch context for our AM
21/02/10 11:38:18 INFO [main] Client: Setting up the launch environment for our AM container
21/02/10 11:38:18 INFO [main] Client: Preparing resources for our AM container
21/02/10 11:38:18 WARN [main] Client: Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading libraries under SPARK_HOME.
21/02/10 11:38:21 INFO [main] Client: Uploading resource file:/tmp/spark-bab232ea-978f-48f0-8378-36e97fb40866/__spark_libs__4172798070712495432.zip -> hdfs://emr-header-1.cluster-203701:9000/user/hadoop/.sparkStaging/application_1612885840821_0009/__spark_libs__4172798070712495432.zip
Warn: cannot load bigboot.cfg from B2SDK_CONF_DIR <null>
21/02/10 11:38:23 INFO [main] OssNativeStore: Filesystem support for magic committers is enabled, write buffer size 1048576
Took 25 sec. Last updated by anonymous at February 10 2021, 11:38:37 AM. (outdated)

6.1.5.2.6. TPCH和TPCDS

Zeppelin自带了TPCH和TPCDS的教程Note，您无需任何配置就可以在E-MapReduce（简称EMR）数据开发集群进行TPCH和TPCDS的性能测试。

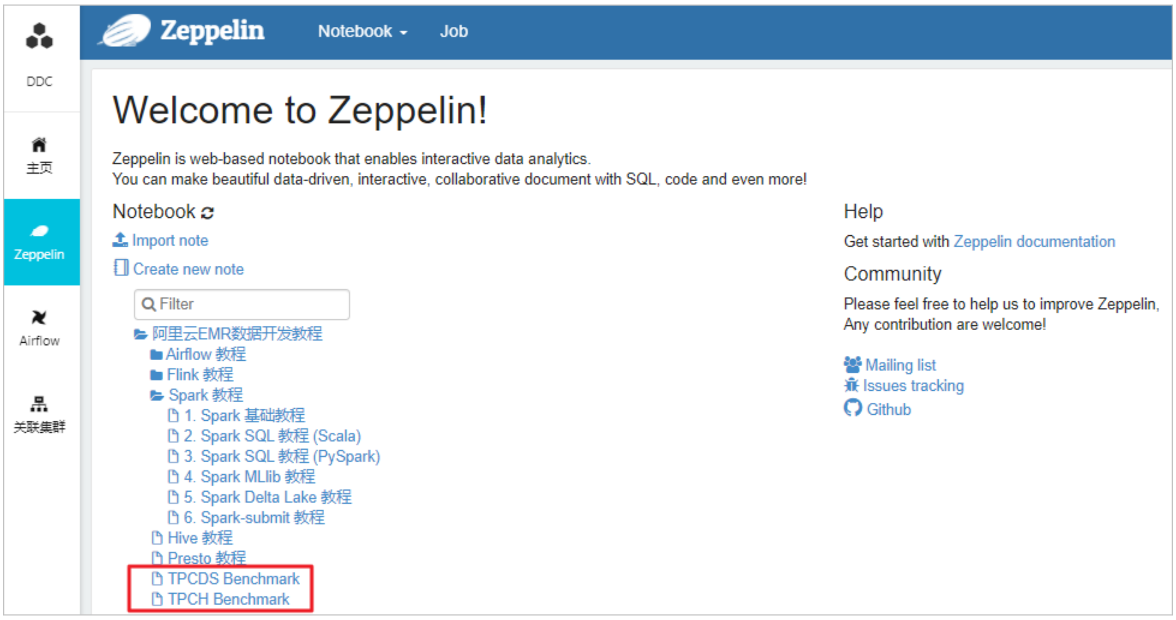
背景信息

EMR数据开发集群中的TPCH支持Hive、Spark、Flink和Presto四个引擎，TPCH详细信息，请参见TPC-H。

EMR数据开发集群中的TPCDS支持Hive、Spark和Flink三个引擎，TPCDS详情信息，请参见TPC-DS。

内置教程

EMR数据开发集群自带了很多教程，详细信息请在如下图页面查看。


DDC
主页
Zeppelin
Airflow
关联集群
Zeppelin
Notebook
Job
Welcome to Zeppelin!
Zeppelin is web-based notebook that enables interactive data analytics.
You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more!
Notebook
Import note
Create new note
Filter
阿里云EMR数据开发教程
Airflow 教程
Flink 教程
Spark 教程
1. Spark 基础教程
2. Spark SQL 教程 (Scala)
3. Spark SQL 教程 (PySpark)
4. Spark MLlib 教程
5. Spark Delta Lake 教程
6. Spark-submit 教程
Hive 教程
Presto 教程
TPCDS Benchmark
TPCH Benchmark
Help
Get started with Zeppelin documentation
Community
Please feel free to help us to improve Zeppelin,
Any contribution are welcome!
Mailing list
Issues tracking
Github

6.1.6. JupyterHub

6.1.6.1. 管理JupyterHub

JupyterHub是一个支持多用户的Notebook服务器，用于创建、管理和代理多个Jupyter Notebook实例。本文为您介绍如何访问JupyterHub的Web UI及JupyterHub的配置项信息。

前提条件

- 已创建EMR Studio集群，详情请参见创建集群。
- 已创建EMR Studio关联的集群，详情请参见创建集群。
- 安全组规则已开启8000和8443端口，详情请参见添加安全组规则。

访问JupyterHub UI

1. 进入详情页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
2. 添加Linux用户。
   您可以通过用户管理功能添加用户，详情请参见添加用户。
3. 在左侧导航栏中，选择**访问链接与端口**。
4. 单击**Studio Workspace UI**所在行的链接。
   输入步骤2中添加的用户名和密码，即可正常访问Web UI页面。

   > **说明** 请稍等几秒即可看到Studio Workspace UI的信息。

5. 关联计算集群。
   i. 在**关联集群**页面，单击**可关联集群**页签。
   ii. 选择待关联集群的集群类型。

   > **说明** 仅显示同一个VPC下的EMR集群。支持关联Hadoop集群、Dataflow集群和DataScience集群三种集群类型。

   iii. 单击待关联集群操作列的**关联集群**。
   iv. 在**关联集群**对话框中，单击**绑定**。
   待**已关联集群**页签，显示关联的集群信息时，表示关联成功。

   > **说明** 绑定集群过程大约需要1~2分钟，请您耐心等待。

6. 在EMR上构造Jupyter运行环境。
   i. 通过SSH方式登录集群，详情请参见登录集群。
   ii. 执行以下命令，构建默认运行环境。

   ```
   cd /var/lib/ecm-agent/cache/ecm/service/JUPYTERHUB/1.3.0.2.2/package/files && bash build_default_env.sh
   ```

   如果您需要在运行环境中安装特殊的软件包，则需要修改*/var/lib/ecm-agent/cache/ecm/service/JUPYTERHUB/1.3.0.2.1/package/files/emr-jupyterenv/*目录下的*requirements.yaml*文件，然后再执行上述命令。例如，如果需要在环境中安装TensorFlow，则需修改*requirements.yaml*文件内容如下。

   ```
   name: emr-jupyterenv
   channels:
     - conda-forge
     - defaults
   dependencies:
     - python<3.8
     - pandas
     - papermill
     - jupyterlab
     - conda-pack
     - pip:
       - ./jupyter-osscontents-manager
       - ./jupyterhub-yarnspawner
       - ./jupyter-emrmagic
   ```

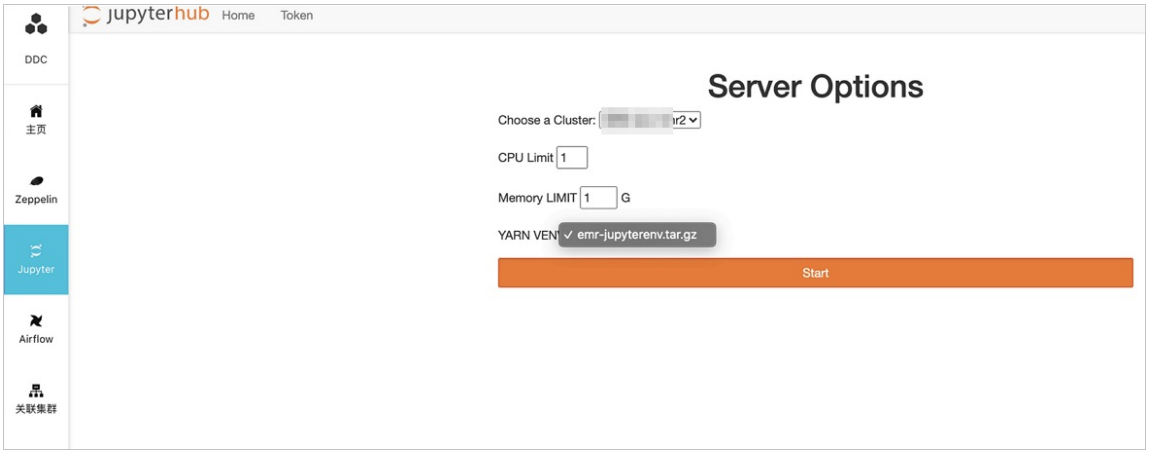
7. 访问Jupyter UI。
   i. 单击左侧导航栏的**Jupyter**。
   ii. 在**Sign in**对话框中，输入用户名和密码，单击**Sign in**。

   > **说明** 用户名和密码为步骤2中您添加的用户信息。

iii. 在Server Options页面，单击Start。

> **说明** 如果关联了多个集群，请选择关联的集群。首次启动Jupyter Server需要20秒左右，请您耐心等候。

在Server Options页面的YARN ENV中，即可看到构建的环境信息。



登录后，即可进入JupyterHub UI页面。

查看配置项

1. 进入详情页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
2. 在左侧导航栏，选择**集群服务 > JupyterHub**。
3. 在JupyterHub服务页面，单击**配置**页签。

   在**服务配置**区域，展示了JupyterHub服务所有的配置项。主要配置项信息如下。

> **注意** 如果您需要修改配置项信息，请参见修改组件参数。配置项修改完成后，需要重启JupyterHub服务，使配置生效。

参数	描述
notebook_root_path	Notebook在OSS中的存储路径。 默认值为*jupyter/notebook*。
venv_root_path	Jupyter运行环境在OSS中的存储路径。 默认值为*jupyter/venv*。 **说明** 仅对在EMR on ACK集群上运行的Jupyter生效。

6.1.6.2. 使用Python3 Kernel运行EMR PySpark

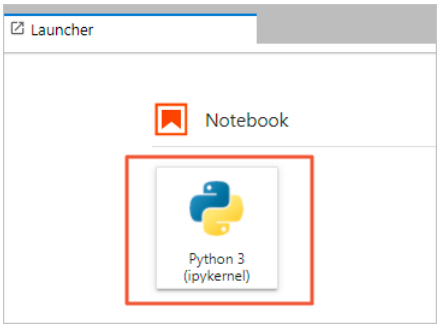
在E-MapReduce的JupyterHub中，您可以直接运行Python任务，也可以通过Python3 Kernel中的魔术命令PySpark和SQL运行任务。本文通过示例为您介绍如何运行Python3 Kernel任务，以及Python3 Kernel中的魔术命令PySpark和SQL中的参数。

前提条件

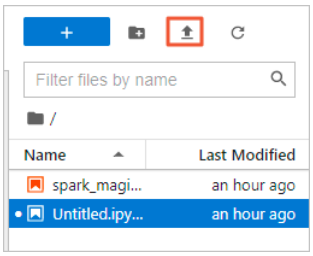
下载Notebook示例文件spark_magic_example.ipynb至本地目录。

操作步骤

1. 访问JupyterHub的Web UI，详情请参见管理JupyterHub。
2. 在JupyterHub的Web UI页面，单击Notebook下的Python3。



3. 单击 图标，上传Notebook示例文件。



在**spark_magic_example.ipynb**面板中，您可以查看提供的魔术命令PySpark和SQL。

- PySpark命令参数
  - `set_spark_opts`

    该参数用于设置Spark运行时参数，可以配置任何运行环境中Spark版本支持的参数。配置接收字典型参数，示例如下。

    ```
    conf = {
        'spark.executor.instances': '3',
        'spark.executor.memory': '2g',
    }
    %set_spark_opts $conf
    ```

  - `get_spark_opts`

    该参数用于展示Spark运行时参数。示例如下。

    ```
    %get_spark_opts
    ```

  - `get_spark_session`

    按照设置的Spark运行时参数，在EMR集群中启动Spark作业并获取Spark Session接。可以接收字符串类型参数，含义为Spark作业名称。示例如下。

    ```
    spark, sc = %get_spark_session test_application
    ```

    > 注意 每个Notebook只能建立一个Spark Session，重复构建时会获取已经生成的Spark Session。

  - `stop_spark_session`

    该参数用于终止Spark作业。

- SQL命令参数
  - `sql` 单元格中，可以编写多条SQL，使用分号（;）隔开。示例如下。

    ```
    %%sql
    create table if not exists test_table_1  (a string, b int);
    insert into test_table_1 values("abc", 1), ("def", 2);
    select a, sum(b) from test_table_1 group by a
    ```

  - `sql` 单元格中，可以使用 `-o` 参数，会将执行后的结果以pandas data frame模式传给指定变量，否则直接展示最后一条SQL的执行结果。示例如下。

    ```
    %%sql -o df  #将执行结果传给df变量。
    select a, sum(b) from test_table group by a
    ```

- `sql` 单元格中，可以定义变量。示例如下。

```
%%sql
select * from test_table where a = '{x}' # x在其它Cell中定义，通常用于Airflow调度程序。
```

4. 单击上方的 ▶ 图标，可以运行当前Cell。

6.2. 组件操作指南

6.2.1. Iceberg

6.2.1.1. Iceberg概述

Iceberg是一种开放的数据湖表格式。您可以借助Iceberg快速地在HDFS或者阿里云OSS上构建自己的数据湖存储服务，并借助开源大数据生态的Spark、Flink、Hive和Presto等计算引擎来实现数据湖的分析。

核心能力

Apache Iceberg设计初衷是为了解决Hive数仓上云的问题，经过多年迭代已经发展成为云上构建数据湖服务的表格式标准。关于Apache Iceberg的更多介绍，请参见Apache Iceberg官网。

目前Iceberg提供以下核心能力：

- 基于HDFS或者对象存储构建低成本的轻量级数据湖存储服务。
- 实现主流开源计算引擎入湖和分析场景的完善对接。
- 完善的ACID语义。
- 支持行级数据变更能力。
- 支持历史版本回溯。
- 支持高效的数据过滤。
- 支持Schema变更（Schema Evolution）。
- 支持分区布局变更（Partition Evolution）。
- 支持隐式分区（Hidden Partitioning）。

为了便于理解数仓和Iceberg数据湖在系统架构、业务价值和成本方面的差异，选择了业界流行的Clickhouse实时数仓、Hive离线数仓和Iceberg数据湖三种具体的技术架构，为您进行了对比，详细信息如下表。

对比项	子项目	开源Clickhouse实时数仓	开源Hive离线数仓	阿里云Iceberg数据湖
系统架构	架构	计算存储一体	计算存储分离	计算存储分离
	多计算引擎支持	不支持	支持	支持
	数据存储在对象存储	不支持	支持不完善	支持
	数据存储在HDFS	不支持	支持	支持
	存储格式开放性	不开放	开放	开放
业务价值	时效性	秒级	小时级/天级	分钟级
	计算灵活性	低	强	强
	事务性	不支持	不完善	支持
	表级语义通用性	差	差	优秀
	行级数据变更	不支持	支持较弱	支持
	数据质量	非常高	较高	较高
维护成本	查询性能	高	较高	较高
	存储成本	非常高	一般	低
	自助服务	不支持	不支持	支持
	资源弹性	一般	一般	优秀

与开源Iceberg对比

从基础功能、数据变更和计算引擎等方面，对比了阿里云Iceberg与开源Iceberg，详细信息如下表。

> **说明** “√”表示支持，“x”表示暂未支持。

类别	项目	子项目	开源Iceberg	Iceberg商业版（阿里云）
基础功能	ACID	无	√	√
	历史版本回溯	无	√	√
	Source和Sink集成	Batch	√	√
		Streaming	√	√
	高效数据过滤	无	√	√
数据变更	Schema Evolution	无	√	√
	Partition Evolution	无	√	√
	CopyOnWrite更新	无	√	√
	MergeOnRead更新	Read	√	√
		Write	√	√
		Compaction	x	x
计算引擎	Apache Spark	读取	√	√
		写入	√	√
	Apache Hive	读取	√	√
		写入	√	√
	Apache Flink	读取	√	√
		写入	√	√
	PrestoDB或Trino	读取	√	√
		写入	√	√
编程语言	Java	无	√	√
	Python	无	√	√
高级功能	原生接入阿里云OSS	无	x	√
	原生接入阿里云DLF	无	x	√
	本地数据缓存加速	无	x	√
	自动合并小文件	无	x	√

> **说明** 以上信息是在2021年9月份，客观分析开源Iceberg和商业版Iceberg现状之后制定的表格。随着后续版本的不断迭代升级，对比项状态可能发生变化。

适用场景

Iceberg作为通用数据湖解决方案中最核心的组件之一，主要适用于以下场景。

场景	描述
实时数据导入和查询	数据实时从上游流入Iceberg数据湖，查询侧即可查询该数据。例如，在日志场景中，启动Flink或Spark流作业，实时地将日志数据导入Iceberg表中，然后可以使用Hive、Spark、Flink或Presto进行实时查询。同时，由于Iceberg支持ACID，保证了数据的流入和查询的隔离性，不会产生脏数据。

场景	描述
删除或更新数据	大部分数仓都难以实现较为高效的行级数据删除或更新，通常需要启动离线作业把整个表原始数据读取出来，然后变更数据后，写入到一个原始表。而Iceberg成功把变更的范围从表级别缩小到了文件级别，从而可以通过局部变更来完成业务逻辑的数据变更或删除。 在Iceberg数据湖中，您可以直接通过执行类似命令 `DELETE FROM test_table WHERE id > 10` ，来完成表中数据的变更。
数据质量控制	借助于Iceberg Schema的校验功能，在数据导入时剔除异常数据，或者对异常数据做进一步处理。
数据Schema变更	数据的Schema并非固定不变，Iceberg支持通过Spark SQL的DDL语句完成表结构变更。Spark DDL详细信息，请参见Spark DDL。 Iceberg在变更表结构的时候，历史数据并不需要全部重新按照新的Schema导出一份，从而使得Schema变更的速度非常快。同时，由于Iceberg支持ACID，有效地隔离了Schema变更对现有读取任务的影响，从而使得您可以读取到结果一致的数据。
实时机器学习	通常在机器学习场景中，需要花费大量的时间处理数据，例如，数据清洗、转换和提取特征等，还需要对历史数据和实时数据进行处理。而Iceberg简化了工作流程，整条数据处理过程是一条完整的、可靠的实时流，其数据的清洗、转换和特征化等操作都是流上的节点动作，无需处理历史数据和实时数据。此外，Iceberg还支持原生的Python SDK，对于机器学习算法的开发者非常友好。

6.2.1.2. 基础使用

本文通过示例为您介绍如何在已经创建好的E-MapReduce（简称EMR）集群中使用Iceberg。

背景信息

本文以数据湖元数据为例，详细配置请参见数据湖元数据配置。

前提条件

已在E-MapReduce控制台上，创建Hadoop的EMR-5.3.0及后续版本的集群，详情请参见创建集群。

使用限制

由于Iceberg的Spark SQL Extensions不适用于Spark 2.4，因此对于EMR-3.38.x及后续版本的Spark只能使用DataFrame API操作Iceberg。本文介绍EMR-5.3.0及后续版本以Spark SQL方式操作Iceberg。

操作步骤

1. 使用SSH方式登录到集群，详情信息请参见登录集群。
2. 执行以下命令，通过Spark SQL读写Iceberg配置。

   在Spark SQL中操作Iceberg，首先需要配置Catalog。Catalog的配置以spark.sql.catalog.<catalog_name>作为前缀。

   以下是在Spark SQL中使用数据湖元数据的配置，集群版本不同默认的Catalog名称不同，需要配置的参数也不同，具体请参见数据湖元数据配置。

   - EMR-5.6.0及后续版本

     ```
     spark-sql --conf spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions \
      --conf spark.sql.catalog.iceberg=org.apache.iceberg.spark.SparkCatalog \
      --conf spark.sql.catalog.iceberg.catalog-impl=org.apache.iceberg.aliyun.dlf.hive.DlfCatalog \
     ```

   - EMR-5.5.x版本

     ```
     spark-sql --conf spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions \
      --conf spark.sql.catalog.dlf=org.apache.iceberg.spark.SparkCatalog \
      --conf spark.sql.catalog.dlf.catalog-impl=org.apache.iceberg.aliyun.dlf.hive.DlfCatalog \
      --conf spark.sql.catalog.dlf.warehouse=<yourOSSWarehousePath> \
     ```

     > **说明** spark.sql.catalog.dlf.warehouse参数可以不设置。如果不设置spark.sql.catalog.dlf.warehouse，则默认使用HDFS的warehouse。

   - EMR-5.3.x~EMR-5.4.x版本（包含）

```
spark-sql --conf spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions \
 --conf spark.sql.catalog.dlf_catalog=org.apache.iceberg.spark.SparkCatalog \
 --conf spark.sql.catalog.dlf_catalog.catalog-impl=org.apache.iceberg.aliyun.dlf.DlfCatalog \
 --conf spark.sql.catalog.dlf_catalog.io-impl=org.apache.iceberg.hadoop.HadoopFileIO \
 --conf spark.sql.catalog.dlf_catalog.oss.endpoint=<yourOSSEndpoint> \
 --conf spark.sql.catalog.dlf_catalog.warehouse=<yourOSSWarehousePath> \
 --conf spark.sql.catalog.dlf_catalog.access.key.id=<yourAccessKeyId> \
 --conf spark.sql.catalog.dlf_catalog.access.key.secret=<yourAccessKeySecret> \
 --conf spark.sql.catalog.dlf_catalog.dlf.catalog-id=<yourCatalogId> \
 --conf spark.sql.catalog.dlf_catalog.dlf.endpoint=<yourDLFEndpoint> \
 --conf spark.sql.catalog.dlf_catalog.dlf.region-id=<yourDLFRegionId>
```

当返回信息中包含如下信息时，表示已进入spark-sql命令行。

```
spark-sql>
```

3. 基本操作。

> **注意** 以下示例中的 `<yourCatalogName>` 为您Catalog的名称，请您根据实际信息修改Catalog名称。

- 创建库

```
CREATE DATABASE IF NOT EXISTS <yourCatalogName>.iceberg_db;
```

- 创建表

```
CREATE TABLE IF NOT EXISTS <yourCatalogName>.iceberg_db.sample(
    id BIGINT COMMENT 'unique id',
    data STRING
)
USING iceberg;
```

Iceberg表支持COMMENT、PARTITIONED BY、LOCATION和TBLPROPERTIES等语法。如果通过TBLPROPERTIES设置表级别属性，代码示例如下。

```
CREATE TABLE IF NOT EXISTS <yourCatalogName>.iceberg_db.sample(
    id BIGINT COMMENT 'unique id',
    data STRING
)
USING iceberg
TBLPROPERTIES (
    'write.format.default'='parquet'
);
```

- 写入数据

```
INSERT INTO <yourCatalogName>.iceberg_db.sample VALUES (1, 'a'), (2, 'b'), (3, 'c');
```

- 查询数据

```
SELECT * FROM <yourCatalogName>.iceberg_db.sample;
SELECT count(1) AS count, data FROM dlf_catalog.iceberg_db.sample GROUP BY data;
```

- 更新数据

```
UPDATE <yourCatalogName>.iceberg_db.sample SET data = 'x' WHERE id = 3;
```

- 删除数据

```
DELETE FROM <yourCatalogName>.iceberg_db.sample WHERE id = 3;
```

6.2.1.3. 开发指南

6.2.1.3.1. Spark批式读写Iceberg

本文以Spark 3.x操作Iceberg表为例，介绍如何通过Spark DataFrame API以批处理的方式读写Iceberg表。

前提条件

已创建Hadoop集群，详情请参见创建集群。

> **说明** 此文档仅适用于EMR-3.38.0及后续版本与EMR-5.4.0及后续版本的Hadoop集群。

操作步骤

1. 新建Maven项目，引入Pom依赖。

   引入Spark及Iceberg的依赖，以下代码示例指定了Spark 3.1.1与Iceberg 0.12.0版本，使用provided引包编译，运行时使用集群上的软件包。

   ```
   <dependency>
       <groupId>org.apache.spark</groupId>
       <artifactId>spark-core_2.12</artifactId>
       <version>3.1.1</version>
       <scope>provided</scope>
   </dependency>
   <dependency>
       <groupId>org.apache.spark</groupId>
       <artifactId>spark-sql_2.12</artifactId>
       <version>3.1.1</version>
       <scope>provided</scope>
   </dependency>
   <dependency>
       <groupId>org.apache.iceberg</groupId>
       <artifactId>iceberg-core</artifactId>
       <version>0.12.0</version>
       <scope>provided</scope>
   </dependency>
   ```

   > **说明** 由于EMR集群的Iceberg软件包与开源依赖包存在一定差异，例如EMR Iceberg默认集成了DLF Catalog，所以建议您在本地使用provided方式引入开源Iceberg依赖进行代码编译，打包放到集群上运行时使用集群环境中的依赖。

2. 配置Catalog。

   使用Spark API操作Iceberg表，首先需要配置Catalog，在SparkConf中加入必要配置项即可。

   以下是在Spark SQL中使用数据湖元数据的配置，集群版本不同默认的Catalog名称不同，需要配置的参数也不同，具体请参见数据湖元数据配置。

   - EMR-3.40及后续版本和EMR-5.6.0及后续版本

     ```
     sparkConf.set("spark.sql.extensions", "org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions")
     sparkConf.set("spark.sql.catalog.iceberg", "org.apache.iceberg.spark.SparkCatalog")
     sparkConf.set("spark.sql.catalog.iceberg.catalog-impl", "org.apache.iceberg.aliyun.dlf.hive.DlfCatalog")
     ```

   - EMR-3.39.x和EMR-5.5.x版本

     ```
     sparkConf.set("spark.sql.extensions", "org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions")
     sparkConf.set("spark.sql.catalog.dlf", "org.apache.iceberg.spark.SparkCatalog")
     sparkConf.set("spark.sql.catalog.dlf.catalog-impl", "org.apache.iceberg.aliyun.dlf.hive.DlfCatalog")
     sparkConf.set("spark.sql.catalog.dlf.warehouse", "<yourOSSWarehousePath>")
     ```

   - EMR-3.38.x版本和EMR-5.3.x~EMR-5.4.x版本（包含）

     ```
     sparkConf.set("spark.sql.extensions", "org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions")
     sparkConf.set("spark.sql.catalog.dlf_catalog", "org.apache.iceberg.spark.SparkCatalog")
     sparkConf.set("spark.sql.catalog.dlf_catalog.catalog-impl", "org.apache.iceberg.aliyun.dlf.DlfCatalog")
     sparkConf.set("spark.sql.catalog.dlf_catalog.io-impl", "org.apache.iceberg.hadoop.HadoopFileIO")
     sparkConf.set("spark.sql.catalog.dlf_catalog.oss.endpoint", "<yourOSSEndpoint>")
     sparkConf.set("spark.sql.catalog.dlf_catalog.warehouse", "<yourOSSWarehousePath>")
     sparkConf.set("spark.sql.catalog.dlf_catalog.access.key.id", "<yourAccessKeyId>")
     sparkConf.set("spark.sql.catalog.dlf_catalog.access.key.secret", "<yourAccessKeySecret>")
     sparkConf.set("spark.sql.catalog.dlf_catalog.dlf.catalog-id", "<yourCatalogId>")
     sparkConf.set("spark.sql.catalog.dlf_catalog.dlf.endpoint", "<yourDLFEndpoint>")
     sparkConf.set("spark.sql.catalog.dlf_catalog.dlf.region-id", "<yourDLFRegionId>")
     ```

3. 写表。

   Spark 3.x支持DataFrameWriterV2 API写入数据到Iceberg表。目前v1 DataFrame API已不推荐使用，以下代码以V2 API写Iceberg表sample为例。

   以下示例中的 `<yourCatalogName>` 为Catalog的名称，请根据实际情况修改Catalog名称。

   创建数据表

   ```
   val df: DataFrame = ...
   df.writeTo("<yourCatalogName>.iceberg_db.sample").create()
   ```

> **说明** 创建表支持create、replace以及createOrReplace语义，另外支持通过tableProperty和partitionedBy配置表的属性与分区字段。

您可以通过以下命令追加或覆盖数据：

- 追加数据

```
val df: DataFrame = ...
df.writeTo("<yourCatalogName>.iceberg_db.sample").append()
```

- 覆盖数据

```
val df: DataFrame = ...
df.writeTo("<yourCatalogName>.iceberg_db.sample").overwritePartitions()
```

4. 读表。

   请根据您Spark的版本，选择读表的方式：

   - Spark 3.x（推荐）

```
 val df = spark.table("<yourCatalogName>.iceberg_db.sample")
```

   - Spark 2.4

```
val df = spark.read.format("iceberg").load("<yourCatalogName>.iceberg_db.sample")
```

示例

本示例是使用Spark DataFrame API批式读写Iceberg表。

> **注意** 示例中数据湖元数据的配置参数，根据集群版本不同，配置的参数不同，默认的Catalog名称也不同。本示例以EMR-5.3.0版本为列，其中 `dlf_catalog` 为Catalog名称。具体版本对应的配置请参见数据湖元数据配置。

1. 通过Spark SQL创建测试使用的数据库iceberg_db，详细信息请参见基础使用。
2. 编写Spark代码。

   以Scala版代码为例，代码示例如下。

```
def main(args: Array[String]): Unit = {
  // 配置使用数据湖元数据。
  val sparkConf = new SparkConf()
  sparkConf.set("spark.sql.extensions", "org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions")
  sparkConf.set("spark.sql.catalog.dlf_catalog", "org.apache.iceberg.spark.SparkCatalog")
  sparkConf.set("spark.sql.catalog.dlf_catalog.catalog-impl", "org.apache.iceberg.aliyun.dlf.DlfCatalog")
  sparkConf.set("spark.sql.catalog.dlf_catalog.io-impl", "org.apache.iceberg.hadoop.HadoopFileIO")
  sparkConf.set("spark.sql.catalog.dlf_catalog.oss.endpoint", "<yourOSSEndpoint>")
  sparkConf.set("spark.sql.catalog.dlf_catalog.warehouse", "<yourOSSWarehousePath>")
  sparkConf.set("spark.sql.catalog.dlf_catalog.access.key.id", "<yourAccessKeyId>")
  sparkConf.set("spark.sql.catalog.dlf_catalog.access.key.secret", "<yourAccessKeySecret>")
  sparkConf.set("spark.sql.catalog.dlf_catalog.dlf.catalog-id", "<yourCatalogId>")
  sparkConf.set("spark.sql.catalog.dlf_catalog.dlf.endpoint", "<yourDLFEndpoint>")
  sparkConf.set("spark.sql.catalog.dlf_catalog.dlf.region-id", "<yourDLFRegionId>")
  val spark = SparkSession
  .builder()
  .config(sparkConf)
  .appName("IcebergReadWriteTest")
  .getOrCreate()
  // 从DataFrame中创建或替换Iceberg表
  val firstDF = spark.createDataFrame(Seq(
  (1, "a"), (2, "b"), (3, "c")
  )).toDF("id", "data")
  firstDF.writeTo("dlf_catalog.iceberg_db.sample").createOrReplace()
  // 将DataFrame写入Iceberg表
  val secondDF = spark.createDataFrame(Seq(
  (4, "d"), (5, "e"), (6, "f")
  )).toDF("id", "data")
  secondDF.writeTo("dlf_catalog.iceberg_db.sample").append()
  // 读Iceberg表
  val icebergTable = spark.table("dlf_catalog.iceberg_db.sample")
  icebergTable.show()
}
```

3. 打包程序并部署到EMR集群。

i. 检查编译Scala代码的Maven插件，可以在*pom.xml*中配置如下插件。

```
<build>
    <plugins>
        <!-- the Maven Scala plugin will compile Scala source files -->
        <plugin>
            <groupId>net.alchim31.maven</groupId>
            <artifactId>scala-maven-plugin</artifactId>
            <version>3.2.2</version>
            <executions>
                <execution>
                    <goals>
                        <goal>compile</goal>
                        <goal>testCompile</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

ii. 您可以在本地完成代码调试后，通过如下命令打包。

```
mvn clean install
```

iii. 使用SSH方式登录到集群，详情信息请参见登录集群。

iv. 上传JAR包至EMR集群。

本示例是上传到EMR集群的根目录下。

4. 执行以下命令，通过spark-submit运行Spark作业。

```
spark-submit \
 --master yarn \
 --deploy-mode cluster \
 --driver-memory 1g \
 --executor-cores 1 \
 --executor-memory 1g \
 --num-executors 1 \
 --class com.aliyun.iceberg.IcebergTest \
 iceberg-demos.jar
```

> **说明** iceberg-demos.jar为您打包好的JAR包。--class和JAR包请根据您实际信息修改。

运行结果如下。

```
+---+----+
| id|data|
+---+----+
|  4|   d|
|  1|   a|
|  5|   e|
|  6|   f|
|  2|   b|
|  3|   c|
+---+----+
```

6.2.1.3.2. Spark流式写入Iceberg

本文为您介绍如何通过Spark Structured Streaming流式写入Iceberg表。

前提条件

- 已在E-MapReduce控制台上，创建Hadoop集群，详情请参见创建集群。

> **说明** 此文档仅适用于EMR-3.38.0及后续版本与EMR-5.4.0及后续版本的Hadoop集群。

- 已在E-MapReduce控制台上，创建Kafka集群，详情请参见创建集群。

使用限制

Hadoop集群和Kafka集群需要在同一VPC和交换机下，不支持跨VPC。

流式写入方式

Spark Structured Streaming通过DataStreamWriter接口流式写数据到Iceberg表，代码如下。

```
val tableIdentifier: String = ...
data.writeStream
    .format("iceberg")
    .outputMode("append")
    .trigger(Trigger.ProcessingTime(1, TimeUnit.MINUTES))
    .option("path", tableIdentifier)
    .option("checkpointLocation", checkpointPath)
    .start()
```

> **说明** 代码中的tableIdentifier是元数据表名或者表路径。流式写入支持以下两种方式：
>
> - append：追加每个批次的数据到Iceberg表，相当于insert into。
> - complete：使用最新批次的数据完全覆盖Iceberg，相当于insert overwrite。

示例

本示例是从上游Kafka中读取数据，写入Iceberg表，打包放到EMR集群上通过spark-submit提交执行。

1. 通过Kafka脚本创建测试使用的topic并准备测试数据。
   i. 使用SSH方式登录到Kafka集群，详情信息请参见登录集群。
   ii. 执行以下命令，创建名为iceberg_test的topic。

   ```
   kafka-topics.sh --zookeeper emr-header-1:2181,emr-worker-1:2181,emr-worker-2:2181 --topic iceberg_test --partitions 3 --replication-factor 2 --create
   ```

   iii. 执行以下命令，生产测试数据。

   ```
   kafka-console-producer.sh --broker-list emr-header-1:9092,emr-worker-1:9092,emr-worker-2:9092 --topic iceberg_test
   ```

2. 通过Spark SQL创建测试使用的数据库iceberg_db和表iceberg_table，详细操作请参见基础使用。
3. 编写Spark代码。

   以Scala版代码为例，代码示例如下。

   > **注意** 示例中数据湖元数据的配置参数，根据集群版本不同，配置的参数不同，Catalog名称也不同。本示例以EMR-5.3.0版本为列，其中 `dlf_catalog` 为Catalog名称。具体版本对应的配置请参见数据湖元数据配置。

```
def main(args: Array[String]): Unit = {
  // 配置使用数据湖元数据。
  val sparkConf = new SparkConf()
  sparkConf.set("spark.sql.extensions", "org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions")
  sparkConf.set("spark.sql.catalog.dlf_catalog", "org.apache.iceberg.spark.SparkCatalog")
  sparkConf.set("spark.sql.catalog.dlf_catalog.catalog-impl", "org.apache.iceberg.aliyun.dlf.DlfCatalog")
  sparkConf.set("spark.sql.catalog.dlf_catalog.io-impl", "org.apache.iceberg.hadoop.HadoopFileIO")
  sparkConf.set("spark.sql.catalog.dlf_catalog.oss.endpoint", "<yourOSSEndpoint>")
  sparkConf.set("spark.sql.catalog.dlf_catalog.warehouse", "<yourOSSWarehousePath>")
  sparkConf.set("spark.sql.catalog.dlf_catalog.access.key.id", "<yourAccessKeyId>")
  sparkConf.set("spark.sql.catalog.dlf_catalog.access.key.secret", "<yourAccessKeySecret>")
  sparkConf.set("spark.sql.catalog.dlf_catalog.dlf.catalog-id", "<yourCatalogId>")
  sparkConf.set("spark.sql.catalog.dlf_catalog.dlf.endpoint", "<yourDLFEndpoint>")
  sparkConf.set("spark.sql.catalog.dlf_catalog.dlf.region-id", "<yourDLFRegionId>")
  val spark = SparkSession
    .builder()
    .config(sparkConf)
    .appName("StructuredSinkIceberg")
    .getOrCreate()
  val checkpointPath = "oss://mybucket/tmp/iceberg_table_checkpoint"
  val bootstrapServers = "192.168.XX.XX:9092"
  val topic = "iceberg_test"
  // 从上游Kafka读取数据
  val df = spark.readStream
    .format("kafka")
    .option("kafka.bootstrap.servers", bootstrapServers)
    .option("subscribe", topic)
    .load()
  import spark.implicits._
  val resDF = df.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")
    .as[(String, String)].toDF("id", "data")
  // 流式写入Iceberg表
  val query = resDF.writeStream
    .format("iceberg")
    .outputMode("append")
    .trigger(Trigger.ProcessingTime(1, TimeUnit.MINUTES))
    .option("path", "dlf_catalog.iceberg_db.iceberg_table")
    .option("checkpointLocation", checkpointPath)
    .start()
  query.awaitTermination()
}
```

请您根据集群的实际情况，修改如下参数。

参数	描述
checkpointPath	Spark流式写数据的Checkpoint路径。
bootstrapServers	Kafka集群中任一Kafka Broker组件的内网IP地址。
topic	Topic名称。

4. 打包程序并部署到EMR集群。

i. 检查编译Scala代码的Maven插件，可以在*pom.xml*中配置如下插件。

```
<build>
    <plugins>
        <!-- the Maven Scala plugin will compile Scala source files -->
        <plugin>
            <groupId>net.alchim31.maven</groupId>
            <artifactId>scala-maven-plugin</artifactId>
            <version>3.2.2</version>
            <executions>
                <execution>
                    <goals>
                        <goal>compile</goal>
                        <goal>testCompile</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

ii. 本地调试完成后，通过以下命令打包。

```
mvn clean install
```

iii. 使用SSH方式登录到集群，详情信息请参见登录集群。

iv. 上传JAR包至EMR集群。

本示例是上传到EMR集群的根目录下。

5. 提交运行Spark作业。

i. 执行以下命令，通过spark-submit提交Spark作业。

```
spark-submit \
 --master yarn \
 --deploy-mode cluster \
 --driver-memory 1g \
 --executor-cores 2 \
 --executor-memory 3g \
 --num-executors 1 \
 --class com.aliyun.iceberg.StructuredSinkIceberg \
 iceberg-demos.jar
```

> **说明** iceberg-demos.jar为您打包好的JAR包。--class和JAR包请根据您实际信息修改。

ii. 通过Spark SQL查询数据的变化，详细操作请参见基础使用。

6.2.1.3.3. Hive访问Iceberg数据

Hive支持通过内表或外表的方式访问Iceberg数据。本文通过示例为您介绍如何使用EMR上的Hive访问EMR Iceberg数据。

前提条件

已创建Hadoop集群，详情请参见创建集群。

> **说明** 此文档仅适用于EMR-3.38.0及后续版本与EMR-5.4.0及后续版本的Hadoop集群。

使用限制

EMR-3.38.0及后续版本与EMR-5.4.0及后续版本的Hadoop集群，支持Hive读写Iceberg的数据。

操作步骤

1. （可选）如果您创建的是EMR-3.38.0与EMR-5.4.0版本的集群，则需要修改以下配置项。

因为EMR-3.38.0与EMR-5.4.0版本的Hive与Iceberg集成存在一定兼容性问题，所以需要修改以下配置。

i. 进入Hive页面。

a. 登录阿里云E-MapReduce控制台。

b. 在顶部菜单栏处，根据实际情况选择地域和资源组。

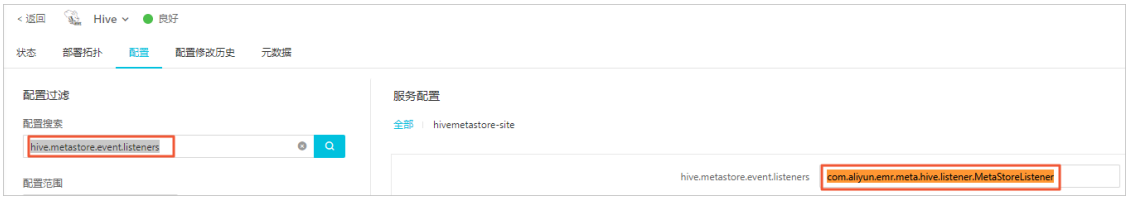
c. 单击上方的**集群管理**页签。

d. 在**集群管理**页面，单击相应集群所在行的**详情**。

e. 在左侧导航栏，单击**集群服务 > Hive**。

ii. 根据您创建集群的版本，修改配置信息。

- EMR-3.38.0版本：在Hive服务的配置页面，搜索参数**hive.metastore.event.listeners**，删除参数值。



- EMR-5.4.0版本：在Hive服务的配置页面，搜索参数**metastore.event.listeners**，删除参数值。

iii. 保存配置。

a. 单击右上角的**保存**。

b. 在**确认修改**对话框中，输入**执行原因**，单击**确定**。

iv. 重启Hive服务，详情请参见重启服务。

2. 进入Hive命令行。

i. 使用SSH方式登录到集群主节点，详情请参见登录集群。

ii. 执行以下命令，进入Hive命令行。

```
hive
```

返回信息如下所示时，表示进入Hive命令行。

```
Logging initialized using configuration in file:/etc/ecm/hive-conf-2.3.5-2.0.3/hive-log4j2.properties Async: true
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different exec
ution engine (i.e. spark, tez) or using Hive 1.X releases.
```

3. 创建表。

- 如果创建集群时，**元数据选择**设置为**数据湖元数据**，则可以按照以下步骤操作。

使用数据湖元数据作为Hive元数据。该场景下，Hive只支持以外部表的方式访问Iceberg表。

a. 根据您实际的集群版本，设置Hive接入Iceberg使用数据湖元数据的必要配置。

- EMR-3.39.0及后续版本和EMR-5.5.0及后续版本

```
SET iceberg.catalog.dlf.catalog-impl=org.apache.iceberg.aliyun.dlf.hive.DlfCatalog;
```

- EMR-3.38.x版本和EMR-5.3.x~EMR-5.4.x版本（包含）

```
SET iceberg.catalog=dlf_catalog;
SET iceberg.catalog.dlf_catalog.type=custom;
SET iceberg.catalog.dlf_catalog.io-impl=org.apache.iceberg.hadoop.HadoopFileIO;
SET iceberg.catalog.dlf_catalog.catalog-impl=org.apache.iceberg.aliyun.dlf.DlfCatalog;
SET iceberg.catalog.dlf_catalog.access.key.id=<yourAccessKeyId>;
SET iceberg.catalog.dlf_catalog.access.key.secret=<yourAccessKeySecret>;
SET iceberg.catalog.dlf_catalog.warehouse=<yourOSSWarehousePath>
SET iceberg.catalog.dlf_catalog.dlf.catalog-id=<yourCatalogId>;
SET iceberg.catalog.dlf_catalog.dlf.endpoint=<yourDLFEndpoint>;
SET iceberg.catalog.dlf_catalog.dlf.region-id=<yourDLFRegionId>;
```

> **说明** 示例中dlf_catalog为您创建的Catalog名称，warehouse配置使用阿里云对象存储OSS路径，其余参数含义请参见数据湖元数据配置。

b. 创建Hive外部表，映射已经存在的iceberg_db.sample表。

```
CREATE EXTERNAL TABLE iceberg_db.sample_tbl
STORED BY 'org.apache.iceberg.mr.hive.HiveIcebergStorageHandler'
LOCATION 'oss://some_bucket/some_path/sample'
TBLPROPERTIES (
  'iceberg.catalog'='<yourCatalogName>',
  'name'='iceberg_db.sample'
);
```

> **说明** 示例中参数：
>
> - `<yourCatalogName>`：Catalog的名称，不同集群版本的Catalog名称有差异，具体请根据数据湖元数据配置修改Catalog名称。
> - `iceberg_db.sample`：是一张已存在的表，您可以提前创建好，详情请参见基础使用。

- 如果创建集群时，**元数据选择**设置为**集群内置MySQL**或**独立RDS MySQL**，则可以按照以下步骤操作。
  - 使用默认Catalog时，执行以下命令，创建表。

```
CREATE TABLE iceberg_db.sample_tbl (
  id BIGINT,
  name STRING
)
STORED BY 'org.apache.iceberg.mr.hive.HiveIcebergStorageHandler';
```

  - 自定义Catalog时，执行以下命令。

> **说明** Iceberg支持Hive类型、Hadoop类型或其他限定的自定义Catalog，本文档以Hive类型为例。

a. 设置Hive接入Iceberg的配置。

```
SET iceberg.catalog.hive_catalog.type=hive;
```

b. 创建Hive表。

```
CREATE TABLE iceberg_db.sample_tbl (
  id bigint,
  name string
)
STORED BY 'org.apache.iceberg.mr.hive.HiveIcebergStorageHandler'
TBLPROPERTIES ('iceberg.catalog'='hive_catalog');
```

4. 执行以下命令，向表中写入数据。

```
INSERT INTO iceberg_db.sample_tbl VALUES (4, 'd'), (5, 'e'), (6, 'f');
```

> **说明** 如果写数据时，遇到异常提示*return code 2 from org.apache.hadoop.hive.ql.exec.mr.MapRedTask*，应该找不到fb303相关类路径，可以在Hive命令行中手动执行命令 `add jar /usr/lib/hive-current/lib/libfb303-0.9.3.jar` 。

5. 执行以下命令，查看表数据。

```
SELECT * FROM iceberg_db.sample_tbl;
```

示例1

本示例使用数据湖元数据，通过Hive外部表的方式对一张已有的Iceberg表进行读写操作。

1. 创建一个EMR-5.4.0的Hadoop集群，元数据选择数据湖元数据，详情请参见创建集群。
2. 修改配置项，详情请参见操作步骤中的步骤1。
3. 进入Hive命令行，详情请参见操作步骤中的步骤2。
4. 根据您实际信息替换以下配置，设置Hive接入Iceberg使用数据湖元数据的必要配置。

```
SET iceberg.catalog=dlf_catalog;
SET iceberg.catalog.dlf_catalog.type=custom;
SET iceberg.catalog.dlf_catalog.io-impl=org.apache.iceberg.hadoop.HadoopFileIO;
SET iceberg.catalog.dlf_catalog.catalog-impl=org.apache.iceberg.aliyun.dlf.DlfCatalog;
SET iceberg.catalog.dlf_catalog.access.key.id=<yourAccessKeyId>;
SET iceberg.catalog.dlf_catalog.access.key.secret=<yourAccessKeySecret>;
SET iceberg.catalog.dlf_catalog.warehouse=<yourOSSWarehousePath>
SET iceberg.catalog.dlf_catalog.dlf.catalog-id=<yourCatalogId>;
SET iceberg.catalog.dlf_catalog.dlf.endpoint=<yourDLFEndpoint>;
SET iceberg.catalog.dlf_catalog.dlf.region-id=<yourDLFRegionId>;
```

> **说明** 示例中dlf_catalog为默认的Catalog名称，warehouse配置使用阿里云对象存储OSS路径，其余参数含义请参见数据湖元数据配置。

5. 执行以下命令，创建数据表iceberg_db.sample_ext，映射已有的iceberg_db.sample表。

```
CREATE EXTERNAL TABLE iceberg_db.sample_ext
STORED BY 'org.apache.iceberg.mr.hive.HiveIcebergStorageHandler'
LOCATION 'oss://mybucket/iceberg-test/warehouse/iceberg_db.db/sample'
TBLPROPERTIES (
    'iceberg.catalog'='dlf_catalog',
     'name'='iceberg_db.sample'
   );
```

6. 执行以下命令，通过外部表查询Iceberg表数据。

```
SELECT * FROM iceberg_db.sample_ext;
```

返回信息如下。

```
OK
1 a
2 b
3 c
Time taken: 19.075 seconds, Fetched: 3 row(s)
```

7. 执行以下命令，向表中写入数据。

```
INSERT INTO iceberg_db.sample_ext VALUES (4, 'd'), (5, 'e'), (6, 'f');
```

8. 执行以下命令，查询Iceberg表数据。

```
SELECT * FROM iceberg_db.sample_ext;
```

返回信息如下。

```
OK
1    a
2    b
3    c
4    d
5    e
6    f
Time taken: 18.908 seconds, Fetched: 6 row(s)
```

示例2

本示例使用Hive默认元数据，创建一张格式为Iceberg的Hive内表并对其进行读写操作。

1. 创建一个EMR-5.4.0的Hadoop集群，元数据选择集群内置MySQL，详情请参见创建集群。
2. 修改配置项，详情请参见操作步骤中的步骤1。
3. 进入Hive命令行，详情请参见操作步骤中的步骤2。
4. 执行以下命令，创建数据库iceberg_db。

```
CREATE DATABASE IF NOT EXISTS iceberg_db;
```

5. 执行以下命令，创建数据表sample_tbl。

```
CREATE TABLE iceberg_db.sample_tbl (
  id BIGINT,
  name STRING
)
STORED BY 'org.apache.iceberg.mr.hive.HiveIcebergStorageHandler';
```

6. 执行以下命令，向表中写入数据。

```
INSERT INTO iceberg_db.sample_tbl VALUES (1, 'a'), (2, 'b'), (3, 'c');
```

7. 执行以下命令，查询Iceberg表数据。

```
SELECT * FROM iceberg_db.sample_tbl;
```

返回信息如下。

```
OK
1    a
2    b
3    c
Time taken: 0.233 seconds, Fetched: 3 row(s)
```

相关文档

- Iceberg相关的介绍，请参见Iceberg概述。
- Iceberg表使用数据湖元数据的必要配置，请参见数据湖元数据配置。

6.2.1.3.4. 数据湖元数据配置

本文为您介绍Iceberg表使用数据湖元数据的必要配置。

支持以下配置：

- Spark配置
- Hive配置

Spark配置

文件系统采用阿里云对象存储服务OSS。集群版本不同默认的Catalog名称不同，需要配置的参数也不同，各版本对应配置如下：

- **EMR-3.40及后续版本和EMR-5.6.0及后续版本**

> **说明** 默认的Catalog名称为iceberg。

参数	描述	备注
spark.sql.extensions	Spark SQL扩展模块。	固定值为org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions。 **说明** Iceberg 0.11.0开始引入，仅Spark 3.x支持。
spark.sql.catalog.iceberg<catalog-name>	Catalog名称。	固定值为org.apache.iceberg.spark.SparkCatalog。
spark.sql.catalog.<catalog-name>.catalog-impl	Catalog的Class类名。	固定值为org.apache.iceberg.aliyun.dlf.hive.DlfCatalog。

- **EMR-3.39.x和EMR-5.5.x版本**

> **说明** 默认的Catalog名称为dlf。

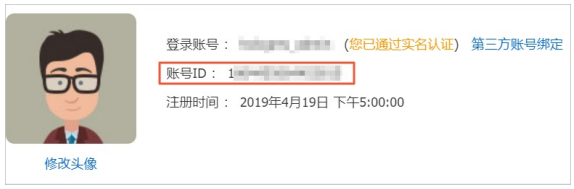
参数	描述	备注

参数	描述	备注
spark.sql.extensions	Spark SQL扩展模块。	固定值为org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions。 **说明** Iceberg 0.11.0开始引入，仅Spark 3.x支持。
spark.sql.catalog.<catalog-name>	Catalog名称。	固定值为org.apache.iceberg.spark.SparkCatalog。
spark.sql.catalog.<catalog-name>.catalog-impl	Catalog的Class类名。	固定值为org.apache.iceberg.aliyun.dlf.hive.DlfCatalog。

- **EMR-3.38.x版本和EMR-5.3.x~EMR-5.4.x版本（包含）**

说明 默认的Catalog名称为dlf_catalog。

参数	描述	备注
spark.sql.extensions	Spark SQL扩展模块。	固定值为org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions。 **说明** Iceberg 0.11.0开始引入，仅Spark 3.x支持。
spark.sql.catalog.<catalog-name>	Catalog名称。	固定值为org.apache.iceberg.spark.SparkCatalog。
spark.sql.catalog.<catalog-name>.catalog-impl	Catalog的Class类名。	固定值为org.apache.iceberg.aliyun.dlf.DlfCatalog。
spark.sql.catalog.<catalog-name>.io-impl	IO写入的Class名称。	固定值为org.apache.iceberg.hadoop.HadoopFileIO。
spark.sql.catalog.<catalog-name>.oss.endpoint	阿里云对象存储服务OSS的Endpoint。	请详情参见访问域名和数据中心。 推荐您为oss.endpoint参数配置OSS的VPC Endpoint。例如，如果您选择的地域为cn-hangzhou地域，则oss.endpoint需要配置为oss-cn-hangzhou-internal.aliyuncs.com。 **说明** 如果您需要跨VPC访问OSS，则可以将oss.endpoint配置为OSS的公网Endpoint。
spark.sql.catalog.<catalog-name>.warehouse	表数据存放在OSS的路径。	无
spark.sql.catalog.<catalog-name>.access.key.id	阿里云账号的Access Key。	获取方法请参见获取AccessKey。
spark.sql.catalog.<catalog-name>.access.key.secret	阿里云账号的Access Secret。	获取方法请参见获取AccessKey。

参数	描述	备注
spark.sql.catalog.<catalog-name>.dlf.catalog-id	阿里云账号的账号ID。	登录账号信息，请通过用户信息页面获取。 
spark.sql.catalog.<catalog-name>.dlf.endpoint	DLF服务的Endpoint。	详情请参见已开通的地域和访问域名。 推荐您设置dlf.endpoint参数为DLF的VPC Endpoint。例如，如果您选择的地域为cn-hangzhou地域，则dlf.endpoint参数需要配置为dlf-vpc.cn-hangzhou.aliyuncs.com。 **说明** 您也可以使用DLF的公网Endpoint，如果您选择的地域为cn-hangzhou地域，则dlf.endpoint参数需要配置为dlf.cn-hangzhou.aliyuncs.com。
spark.sql.catalog.<catalog-name>.dlf.region-id	DLF服务的地域名。	详情请参见已开通的地域和访问域名。 **说明** 请和dlf.endpoint选择的地域保持一致。

Hive配置

请根据您创建的集群版本进行相应的配置：

- **EMR-3.39.0及后续版本和EMR-5.5.0及后续版本**

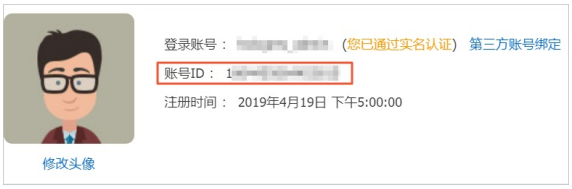
说明 默认的Catalog名称为dlf。

参数	描述	备注
iceberg.catalog.<catalog-name>.catalog-impl	Catalog的Class类名。	固定值为org.apache.iceberg.aliyun.dlf.hive.DlfCatalog。

- **EMR-3.38.x版本和EMR-5.3.x~EMR-5.4.x版本（包含）**

说明 默认的Catalog名称为dlf_catalog。

参数	描述	备注
iceberg.catalog	Catalog名称。	请填写为自定义的英文名。
iceberg.catalog.<catalog-name>.type	Catalog类型。	固定值为custom。
iceberg.catalog.<catalog-name>.catalog-impl	Catalog的Class类名。	固定值为org.apache.iceberg.aliyun.dlf.DlfCatalog。
iceberg.catalog.<catalog-name>.io-impl	IO写入的Class名称。	固定值为org.apache.iceberg.hadoop.HadoopFileIO。
iceberg.catalog.<catalog-name>.warehouse	表数据存放在warehouse路径。	可以是HDFS路径或者阿里云对象存储服务OSS路径。
iceberg.catalog.<catalog-name>.access.key.id	阿里云账号的Access Key。	获取方法请参见获取AccessKey。

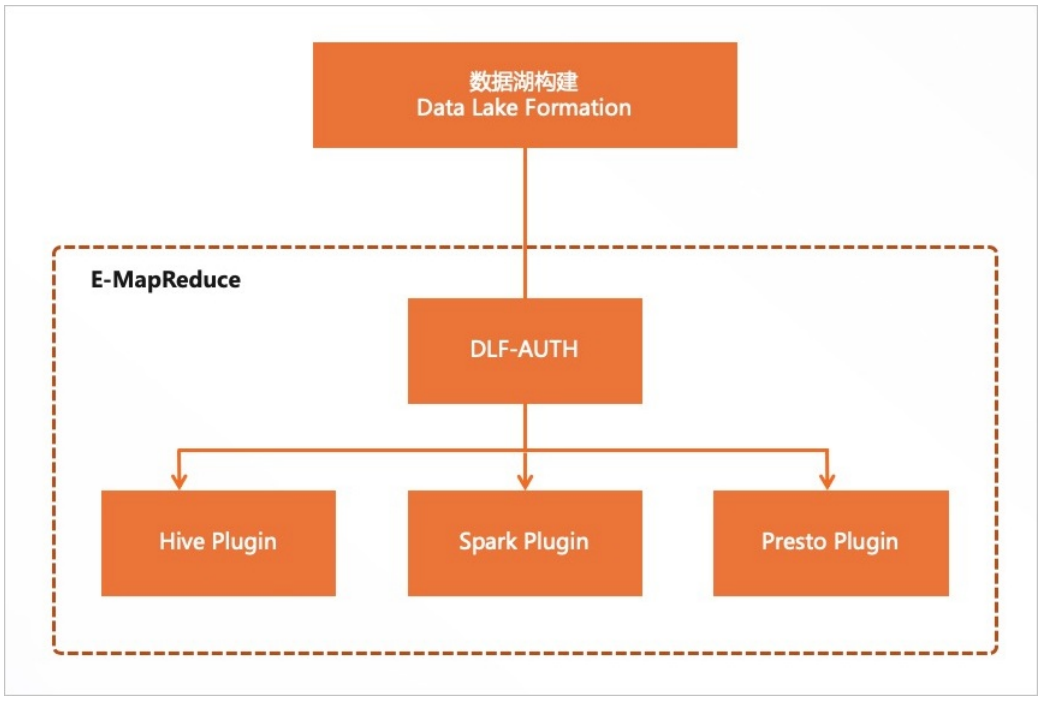
参数	描述	备注
iceberg.catalog.<catalog-name>.access.key.secret	阿里云账号的Access Secret。	获取方法请参见获取AccessKey。
iceberg.catalog.<catalog-name>.dlf.catalog-id	阿里云账号的账号ID。	登录账号信息，请通过用户信息页面获取。 
iceberg.catalog.<catalog-name>.dlf.endpoint	DLF服务的Endpoint。	详情请参见已开通的地域和访问域名。 推荐您设置dlf.endpoint参数为DLF的VPC Endpoint。例如，如果您选择的地域为cn-hangzhou地域，则dlf.endpoint参数需要配置为dlf-vpc.cn-hangzhou.aliyuncs.com。 说明 您也可以使用DLF的公网Endpoint，如果您选择的地域为cn-hangzhou地域，则dlf.endpoint参数需要配置为dlf.cn-hangzhou.aliyuncs.com。
iceberg.catalog.<catalog-name>.dlf.region-id	DLF服务的地域名。	详情请参见已开通的地域和访问域名。 说明 请和dlf.endpoint选择的地域保持一致。

6.2.2. DLF-Auth

DLF-Auth组件是阿里云数据湖构建（Data Lake Formation，DLF）产品提供的，通过该组件可以开启数据湖构建DLF的权限功能，实现数据湖上全托管的统一的权限管理。本文为您介绍如何开启DLF-Auth权限。

背景信息

数据湖构建DLF是一款全托管的快速帮助用户构建云上数据湖的服务，提供了云上数据湖统一的权限管理和元数据管理，详细信息请参见数据湖构建产品简介。



前提条件

- 已创建E-MapReduce集群，详情请参见创建集群。

  > **说明** 在基础配置页面，**元数据选择**使用默认的**DLF统一元数据**。

- 数据湖构建DLF的数据权限管理功能是白名单模式，如需此功能，请提交工单处理，产品名称选择为数据湖构建。

使用限制

- 数据湖构建DLF权限仅支持通过RAM用户进行权限管理，因此需要在EMR控制台通过用户管理功能添加用户。
- 数据湖构建DLF的数据权限管理功能支持区域请参见已开通的地域和访问域名，其他区域如需此功能，请提交工单处理，提交工单时产品名称选择数据湖构建。
- DLF-Auth启用Hive、Spark或Presto后，Ranger将无法点击启用或禁用Hive、Spark或Presto；Ranger启用Hive、Spark或Presto后，DLF-Auth将无法点击启用或禁用Hive、Spark或Presto。
- 支持DLF-Auth组件的EMR版本：
  - EMR 3.X系列：EMR-3.38.0及后续版本。
  - EMR 5.X系列：EMR-5.4.0及后续版本。

  > **说明** 其余版本，如果需要使用DLF-Auth组件功能，请提交工单处理。

操作流程

通过本文操作，您可以开启DLF-Auth，实现数据湖上全托管的统一的权限管理。

1. 步骤一：开启Hive权限控制
2. 步骤二：添加RAM用户
3. 步骤三：验证权限
4. （可选）步骤四：开启Hive LDAP认证

   如果开启了DLF-Auth权限，建议您开启Hive LDAP认证，以便于连接Hive的用户都可以通过LDAP认证后执行相关脚本。

步骤一：开启Hive权限控制

1. 进入DLF-Auth页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务 > DLF-Auth**。
2. 开启Hive权限控制。
   i. 在DLF-Auth服务页面，选择右上角的**操作 > 启用Hive**。
   ii. 在**执行集群操作**对话中，输入执行原因，单击**确定**。
   iii. 在**确认**对话中，单击**确定**。
3. （可选）单击上方的**查看操作历史**。

   直至操作状态显示**成功**。
4. 重启HiveServer2。
   i. 在左侧导航栏，选择**集群服务 > Hive**。
   ii. 在Hive服务页面，选择右上角的**操作 > 重启HiveServer2**。
   iii. 在**执行集群操作**对话中，输入**执行原因**，单击**确定**。
   iv. 在**确认**对话中，单击**确定**。
5. （可选）单击上方的**查看操作历史**。

   直至操作状态显示**成功**。

步骤二：添加RAM用户

您可以通过用户管理功能添加用户，详细操作如下。

1. 进入集群详情页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。

2. 在左侧导航栏，单击**用户管理**。
3. 在**用户管理**页面，单击**添加用户**。
4. 在**添加用户**对话框的**用户名**下拉列表中，选择已有的RAM用户作为EMR用户的名称，输入**密码**和**确认密码**。
5. 单击**确定**。

步骤三：验证权限

1. 授权前验证当前用户权限。
   i. 使用SSH方式登录到集群，详情请参见登录集群。
   ii. 执行以下命令访问HiveServer2。

   ```
   beeline -u jdbc:hive2://emr-header-1:10000 -n <user> -p <password>
   ```

   > **说明** <user>和<password>为步骤二：添加RAM用户中您设置的用户名和密码。

   iii. 查看已有数据表信息。

   例如，执行以下命令，查看test表信息。`testdb.test` 请根据您实际信息修改。

   ```
   select * from testdb.test;
   ```

   因为当前用户没有权限，会报没有权限而查询失败的错。

   ```
   0: jdbc:hive2://emr-header-1:10000> select * from testdb.test;
   22/07/07 19:03:32 [main]: WARN conf.HiveConf: HiveConf of name hive.metastore.delta.compatible.mode.enabled does not e
   xist
   Error: Error while compiling statement: FAILED: HiveAccessControlException Permission denied: user [dlf-test] does not
    have [SELECT] privilege on [database=testdb/table=test/*]. Permission denied: DLF checkPermission failed. message=[Ac
   tion: CHECK_PERMISSIONS ErrorCode: NoPermission Message: Authorization Failed [4019], You have NO privilege 'Select' o
   n {acs:dlf:cn-hangzhou:1250460021754461:metastore/catalogs/1250460021754461/databases/testdb/tables/test}. Deny as def
   ault for Resource.  Context ID:dd198392-9b30-4899-9b32-61e1a993bd55.    --->Tips: Principal:acs:ram::1250460021754461:
   user/dlf-test Not pass by dlf permission check. RequestId: C4EC27BE-31ED-5A4B-AD40-2C8EF60350C0] (state=42000,code=400
   00)
   ```

2. 为RAM用户添加权限。
   i. 登录数据湖构建控制台。
   ii. 在左侧导航栏中，选择**数据权限 > 数据授权**。
   iii. 在**数据授权**页面，单击**新增授权**。
   iv. 在**新增授权**页面，配置以下参数。

   参数详情信息，请参见新增授权。

参数		描述
授权主体	**主体类型**	默认RAM用户。
	主体选择	在**主体选择**下拉列表中，选择您在步骤二：添加RAM用户中添加的用户。
授权资源	**授权方式**	默认资源授权。
	资源类型	根据您实际情况选择。 本文示例为元数据表。
权限配置	**数据权限**	本文示例为**Select**。
	授权权限	

   v. 单击**确定**。
3. 授权后验证当前用户权限。

   参见步骤1重新查看数据表的信息，因为已经授权，所以可以查询到相关数据表的信息。

（可选）步骤四：开启Hive LDAP认证

1. 进入Hive页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。

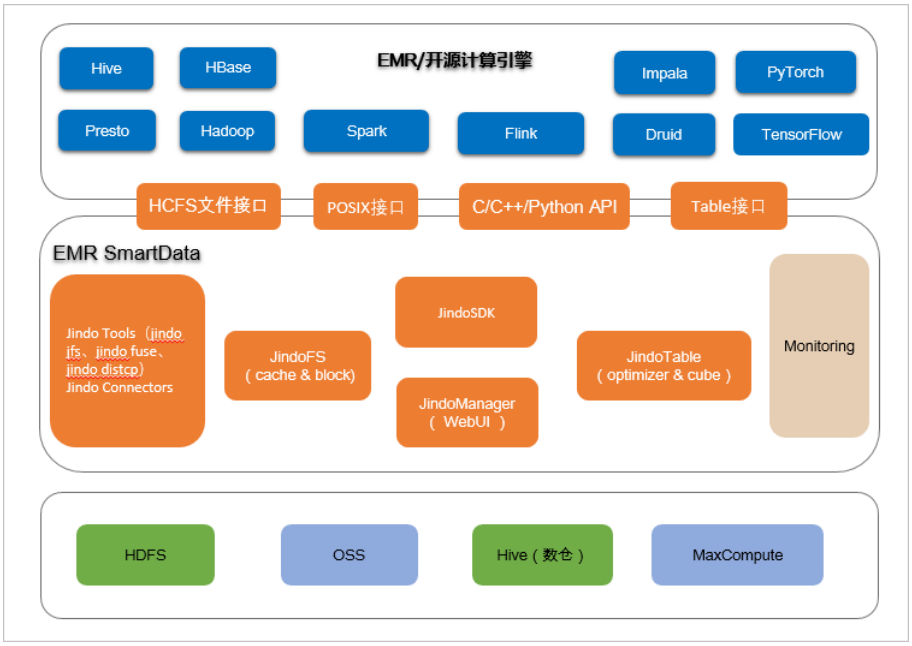
v. 在左侧导航栏，选择**集群服务 > Hive**。

2. 开启LDAP认证。
   i. 在Hive服务页面，选择右上角的**操作 > 开启LDAP认证**。
   ii. 在**执行集群操作**对话中，单击**确认**。
3. 单击上方的**查看操作历史**。
   直至操作状态显示**成功**。
4. 重启HiveServer2。
   i. 在Hive服务页面，选择右上角的**操作 > 重启HiveServer2**。
   ii. 在**执行集群操作**对话中，输入**执行原因**，单击**确定**。
   iii. 在**确认**对话中，单击**确定**。

6.2.3. SmartData

SmartData是E-MapReduce（简称EMR）产品的核心自研组件，为EMR各个计算引擎提供统一的存储优化、缓存优化、计算加速优化和多个存储功能扩展，涵盖数据访问、数据治理和数据安全。

SmartData组件在EMR产品中的位置如下所示。



SmartData组件包括：

- JindoFS核心子系统：为各种远端存储系统提供缓存和缓存加速，详情请参见JindoFS介绍和使用。
- JindoTable核心子系统：为表格数据源（例如Hive数仓）提供表和分区级别的优化和治理，详情请参见JindoTable使用说明。
- JindoManager：提供JindoFS&JindoTable相关服务和功能的管理页面，例如，查看文件和表在缓存上的各种统计指标。
- JindoSDK：为EMR各种开源计算引擎提供统一的SDK，支持Java、C、C++和Python语言，提供多种访问和API接口，包括HCFS文件系统接口、POSIX接口和Table表格接口。
- 工具集： 提供相关的工具集，例如Jindo tool和迁移工具Jindo DistCp。
- 各种Connectors：包括Hadoop connector、Flink connector和TensorFlow connector，支持Kite SDK、Apache Beams、Flume、Sqoop和Kafka。

SmartData目前通过JindoFS和JindoTable支持的数据源，包括阿里云OSS、Apache Hadoop HDFS、Hive数仓和阿里云MaxCompute。

SmartData作为EMR产品核心自研组件，独立开发与版本发布，详细版本请参见版本概述。

SmartData详细使用，请查看相应文档：

- SmartData 3.8.x版本简介
- SmartData 3.7.x版本简介
- SmartData 3.6.x版本简介
- SmartData 3.5.x版本简介
- SmartData 3.4.x版本简介
- SmartData 3.2.x版本简介

- SmartData 3.1.x版本简介
- SmartData 3.0.x版本简介
- SmartData 2.7.3-2.7.4版本
- SmartData 2.6.0-2.7.2版本简介
- SmartData使用说明（EMR-3.22.0~3.25.1版本）
- SmartData使用说明（EMR-3.20.0~3.22.0版本）

6.2.4. Oozie

Oozie是一个开源的大数据工作流调度引擎，用于调度大数据作业，完成复杂的数据生产业务。本文介绍如何在E-MapReduce上使用Oozie。

前提条件

已创建E-MapReduce的Hadoop集群，并且选择了Oozie服务。详情请参见创建集群。

使用限制

该文档仅适用于Oozie 5.2.x版本。

访问Oozie UI页面

支持以下两种方式访问：

- 如果您需要通过控制台的方式访问Web UI时，请参见访问链接与端口。
- 如果通过在本地服务器上建立SSH隧道以端口转发的方式来访问Web UI时，请参见通过SSH隧道方式访问开源组件Web UI。

提交Workflow作业

因为E-MapReduce集群中，默认安装了*sharelib*，所以您使用Oozie提交Workflow作业时，不需要再安装*sharelib*。

1. 在*job.properties*文件中指定不同的NameNode和JobTracker（ResourceManager）。
   - 非HA集群

     ```
     nameNode=hdfs://emr-header-1:9000
     jobTracker=emr-header-1:8032
     ```

   - HA集群

     ```
     nameNode=hdfs://emr-cluster
     jobTracker=rm1,rm2
     ```

2. 提交Workflow作业。
   - 在非HA集群上提交Workflow作业
     a. 登录集群的主Master节点，详情请参见登录集群。
     b. 执行以下命令，下载并解压缩示例代码。

     ```
     su oozie
     cd /tmp
     wget http://emr-sample-projects.oss-cn-hangzhou.aliyuncs.com/oozie-examples/oozie-examples.zip
     unzip oozie-examples.zip
     ```

     c. 同步Oozie Workflow代码至HDFS。

     ```
     hadoop fs -copyFromLocal examples/ /user/oozie/examples
     ```

     d. 提交Oozie Workflow样例作业。

     ```
     $OOZIE_HOME/bin/oozie job -config examples/apps/map-reduce/job.properties -run
     ```

     执行成功之后，返回如下信息。

     ```
     job: 0000000-160627195651086-oozie-oozi-W
     ```

     e. 访问Oozie UI页面，详情请参见访问Oozie UI页面。

     您可以看查看提交的Oozie Workflow Job。
   - 在HA集群上提交Workflow作业
     a. 登录集群的主Master节点，详情请参见登录集群。

     ```
     ssh root@主Master公网IP地址
     ```

b. 执行以下命令，下载并解压缩示例代码。

```
su oozie
cd /tmp
wget http://emr-sample-projects.oss-cn-hangzhou.aliyuncs.com/oozie-examples/oozie-examples-ha.zip
unzip oozie-examples-ha.zip
```

c. 同步Oozie Workflow代码至HDFS。

```
hadoop fs -copyFromLocal examples/ /user/oozie/examples
```

d. 提交Oozie Workflow样例作业。

```
$OOZIE_HOME/bin/oozie job -config examples/apps/map-reduce/job.properties -run
```

执行成功之后，返回如下信息。

```
job: 0000000-160627195651086-oozie-oozi-W
```

e. 访问Oozie UI页面，详情请参见访问Oozie UI页面。

您可以看查看提交的Oozie Workflow Job。

6.2.5. ESS

ESS（EMR Remote Shuffle Service）是E-MapReduce（简称EMR）在优化计算引擎的Shuffle操作上，推出的扩展组件。

背景信息

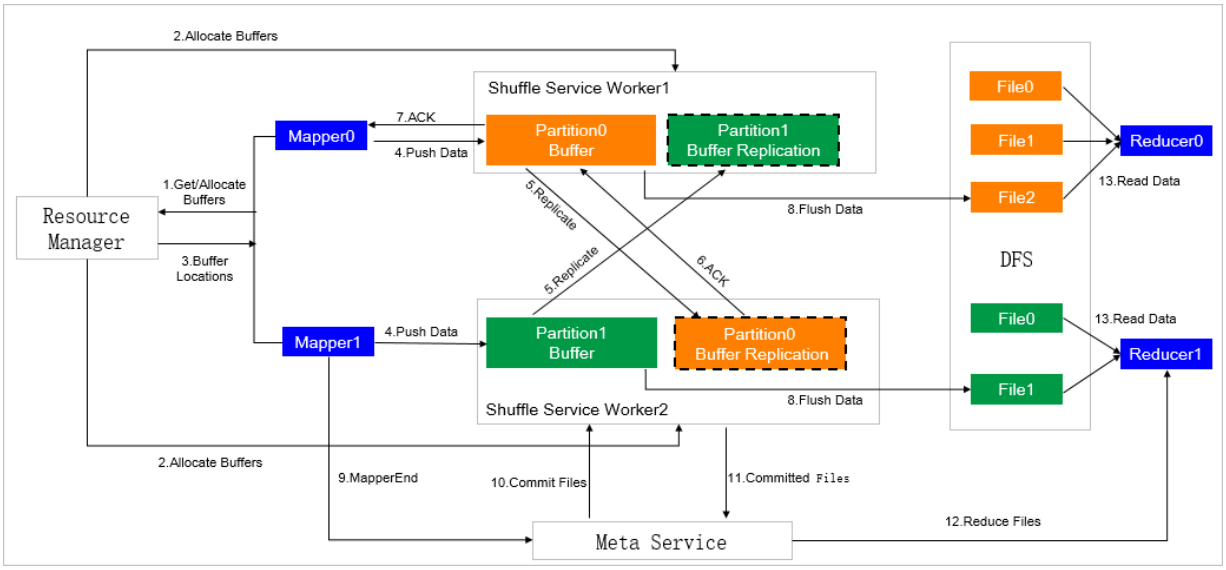
目前Shuffle方案缺点如下：

- Shuffle Write在大数据量场景下会溢出，导致写放大。
- Shuffle Read过程中有大量的网络小包导致Connection reset问题。
- Shuffle Read过程中存在大量小数据量的IO请求和随机读，对磁盘和CPU造成高负载。
- 对于M*N次的连接数，在M和N数千的规模下，作业基本无法完成。
- NodeManager和Spark Shuffle Service是同一进程，当Shuffle的数据量特别大时，通常会导致NodeManager重启，从而影响YARN调度的稳定性。

EMR推出的基于Shuffle的ESS服务，可以优化目前Shuffle方案的问题。ESS优势如下：

- 使用Push-Style Shuffle代替Pull-Style，减少Mapper的内存压力。
- 支持IO聚合，Shuffle Read的连接数从M*N降到N，同时更改随机读为顺序读。
- 支持两副本机制，降低Fetch Fail概率。
- 支持计算与存储分离架构，可以部署Shuffle Service至特殊硬件环境中，与计算集群分离。
- 解决Spark on Kubernetes时对本地磁盘的依赖。

ESS设计架构图如下。



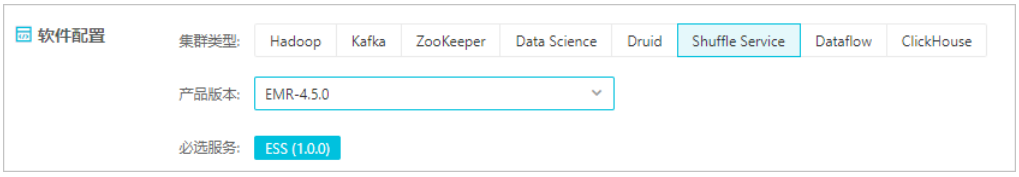
使用限制

此文档仅适用于EMR-3.39.1之前、EMR-4.x系列版本和EMR-5.5.0之前版本。EMR-3.39.1及之后版本，EMR-5.5.0及之后版本，详情请参见RSS。

创建集群

以EMR-4.5.0版本为例，您可以通过以下两种方式创建ESS的集群：

- 创建E-MapReduce的Shuffle Service集群。



- 创建E-MapReduce的Hadoop集群。

集群创建详情请参见创建集群。

使用ESS

Spark使用ESS时，需在提交Spark作业时添加以下配置项，配置详情请参见作业编辑。

Spark相关的参数，请参见Spark Configuration。

参数	描述
spark.shuffle.manager	固定值org.apache.spark.shuffle.ess.EssShuffleManager。
spark.ess.master.address	填写格式<ess-master-ip>:<ess-master-port>。 涉及参数如下： • `<ess-master-ip>`：Master节点的公网IP地址。 • `<ess-master-port>`：固定值9097。
spark.shuffle.service.enabled	设置为false。 使用EMR的Remote Shuffle Service时需要关闭原有的External Shuffle Service。
spark.shuffle.useOldFetchProtocol	设置为true。 兼容旧的Shuffle协议。
spark.sql.adaptive.enabled	设置为false。 EMR的Remote Shuffle Service暂不支持Adaptive Execution。
spark.sql.adaptive.skewJoin.enabled	

配置项说明

您可以在ESS服务配置页面，查看ESS所有的配置项。

参数	描述	默认值
ess.push.data.replicate	是否开启两副本。取值包含： • true：开启两副本。 • false：不开启两副本。 **说明** 建议生产环境开启两副本。	true
ess.worker.flush.queue.capacity	每个目录的Flush buffer数量。 **说明** 为了提升性能，您可以配置多块磁盘。为了提升整体的读写吞吐量，建议一块磁盘不多于2个目录。 每个目录的Flush buffer所消耗堆内的内存为ess.worker.flush.buffer.size * ess.worker.flush.queue.capacity，即 `256 KB * 512 = 128 MB` 。每个目录提供的槽位（slots）数量是该参数的一半。例如，总共28个目录，则整体内存消耗是 `128 MB * 28 = 3.5 GB` ，整体的slots数量是 `512 * 28 / 2 = 7168` 。	512
ess.flush.timeout	Flush到存储层的超时时间。	240s
ess.application.timeout	Application心跳超时时间，超时会清理Application相关资源。	240s
ess.worker.flush.buffer.size	Flush buffer大小，超过最大值会触发刷盘。	256k
ess.metrics.system.enable	是否打开监控。取值包含： • true：打开监控。 • false：不打开监控。	false
ess_worker_offheap_memory	Worker堆外内存大小。	4g
ess_worker_memory	Worker堆内内存大小。	4g
ess_master_memory	Master堆内内存大小。	4g

6.2.6. RSS

RSS（EMR Remote Shuffle Service）是E-MapReduce（简称EMR）为了提升Shuffle稳定性和性能推出的扩展组件。

背景信息

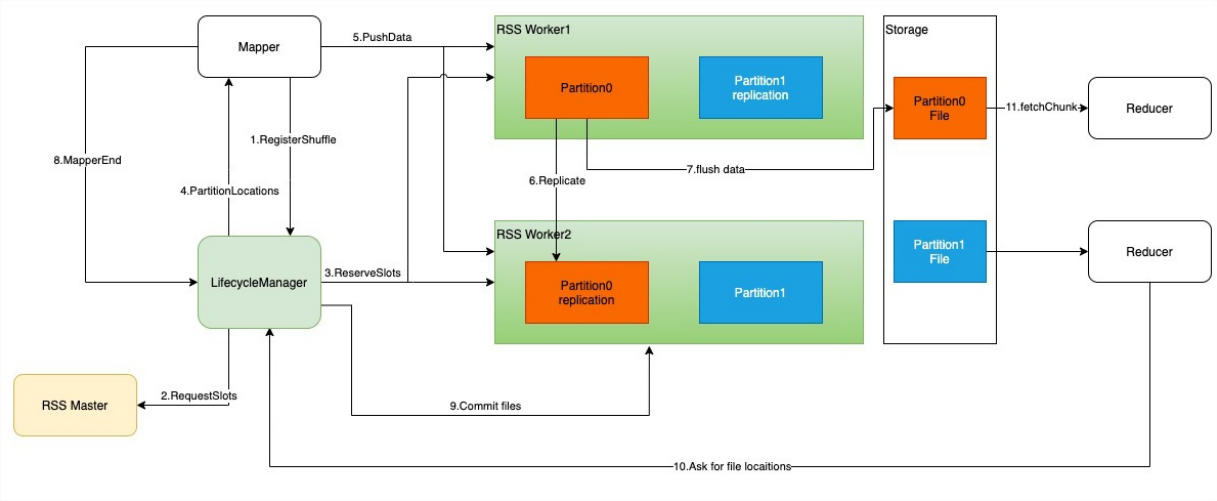
目前Shuffle方案缺点如下：

- Shuffle Write在大数据量场景下会溢出，导致写放大。
- Shuffle Read过程中存在大量的网络小包导致的Connection reset问题。
- Shuffle Read过程中存在大量小数据量的IO请求和随机读，对磁盘和CPU造成高负载。
- 对于M*N次的连接数，在M和N数千的规模下，作业基本无法完成。
- NodeManager和Spark Shuffle Service是同一进程，当Shuffle的数据量特别大时，通常会导致NodeManager重启，从而影响YARN调度的稳定性。

EMR推出的基于Shuffle的RSS服务，可以优化目前Shuffle方案的问题。RSS优势如下：

- 使用Push-Style Shuffle代替Pull-Style，减少Mapper的内存压力。
- 支持IO聚合，Shuffle Read的连接数从M*N降到N，同时更改随机读为顺序读。
- 支持两副本机制，降低Fetch Fail概率。
- 支持计算与存储分离架构，可以部署Shuffle Service至特殊硬件环境中，与计算集群分离。
- 解决Spark on Kubernetes时对本地磁盘的依赖。

RSS设计架构图如下。



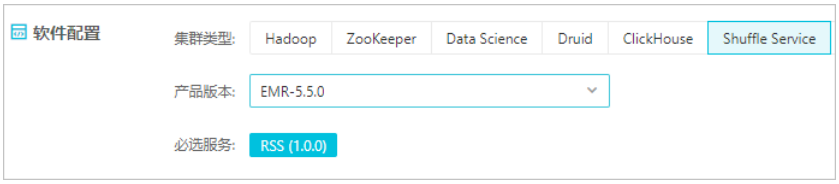
使用限制

此文档仅适用于EMR-3.39.1及后续版本，EMR-5.5.0及后续版本。

创建集群

以EMR-5.5.0版本为例，您可以通过以下两种方式创建RSS集群：

- 创建独立的Shuffle Service集群。



- 创建包含RSS组件的Hadoop集群。

集群创建详情请参见创建集群。

使用RSS

Spark使用RSS时，需在提交Spark作业时添加以下配置项，配置详情请参见作业编辑。

Spark相关的参数，请参见Spark Configuration。

参数	描述
spark.shuffle.manager	固定值org.apache.spark.shuffle.RSS.RSSShuffleManager。
spark.serializer	固定值org.apache.spark.serializer.KryoSerializer。
spark.rss.master.address	填写格式<rss-master-ip>:<rss-master-port>。 涉及参数如下： • `<rss-master-ip>`：Master节点的公网IP地址。 • `<rss-master-port>`：固定值9097。
spark.shuffle.service.enabled	默认值为false。 使用EMR的Remote Shuffle Service时需要关闭原有的External Shuffle Service。
spark.rss.shuffle.writer.mode	RSS的wirter支持的模式： • hash（默认值）：在Partition并发度过大的情况下会使用较多的内存。 • sort：使用固定大小内存，在Partition并发度很大的情况下，能够稳定工作。
spark.rss.push.data.replicate	是否开启两副本。取值如下： • true（默认值）：开启两副本。 • false：不开启两副本。
spark.sql.adaptive.enabled	默认值为false。 EMR的Remote Shuffle Service暂不支持Adaptive Execution。
spark.sql.adaptive.localShuffleReader.enabled	
spark.sql.adaptive.skewJoin.enabled	

配置项说明

您可以在RSS服务配置页面，查看RSS所有的配置项。

参数	描述	默认值
rss.worker.flush.queue.capacity	每个目录的Flush buffer数量。 **说明** 为了提升性能，您可以配置多块磁盘。为了提升整体的读写吞吐量，建议一块磁盘不多于2个目录。 每个目录的Flush buffer所消耗堆内的内存为RSS.worker.flush.buffer.size * RSS.worker.flush.queue.capacity，即 `256 KB * 512 = 128 MB` 。每个目录提供的槽位（slots）数量是该参数的一半。例如，总共28个目录，则整体内存消耗是 `128 MB * 28 = 3.5 GB` ，整体的slots数量是 `512 * 28 / 2 = 7168` 。	512
rss.flush.timeout	Flush到存储层的超时时间。	240s
rss.application.timeout	Application心跳超时时间，超时会清理Application相关资源。	240s
rss.worker.flush.buffer.size	Flush buffer大小，超过最大值会触发刷盘。	256k
rss.metrics.system.enable	是否打开监控。取值如下： • true：打开监控。 • false：不打开监控。	false
rss_worker_offheap_memory	Worker堆外内存大小。	4g
rss_worker_memory	Worker堆内内存大小。	4g
rss_master_memory	Master堆内内存大小。	4g

6.2.7. OpenLDAP

EMR-3.22.0及之后版本，默认启动了OpenLDAP服务。E-MapReduce支持Knox与OpenLDAP集成，您可以通过OpenLDAP管理用户。

前提条件

已创建E-MapReduce集群，详情请参见创建集群。

查看节点信息

1.
2. 在顶部菜单栏处，根据实际情况选择地域和资源组。
3. 单击上方的**集群管理**页签。
4. 在**集群管理**页面，单击集群右侧的**详情**。
5. 在左侧导航栏，单击**集群服务 > OpenLDAP**。
6. 单击**部署拓扑**。

   您可以查看OpenLDAP的节点信息，OpenLDAP部署在Master节点。如果是高可用集群，OpenLDAP部署在两个Master节点。

管理LDAP用户

- 方式一：在E-MapReduce控制台，管理集群中的LDAP用户。

  在**用户管理**页面，通过添加或删除用户来管理集群中的LDAP用户，详细信息请参见管理用户。
- 方式二：通过 `ldap` 命令，管理集群中的LDAP用户。

  例如，添加uid为arch，密码为12345678的LDAP用户。

  i. 通过SSH方式连接集群，详情请参见登录集群。

  ii. 创建*arch.ldif*文件，文件内容如下。

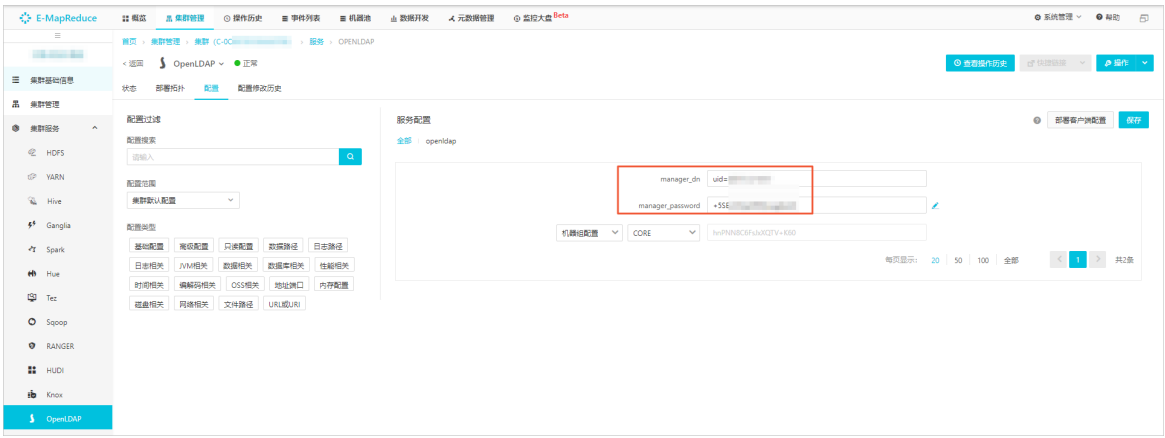
  ```
  dn: uid=arch,ou=people,o=emr
  cn: arch
  sn: arch
  objectClass: inetOrgPerson
  userPassword: 12345678
  uid: arch
  ```

  iii. 执行如下命令，添加LDAP用户。

  ```
  ldapadd -H ldap://emr-header-1:10389 -f arch.ldif -D uid=${uid} -w ${rootDnPW}
  ```

  > **说明**
  > - `${uid}`：为下图中获取到manager_dn的值。
  > - `${rootDnPW}`：为下图中获取到manager_password的值。
  > - `10389`：为OpenLDAP服务的监听端口。

  您可以在E-MapReduce控制台，OpenLDAP服务的**配置**页面，获取**manager_dn**和**manager_password**。

  iv. （可选）查看到该LDAP用户信息。

  ```
  ldapsearch -w ${rootDnPW}  -D "${uid}" -H ldap://emr-header-1:10389 -b uid=arch,ou=people,o=emr
  ```

  您也可以通过以下命令，删除添加的LDAP用户。

  ```
  ldapdelete -x -D "${uid}" -w ${rootDnPW} -r uid=arch,ou=people,o=emr -H ldap://emr-header-1:10389
  ```

6.2.8. Sqoop

Sqoop是一款Apache社区的开源软件，支持在Hadoop生态软件和结构化数据集（例如数据库）之间进行高效的批量数据传输。

前提条件

已创建Hadoop集群，详情请参见创建集群。

背景信息

常见数据传输场景如下：

- 将MySQL数据导入HDFS
- 将HDFS数据导入MySQL
- 将Hive数据导入MySQL
- 将MySQL数据导入Hive
- 将MySQL数据导入OSS
- 将OSS数据导入MySQL
- 使用SQL作为导入条件

> **注意** 在数据迁移前，请切换您的用户为hadoop。

将MySQL数据导入HDFS

在Master节点上执行如下命令。

```
sqoop import --connect <dburi>/<dbname> --username <username> --password <password> --table <tablename> --target-dir <hdfs-dir> --split-by <split-column> --check-column <col> --incremental <mode> --last-value <value>
```

参数	描述
dburi	数据库的访问链接。例如*jdbc:mysql://192.168.xxx.xxx:3306/*。
dbname	数据库的名称。
username	数据库登录用户名。
password	数据库登录密码。
tablename	MySQL表的名称。
hdfs-dir	HDFS的写入目录。例如*/user/hive/result*。
split-column	可选参数。指定一个用于切分任务的列，默认为表主键列。
col	可选参数。增量导入场景的检查列。
mode	可选参数。增量导入模式，支持append和lastmodified两种模式。
value	可选参数。指定上次增量导入的检查列的最大值。

详细的参数信息请参见Sqoop Import。

将HDFS数据导入MySQL

创建好对应HDFS中的数据结构的MySQL表后，在集群的Master节点上执行如下命令。

```
sqoop export --connect <dburi>/<dbname> --username <username> --password <password> --table <tablename> --export-dir <hdfs-dir>
```

参数	描述
dburi	数据库的访问链接。例如*jdbc:mysql://192.168.xxx.xxx:3306/*。
dbname	数据库的名称。
username	数据库登录用户名。

参数	描述
password	数据库登录密码。
tablename	MySQL表的名称。
hdfs-dir	HDFS的写入目录。例如*/user/hive/result*。

详细的参数信息请参见Sqoop Import。

将MySQL数据导入Hive

在集群的Master节点上执行如下命令。

```
sqoop import --connect <dburi>/<dbname> --username <username> --password <password> --table <tablename> --check-column <col
> --incremental <mode> --last-value <value> --fields-terminated-by "\t" --lines-terminated-by "\n" --hive-import --target-d
ir <hdfs-dir> --hive-table <hive-tablename>
```

参数	描述
dburi	数据库的访问链接。例如*jdbc:mysql://192.168.xxx.xxx:3306/*。
dbname	数据库的名称。
username	数据库登录用户名。
password	数据库登录密码。
tablename	MySQL表的名称。
col	可选参数。增量导入场景的检查列。
mode	可选参数。增量导入模式，支持append和lastmodified两种模式。
value	可选参数。指定上次增量导入的检查列的最大值。
hdfs-dir	HDFS的写入目录。例如*/user/hive/result*。
hive-tablename	Hive中的表名。

详细的参数信息请参见Sqoop Import。

将Hive数据导入MySQL

执行命令与导入HDFS数据至MySQL一致，但需要指定Hive表对应的HDFS路径。详情请参见将HDFS数据导入MySQL。

将MySQL数据导入OSS

在集群的Master节点上执行如下命令。

```
sqoop import --connect <dburi>/<dbname> --username <username> --password <password> --table <tablename> --target-dir <oss-d
ir> --temporary-rootdir <oss-tmpdir> --check-column <col> --incremental <mode> --last-value <value>
```

参数	描述
dburi	数据库的访问链接。例如*jdbc:mysql://192.168.xxx.xxx:3306/*。
dbname	数据库的名称。
username	数据库登录用户名。
password	数据库登录密码。
tablename	MySQL表的名称。
oss-dir	OSS的写入目录。例如*oss://<accessid>:<accesskey>@<bucketname>.oss-cn-hangzhou-internal.aliyuncs.com/result*。

参数	描述
oss-tmpdir	临时写入目录。指定mode为append模式时，需要指定该参数。 采用append模式后，Sqoop会先将数据导入临时目录，然后将文件重命名为正常目标目录。如果目标目录已经存在于HDFS中，则Sqoop拒绝导入并覆盖该目录的内容。
col	可选参数。增量导入场景的检查列。
mode	可选参数。增量导入模式，支持append和lastmodified两种模式。
value	可选参数。指定上次增量导入的检查列的最大值。

详细的参数信息请参见Sqoop Import。

将OSS数据导入MySQL

创建好对应OSS中数据结构的MySQL表后，在集群的Master节点上执行如下命令。

```
sqoop export --connect <dburi>/<dbname> --username <username> --password <password> --table <tablename> --export-dir <oss-dir>
```

参数	描述
dburi	数据库的访问链接。例如*jdbc:mysql://192.168.xxx.xxx:3306/*。
dbname	数据库的名称。
username	数据库登录用户名。
password	数据库登录密码。
tablename	MySQL表的名称。
oss-dir	OSS的写入目录。例如*oss://<accessid>:<accesskey>@<bucketname>.oss-cn-hangzhou-internal.aliyuncs.com/result*。

详细的参数信息请参见Sqoop Import。

使用SQL作为导入条件

命令和参数如下所示。

```
sqoop import --connect <dburi>/<dbname> --username <username> --password <password> --query <query-sql> --split-by <sp-column> --hive-import --hive-table <hive-tablename> --target-dir <hdfs-dir>
```

参数	描述
dburi	数据库的访问链接。例如*jdbc:mysql://192.168.xxx.xxx:3306/*。
dbname	数据库的名称。
username	数据库登录用户名。
password	数据库登录密码。
query-sql	使用的查询语句。例如`SELECT * FROM profile WHERE id>1 AND \$CONDITIONS`。
sp-column	进行切分的条件。通常跟MySQL表的主键有关。
hdfs-dir	HDFS的写入目录。例如*/user/hive/result*。
hive-tablename	Hive中的表名。

详细的参数信息请参见Sqoop Import。

集群和其他数据库的网络配置请参见在 E-MapReduce上使用 Sqoop工具与数据库同步数据进行网络配置。

6.2.9. Knox

本文介绍如何在E-MapReduce上配置Knox，以通过公网方式访问HDFS、YARN、Spark和Ganglia等Web UI页面。

前提条件

已创建E-MapReduce集群，详情请参见创建集群。

准备工作

- 设置安全组访问：
  i. 获取您当前设备的公网访问IP地址。

  为了安全的访问集群组件，在设置安全组策略时，推荐您只针对当前的公网访问IP地址开放。获取您当前公网访问IP地址的方法是，访问IP地址，即可查看您当前的公网访问IP地址。

  ii. 添加8443端口：

  a. 在阿里云E-MapReduce控制台，集群**详情**页面的**网络信息**区域，单击**安全组ID**链接。

  b. 在**安全组规则**页面，单击**添加安全组规则**。

  c. 在**添加安全组规则**对话框中，**端口范围**填写**8443/8443**。

  d. **授权对象**填写步骤i中获取的公网访问IP地址。

  e. 单击**确定**。

  > 注意
  > - 为防止被外部的用户攻击导致安全问题，**授权对象**禁止填写为**0.0.0.0/0**。
  > - 如果您创建集群时，没有挂载公网IP，可以在ECS控制台为该ECS实例添加公网IP。添加成功后，返回EMR控制台，在集群管理下的**主机列表**页面，单击**同步主机信息**可以立即同步主机信息。
  > - Knox节点新挂载公网IP后，需要提交工单处理，进行域名和公网IP的绑定操作。

- 设置Knox用户

  访问Knox时需要验证身份，即需要输入您的用户名和密码。Knox的用户身份验证基于LDAP，您可以使用自有LDAP服务，也可以使用集群中Apache Directory Server的LDAP服务。

  - 使用集群中的LDAP服务

    方式一（推荐）

    在集群的**用户管理**页面，直接添加Knox访问账号，详情请参见用户管理。

    方式二：

    a. 通过SSH方式连接集群，详情请参见登录集群。

    b. 准备您的用户数据，例如Tom。

    执行以下命令，编辑*users.ldif*文件。

    ```
    su knox
    cd /usr/lib/knox-current/templates
    vi users.ldif
    ```

    本示例替换文件中的emr-guest和EMR GUEST为`Tom`，设置userPassword的值为您自定义的密码。

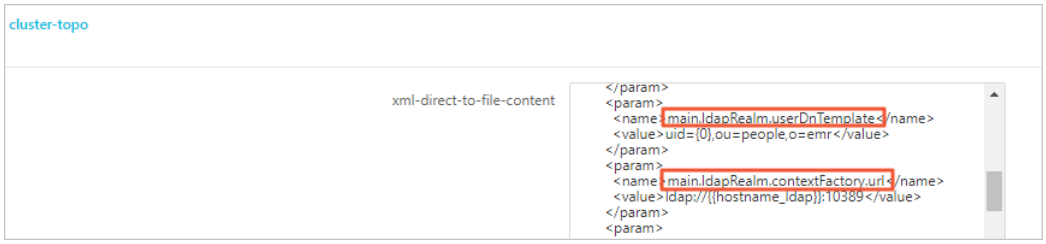
    ```
    # Entry for a sample end user
    # Please replace with site specific values
    dn: uid=emr-guest,ou=people,o=emr
    objectclass:top
    objectclass:person
    objectclass:organizationalPerson
    objectclass:inetOrgPerson
    cn: EMR GUEST
    sn: User
    uid: emr-guest
    userPassword:emr-guest-password
    ```

    c. 执行以下命令，导入用户数据至LDAP。

    ```
    sh ldap-sample-users.sh
    ```

- 使用自有LDAP服务的情况：
  a. 在E-MapReduce控制台的Knox服务的**配置**页面，单击**cluster-topo**页签。
  b. 配置**xml-direct-to-file-content**中参数。

配置项	描述
`main.ldapRealm.userDnTemplate`	设置为您的用户DN模板。
`main.ldapRealm.contextFactory.url`	设置为您的LDAP服务器域名和端口。



  c. 单击右上角的**保存**。
  d. 在**确认修改**对话框中，配置各项参数，单击**确定**。
  e. 在右上角选择**操作 > 重启 Knox**。
  f. 在**执行集群操作**对话框中，配置各项参数，单击**确定**。
  在**确认**对话框中，单击**确定**。
  g. 开启Knox访问公网LDAP服务的端口，例如10389。
  您可以参见8443端口的开启步骤，选择**出方向**，开启10389端口。

使用Knox帐户访问其他组件的Web UI

您可以使用Knox账号访问HDFS、YARN、Spark和Ganglia等Web UI页面。

- 使用E-MapReduce链接访问：
  i. 登录阿里云E-MapReduce控制台。
  ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
  iii. 单击上方的**集群管理**页签。
  iv. 在**集群管理**页面，单击集群右侧的**详情**。
  v. 在左侧导航栏，单击**访问链接与端口**。
  vi. 在**访问链接与端口**页面，单击服务所在行的链接。
- 使用集群公网IP地址访问：
  i. 在**集群基础信息**页面，查看公网IP地址。
  ii. 在浏览器中访问相应服务的URL。
    - HDFS UI：*https://{集群公网IP地址}:8443/gateway/cluster-topo/hdfs/*
    - Yarn UI：*https://{集群公网IP地址}:8443/gateway/cluster-topo/yarn/*
    - SparkHistory UI：*https://{集群公网IP地址}:8443/gateway/cluster-topo/sparkhistory/*
    - Ganglia UI：*https://{集群公网IP地址}:8443/gateway/cluster-topo/ganglia/*
    - Storm UI：*https://{集群公网IP地址}:8443/gateway/cluster-topo/storm/*
    - Oozie UI：*https://{集群公网IP地址}:8443/gateway/cluster-topo/oozie/*
    - Tez UI：*https://{集群公网IP地址}:8443/gateway/cluster-topo/tez-ui2/*
    - ImpalaCatalogd UI：*https://{集群公网IP地址}:8443/gateway/cluster-topo/impalalog/*
    - ImpalaStatestored UI：*https://{集群公网IP地址}:8443/gateway/cluster-topo/impalastore/*

用户权限管理（ACLs）

Knox提供服务级别的权限管理，可以限制特定的用户、用户组和IP地址访问特定的服务，详情请参见Apache Knox授权。

示例如下：

- 场景：YARN UI只允许用户Tom访问。
- 操作步骤：
  i. 在E-MapReduce控制台的Knox的**配置**页面，单击**cluster-topo**页签。

ii. 配置**xml-direct-to-file-content**中参数。

在 `<gateway>...</gateway>` 标签之间添加ACLs代码。

```
<provider>
      <role>authorization</role>
      <name>AclsAuthz</name>
      <enabled>true</enabled>
      <param>
          <name>YARNUI.acl</name>
          <value>Tom;*;*</value>
      </param>
</provider>
```

> **说明** 其中 `value` 填写格式为 `username;group;ipaddr`，分别表示用户，用户组和IP地址。如果无需特别指定时，您可以使用星号（*）代替，以匹配任何值。

iii. 单击右上角的**保存**。

iv. 在**确认修改**对话框中，配置各项参数，单击**确定**。

v. 在右上角选择**操作 > 重启 Knox**。

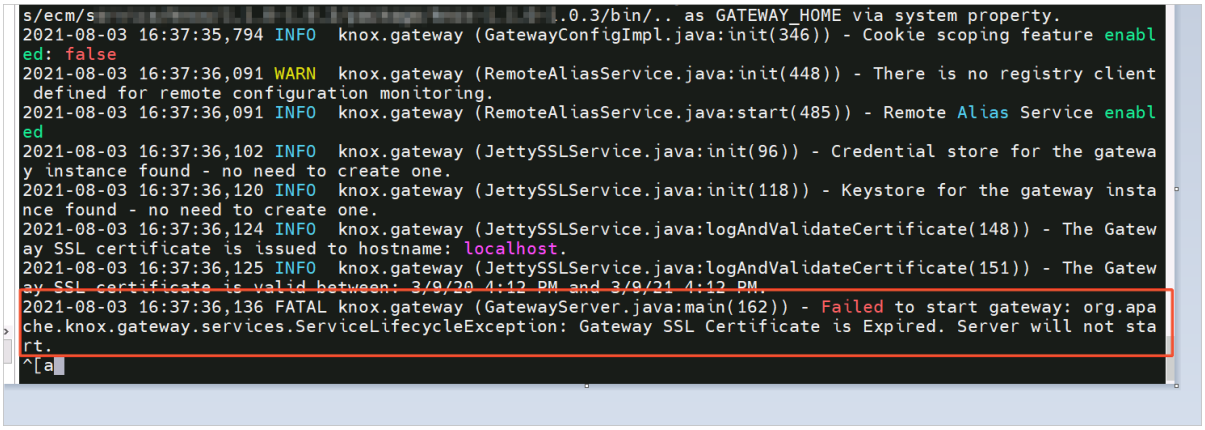
vi. 在**执行集群操作**对话框中，配置各项参数，单击**确定**。

在**确认**对话框中，单击**确定**。

> **警告** Knox会开放对应服务的REST API，您可以通过各服务的REST API操作服务。例如：HDFS文件的添加、删除等操作。出于安全原因，请勿使用Knox安装目录下的LDAP用户名和密码作为Knox的访问用户。

常见问题

- Q：Knox组件异常停止，启动Knox的时候报错Failed to start gateway: org.apache.hadoop.gateway.services.ServiceLifecycleException: Gateway SSL Certificate is Expired，具体信息如下图所示。



- A：您可以按照以下步骤处理。

i. 使用SSH登录集群，详情请参见登录集群。

ii. 执行以下命令，将之前错误的证书重命名。

```
sudo mv /usr/lib/knox-current/data/security/keystores/gateway.jks /usr/lib/knox-current/data/security/keystores/bak_gateway.jks
```

> **说明** 您也可以将之前错误的证书移动到其他目录。

iii. 启动Knox。

a. 在Knox服务页面，单击Knox操作列的**启动**。

b. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。

c. 在**确认**对话框中，单击**确定**。

您可以单击上方的**查看操作历史**，直至操作状态显示**成功**。

6.2.10. Superset

Superset是一款轻量级BI工具。您可以使用Superset连接多个数据源自助分析并可视化、定义图表和看板、导入或导出看板，并且可以对用户和角色进行权限管理。本文以EMR-3.34.0版本的集群为例为您介绍如何使用Superset。

背景信息

Superset对E-MapReduce Druid做了深度集成，同时也支持多种关系型数据库。因为E-MapReduce Druid也支持SQL，所以您可以通过Superset以两种方式访问E-MapReduce Druid，即Apache Druid原生查询语言和SQL方式。

前提条件

已创建E-MapReduce（简称EMR）的Hadoop或Druid集群，并选择了Superset服务，详情请参见创建集群。

使用限制

- Superset默认安装在emr-header-1节点，暂不支持HA。
- Superset不支持通过KNOX访问Web UI。
- 在使用Superset前，确保您的主机能够正常访问emr-header-1，详情请参见通过SSH隧道方式访问开源组件Web UI。

使用Superset访问Druid

1. 登录Superset。

   您需要在SSH连接中创建隧道以查看开源组件的Web页面，详情请参见通过SSH隧道方式访问开源组件Web UI。

   默认用户名和密码均为admin，请您登录后及时修改密码。

   > **说明** 首次登录后默认是英文界面。

2. 在Superset页面，单击右上角的 图标，选择**Chinese**。

   显示中文界面。

3. 添加Druid集群。

   i. 选择**数据源 > Druid集群**。

   ii. 单击 图标。

iii. 在**添加Druid集群**页面，配置如下参数。

参数	描述
代理主机	固定填写为**emr-header-1**。
代理端口	需要在开源端口前加1。 例如，开源Broker端口为8082，E-MapReduce中为18082。
Cluster Name	您在E-MapReduce中创建的集群名称。

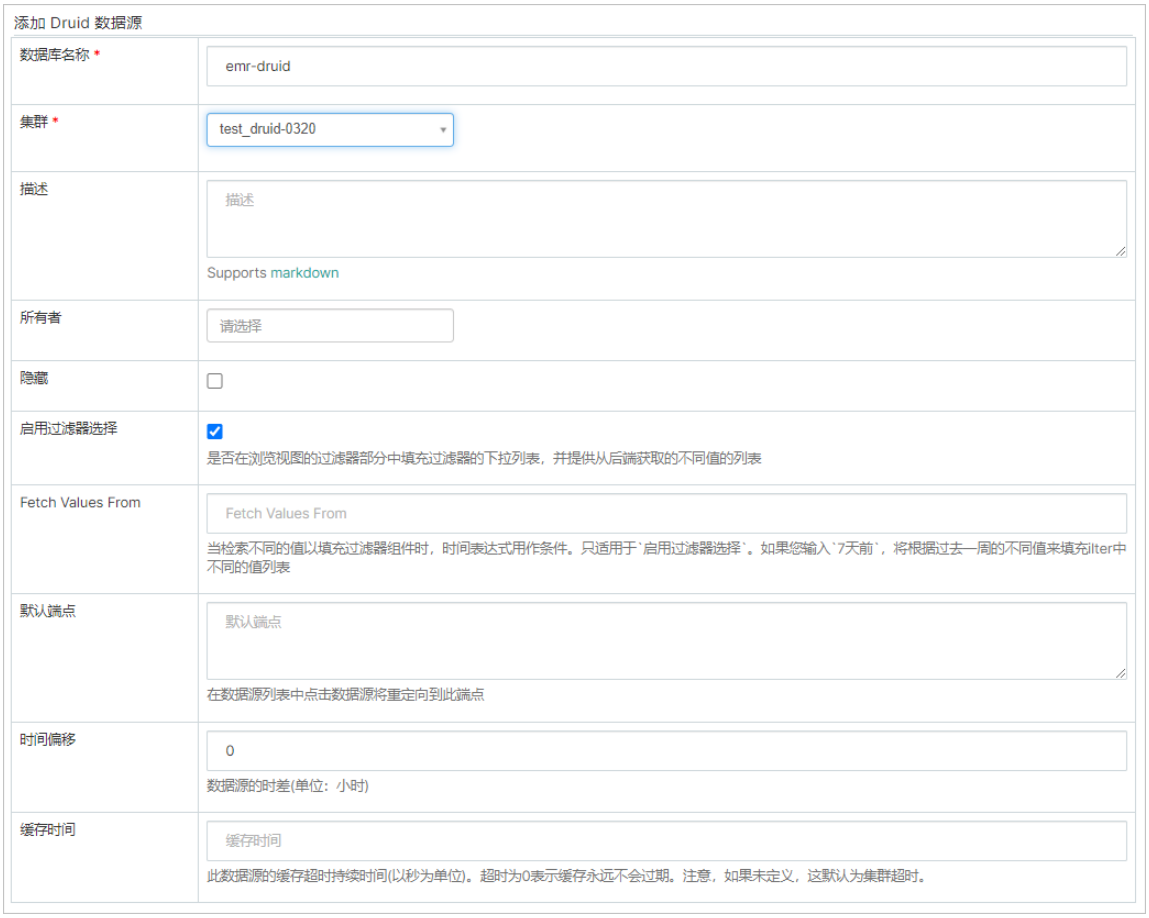
iv. 单击**保存**。

4. 添加数据源。

i. 选择**数据源 > Druid数据源**。

ii. 单击 图标。

iii. 在**添加Druid数据源**页面，配置如下参数。



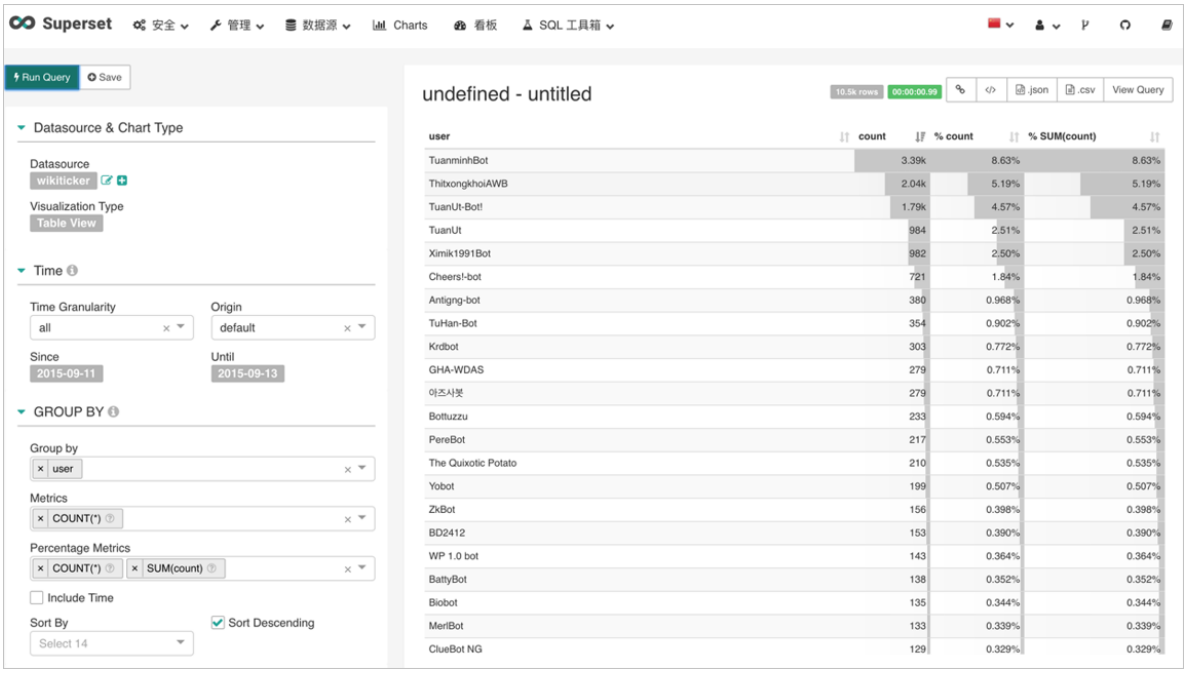
参数	描述
数据库名称	您可以自定义数据库的名称。
集群	您添加的Druid集群的名称。

iv. 单击**保存**。

保存之后，您可以单击  图标，填写相应的维度列与指标列等信息。

5. 查看添加的E-MapReduce Druid。

数据源添加成功后，您可以单击数据源名称，进入查询页面进行查询。



使用Superset访问Hive数据库

Superset提供了SQLAlchemy以多种语言支持各种各样的数据库，包括MySQL、Oracle、PostgreSQL和Microsoft SQL Server等关系型数据库，以及Hive、Presto和Druid等大数据查询引擎。这里以E-MapReduce Hadoop集群默认安装的Hive引擎为例，更多的数据库类型访问方式请参见SQLAlchemy。

1. 登录Superset。

   您需要在SSH连接中创建隧道以查看开源组件的Web页面，详情请参见通过SSH隧道方式访问开源组件Web UI。

   默认用户名和密码均为admin，请您登录后及时修改密码。

2. 在Superset页面，单击 图标，选择**Chinese**。

   登录后默认是英文界面。

3. 添加Hive数据库。

   i. 选择**数据源 > 数据库**。

   ii. 单击 图标。

   iii. 在**添加数据库**页面，填写数据库名称和SQLAlchemy URI。

参数	描述
数据库	您添加的数据库的名称。
SQLAlchemy URI	填写为**hive://emr-header-1:10000/**。

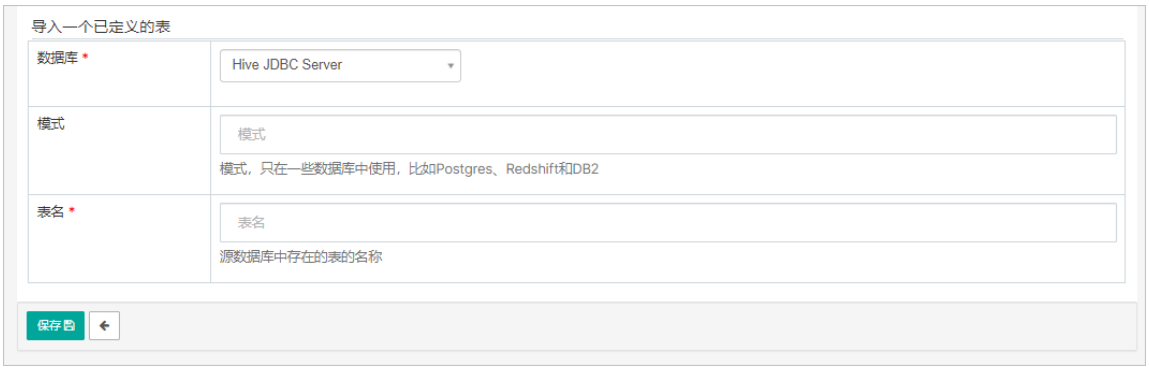
   iv. 单击**保存**。

4. 添加数据表。

   i. 选择**数据源 > 数据表**。

ii. 单击 图标。

iii. 在**导入一个已定义的表**页面，配置如下参数。



参数	描述
数据库	您添加的数据库的名称。
表名	您添加的数据库中存在的表的名称。 本文示例添加的是test表。

iv. 单击**保存**。

5. 查询数据库。

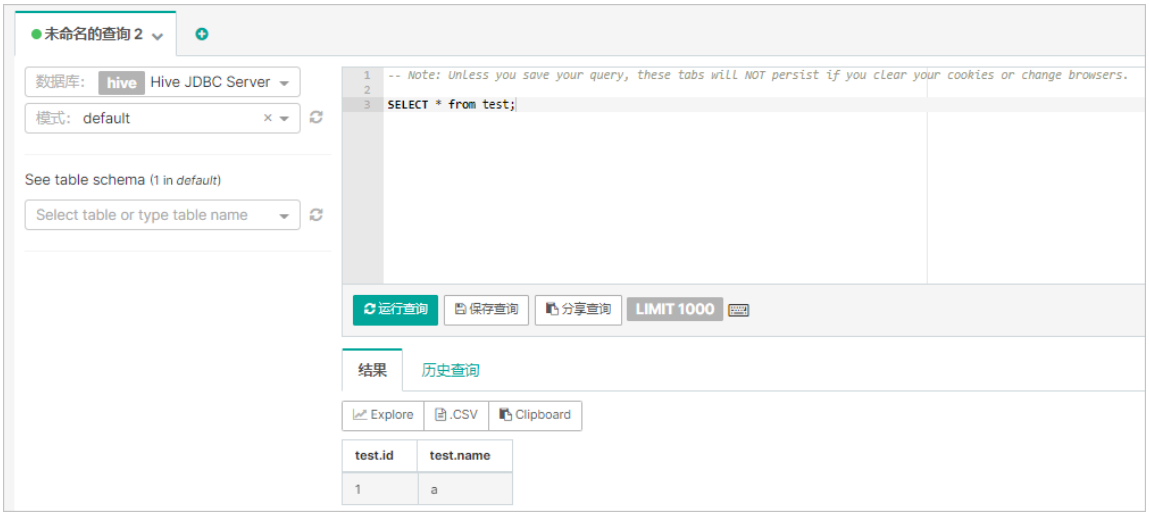
i. 选择**SQL工具箱 > SQL编辑器**。

ii. 选择添加的数据库Hive JDBC Server。

iii. 选择**default**模式。

iv. 您可以执行Hive命令查看数据库信息。

如下图所示。



常见问题

- 问题现象：EMR-4.6和EMR-3.33之前版本的集群，使用admin用户第一次登录Superset的Web UI时，报错invalid login。
- 解决方法：

i. 使用SSH方式登录到集群主节点，详情请参见登录集群。

> 注意 请使用root用户进行以下操作。

ii. 执行以下命令，进入Superset命令行。

```
source /usr/lib/superset-current/bin/activate
```

iii. 执行以下命令，创建管理员用户。

```
superset fab create-admin
```

根据如下提示信息输入用户名、密码和确认密码等。

```
Username [admin]:
User first name [admin]:
User last name [user]:
Email [admin@fab.org]:
Password:
Repeat for confirmation:
Recognized Database Authentications.
Admin User admin created.
```

iv. 初始化用户。

a. 执行以下命令，初始化数据库。

```
superset db upgrade
```

b. 执行以下命令，初始化Superset。

```
superset init
```

创建完成后，您需要创建隧道以查看开源组件的Web页面，然后使用新创建的用户登录Superset。创建隧道详情，请参见通过SSH隧道方式访问开源组件Web UI。

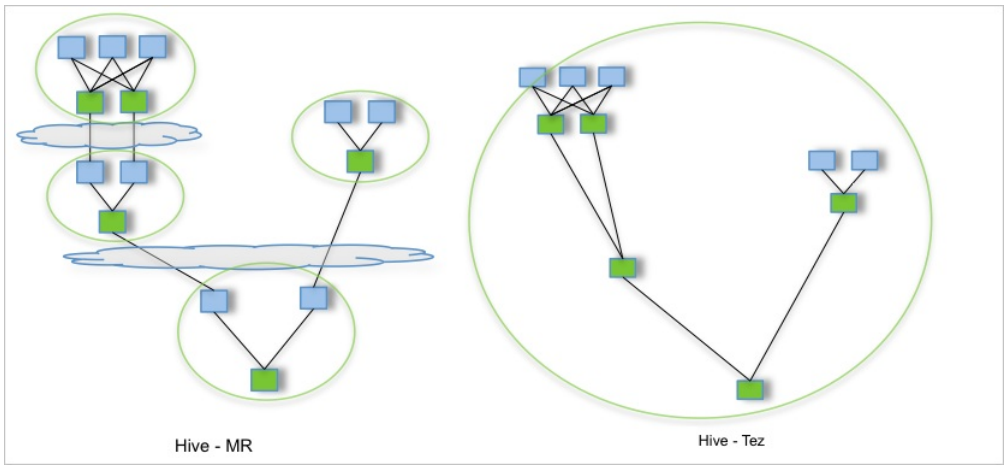
6.2.11. Tez

Tez是Apache构建在Hadoop之上的支持分布式DAG（Directed Acyclic Graph）的计算框架，支持通过复杂的DAG描述并处理大数据任务。

背景信息

Tez主要使用在Apache Hive中，作为Hive的一种运行时引擎，可以优化Hive SQL的查询引擎。与Hive On MR（MapReduce）相比，Hive On Tez具有更好的查询性能和稳定性。

Hive基于MapReduce提交任务和基于Tez提交任务流程图如下所示：



Tez的详细信息，请参见Apache TEZ。

开启Tez引擎

Hive支持使用Tez引擎执行SQL任务，在执行任务前您可以按照如下操作手动开启Tez引擎。

1. 进入详情页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
2. 在左侧导航栏中，选择**集群服务 > Hive**。
3. 在Hive服务页面，单击上方的**配置**页签。
4. 修改配置。
   i. 在配置搜索中，输入配置项**hive.execution.engine**，单击 图标。

ii. 设置**hive.execution.engine**的值为**tez**。

5. 保存配置。
   i. 在Hive服务页面，单击右上角的**保存**。
   ii. 在**确认修改**对话框中，输入执行原因，单击**确定**。
6. 重启HiveServer2。
   i. 在Hive服务页面，选择右上角的**操作 > 重启HiveServer2**。
   ii. 在**执行集群操作**对话中，输入**执行原因**，单击**确定**。
   iii. 在**确认**对话中，单击**确定**。

访问Tez Web UI

您可以在**访问链接与端口**页面，访问Tez Web UI。

访问Tez Web UI的详情，请参见访问链接与端口。

6.2.12. Livy

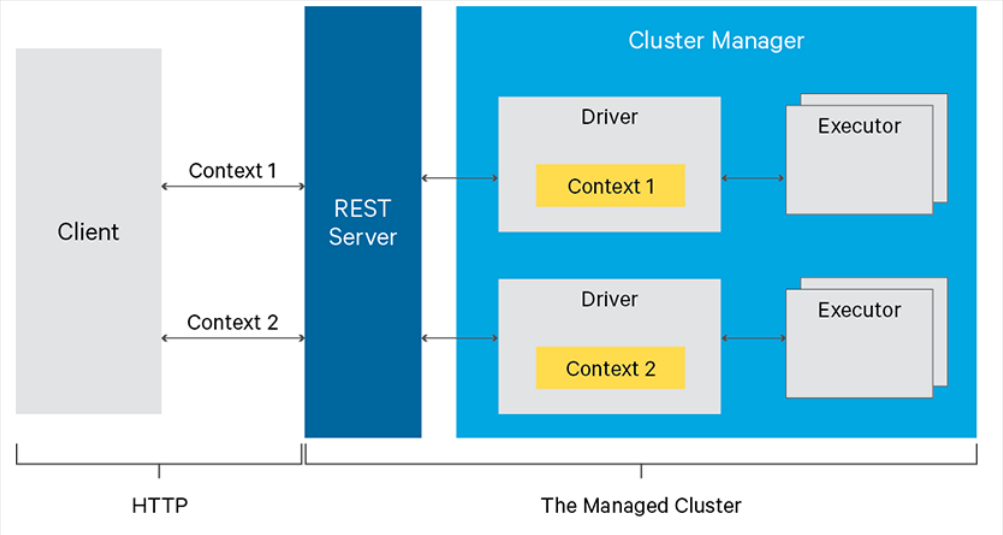
Livy是一个通过REST接口或RPC client库与Spark服务进行交互的服务。Livy支持提交Spark作业或者Spark代码片段，同步或者异步的进行结果检索以及Spark Context上下文管理，Livy简化了Spark和应用程序服务器之间的交互，从而使Spark能够用于交互式Web或移动应用程序。

背景信息

Livy还支持如下功能：

- 长时间运行的Spark Context，可以被多个Spark作业和客户端使用。
- 在多个Spark作业和客户端之间共享缓存RDD。
- 同时管理多个Spark Context。
- 可以通过预编译的JAR包、代码片段、Java API和Scala API等多种方式提交作业。
- 支持一定的安全机制。

Livy的基本架构如下图所示：



提交作业

您可以通过以下方式提交作业：

- REST API
- Programmatic API
- Java API
- Scala API

6.2.13. Phoenix

Apache Phoenix是构建在HBase上的SQL中间层。Phoenix引擎支持使用SQL进行HBase数据的查询。

背景信息

已创建集群，并且选择了Phoenix服务。

创建集群详情，请参见创建集群。

使用Phoenix

1. 使用SSH方式连接集群，详情请参见登录集群。
2. 执行以下命令，切换为hadoop用户。

   ```
   su hadoop
   ```

3. 执行以下命令，进入*bin*目录。

   ```
   cd /usr/lib/phoenix-current/bin
   ```

4. 执行以下命令，使用Phoenix的命令行工具。

   ```
   sqlline.py
   ```

   返回信息下图所示：

   ```
   [root@emr-header-1 bin]# sqlline.py
   Setting property: [incremental, false]
   Setting property: [isolation, TRANSACTION_READ_COMMITTED]
   issuing: !connect jdbc:phoenix: none none org.apache.phoenix.jdbc.PhoenixDriver
   Connecting to jdbc:phoenix:
   SLF4J: Class path contains multiple SLF4J bindings.
   SLF4J: Found binding in [jar:file:/opt/apps/ecm/service/phoenix/4.14.1-1.0.2/package/apache-phoenix-4.14.1-1.0.2-HBase-1.4-bin/phoenix-4.14.1-HBase-1.4-client.jar!/org/slf4j/impl/StaticLoggerBinder.class]
   SLF4J: Found binding in [jar:file:/opt/apps/ecm/service/hadoop/2.8.5-1.6.2/package/hadoop-2.8.5-1.6.2/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
   SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
   21/03/23 10:15:30 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
   21/03/23 10:15:31 WARN shortcircuit.DomainSocketFactory: The short-circuit local reads feature cannot be used because libhadoop cannot be loaded.
   Connected to: Phoenix (version 4.14)
   Driver: PhoenixEmbeddedDriver (version 4.14)
   Autocommit status: true
   Transaction isolation: TRANSACTION_READ_COMMITTED
   Building list of tables and columns for tab-completion (set fastconnect to true to skip)...
   136/136 (100%) Done
   Done
   sqlline version 1.2.0
   0: jdbc:phoenix:>
   ```

5. 您可以使用SQL进行数据查询，常见操作如下：
   - 创建表

     ```
     CREATE TABLE example(
         my_pk bigint not null,
         m.first_name varchar(50),
         m.last_name varchar(50)
         CONSTRAINT pk PRIMARY KEY (my_pk));
     ```

   - 插入数据

     ```
     UPSERT INTO example(my_pk,m.first_name,m.last_name) VALUES(100,'Jack','Ben');
     UPSERT INTO example(my_pk,m.first_name,m.last_name) VALUES(200,'Jack3','Ben3');
     ```

   - 查询数据

     ```
     SELECT * FROM example;
     ```

     返回信息如下所示：

     ```
     +--------+-------------+------------+
     | MY_PK  | FIRST_NAME  | LAST_NAME  |
     +--------+-------------+------------+
     | 100    | Jack        | Ben        |
     | 200    | Jack3       | Ben3       |
     +--------+-------------+------------+
     ```

   - 删除表

     ```
     DROP TABLE example;
     ```

6.2.14. TensorFlow

Data Science集群内置Python 3的Tensorflow 1.15.0版本，可以直接使用。其中Master节点只支持购买CPU资源计算TensorFlow作业，Core节点支持购买CPU或GPU资源计算TensorFlow作业。本文主要介绍如何查看TensorFlow的版本、切换TensorFlow版本以及如何安装Python包。

使用引导

- 查看版本信息
- 切换TensorFlow版本
- 安装Python包

查看版本信息

1. 使用SSH方式登录到集群主节点，详情请参见登录集群。
2. 使用**pip3 list**命令，查看TensorFlow的版本信息。

```
[root@emr-header-1 ~]# pip3 list
Package              Version
-------------------- -------
absl-py              0.9.0
analytics-zoo        0.8.1
astor                0.8.1
BigDL                0.10.0
conda-pack           0.3.1
gast                 0.2.2
google-pasta         0.2.0
grpcio               1.29.0
h5py                 2.10.0
importlib-metadata   1.6.1
Keras-Applications   1.0.8
Keras-Preprocessing  1.1.2
Markdown             3.2.2
numpy                1.18.5
opt-einsum           3.2.1
pip                  19.2.3
protobuf             3.12.2
py4j                 0.10.7
PyHamcrest           2.0.2
pypandoc             1.5
pyspark              2.4.3
setuptools           41.2.0
six                  1.15.0
tensorboard          1.15.0
tensorflow           1.15.0
tensorflow-estimator 1.15.1
termcolor            1.1.0
Werkzeug             1.0.1
wheel                0.34.2
wrapt                1.12.1
zipp                 3.1.0
```

切换TensorFlow版本

1. 下载切换TensorFlow版本的压缩包。

   切换TensorFlow版本压缩包：install_tf_header.tar.gz

2. 使用文件传输工具，上传*install_tf_header.tar.gz*至Data Science集群Master节点的任意目录下。

   > **说明** 本文上传至*/root*目录。

3. 使用SSH方式登录到集群主节点，详情请参见登录集群。

4. 执行如下命令，切换TensorFlow版本。

   i. 解压缩文件。

   ```
   tar -zxvf install_tf_header.tar.gz
   ```

   ii. 切换TensorFlow版本。

   - 命令格式

     ```
     sh install_tf_header.sh <version>
     ```

     `version` 为您需要切换的版本号。

   - 示例：执行如下命令，切换TensorFlow版本为2.0.3。

     ```
     sh install_tf_header.sh 2.0.3
     ```

5. 使用**pip3 list**命令，查看TensorFlow版本号。

```
scipy                1.5.3
Send2Trash           1.5.0
setuptools           41.2.0
six                  1.15.0
tensorboard          2.0.2
tensorflow           2.0.3
tensorflow-estimator 2.0.1
termcolor            1.1.0
terminado            0.8.3
testpath             0.4.4
threadpoolctl        2.1.0
thrift               0.13.0
thrift-sasl          0.4.2
tornado              6.0.4
tqdm                 4.50.2
traitlets            4.3.3
```

TensorFlow版本已经切换为2.0.3版本。

安装Python包

1. 下载安装Python的压缩包。

安装Python的压缩包：install_app_onds.tar.gz

2. 使用文件传输工具，上传*install_app_onds.tar.gz*至Data Science集群Master节点的任意目录下。

   > **说明** 本文上传至*/root*目录。

3. 使用SSH方式登录到集群主节点，详情请参见登录集群。
4. 执行如下命令，在Data Science集群所有节点安装Python包。
   i. 解压缩文件。

   ```
   tar -zxvf install_app_onds.tar.gz
   ```

   ii. 安装Python包。
   - 命令格式

     ```
     sh install_app_onds.sh <package_name> <version>
     ```

     其中涉及参数如下：
     - `package_name` 为您需要安装的Python包名。
     - `version` 为您需要安装Python包的版本号。
   - 示例：执行如下命令，在Data Science集群的所有节点上安装GNU Readline包，版本号为8.0.0。

     ```
     sh install_app_onds.sh gnureadline 8.0.0
     ```

6.2.15. GKS

本文为您介绍如何访问Kubernetes Dashboard UI和Grafana UI页面。

前提条件

- 已创建DataScience集群，并且选择了Kubeflow，详情请参见创建集群。
- 已打开32699和31808端口，详情请参见管理安全组。

> **注意** 设置安全组规则时要针对有限的IP范围。禁止在配置的时候开放0.0.0.0/0规则。

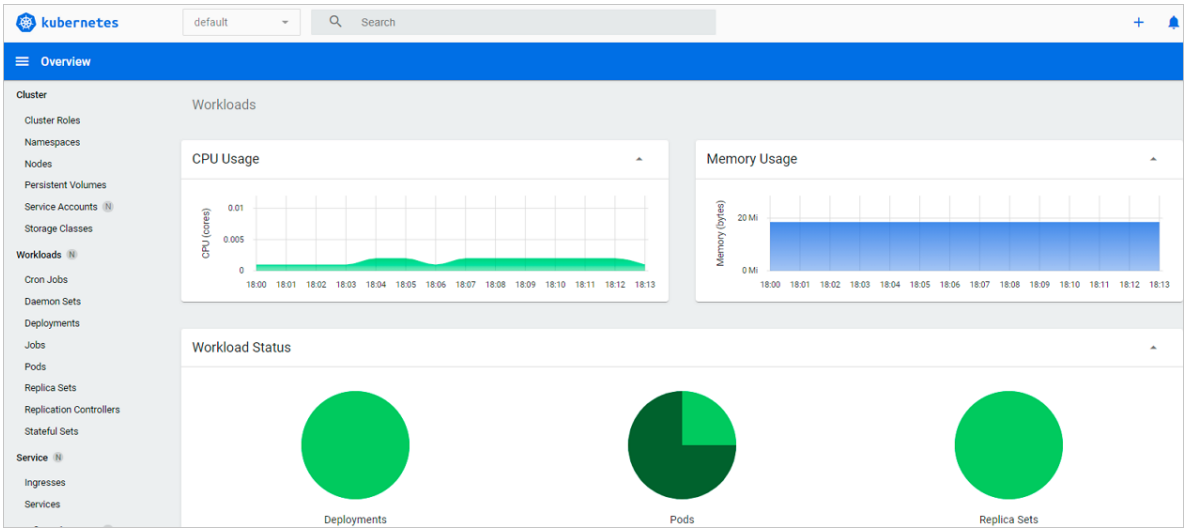
访问Kubernetes Dashboard UI

1. 进入详情页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
2. 在**集群基础信息**页面的**主机信息**区域，查看公网IP地址。

3. 在地址栏中，输入*http:///<yourPublicIPAddress>:32699*，按回车键。

   > **说明** <yourPublicIPAddress>是在步骤2中获取到的公网IP地址。

   进入后，默认页面如下。



访问Grafana UI

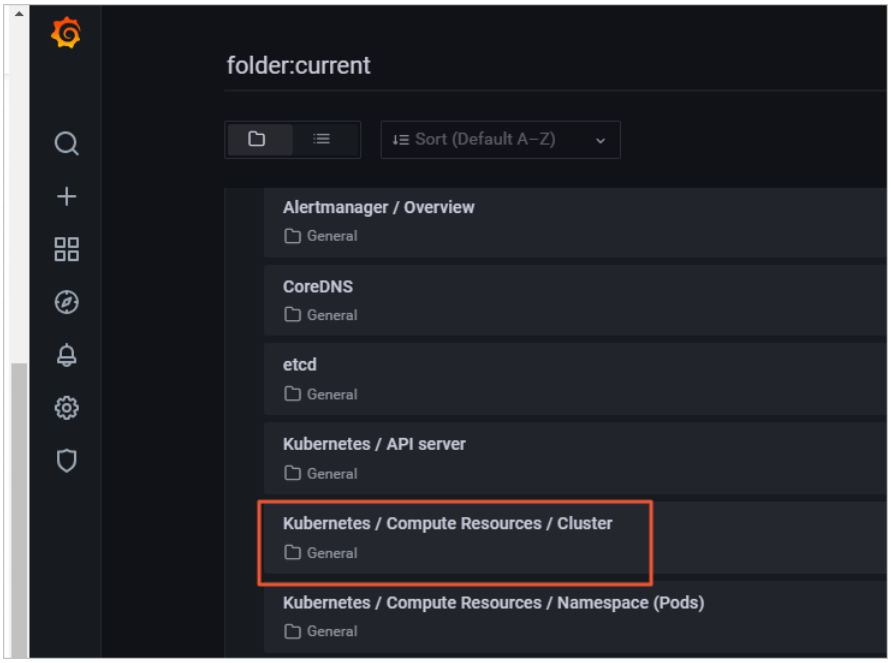
注意 仅EMR-3.35.7及后续版本支持访问Grafana UI。

1. 进入详情页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
2. 在**集群基础信息**页面的**主机信息**区域，查看公网IP地址。

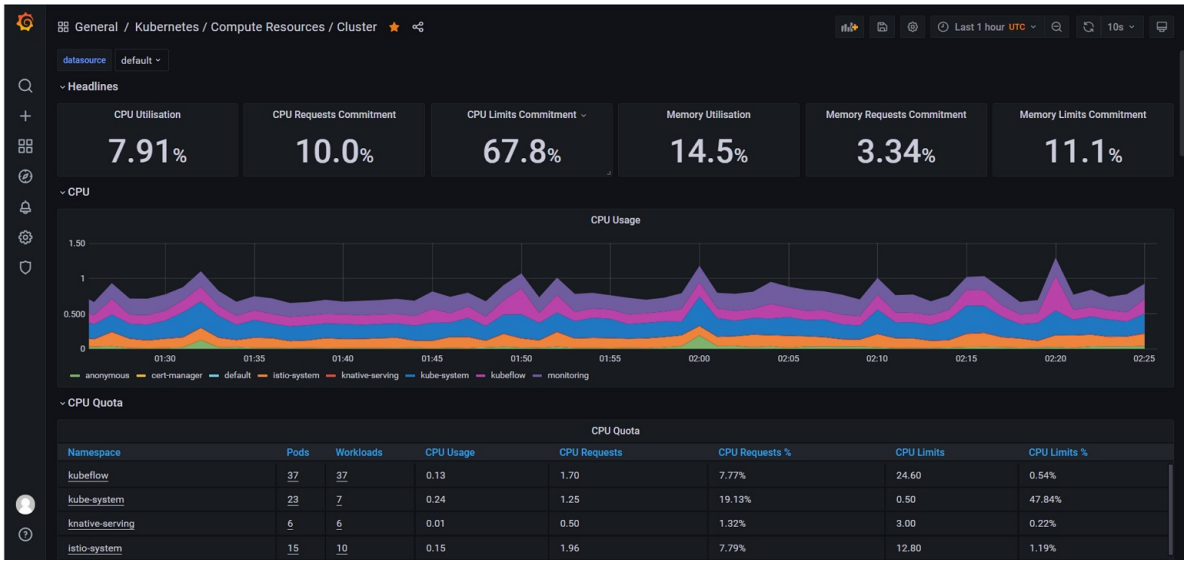
3. 在地址栏中，输入*http://<yourPublicIPAddress>:31808*，按回车键。

   说明 <yourPublicIPAddress>是在步骤2中获取到的公网IP地址。

   默认用户名和密码均为admin。
4. 单击上方的**General**。
5. 单击**General**中的**Kubernetes/Compute Resources/Cluster**。



您可以设置监控报警和查看集群资源情况。



6.2.16. HDFS

6.2.16.1. 概述

HDFS（Hadoop Distributed File System）是一种Hadoop分布式文件系统，具备高度容错特性，支持高吞吐量数据访问，可以在处理海量数据（TB或PB级别以上）的同时最大可能的降低成本。HDFS适用于大规模数据的分布式读写，特别是读多写少的场景。

特性

E-MapReduce集群中HDFS的优势如下：

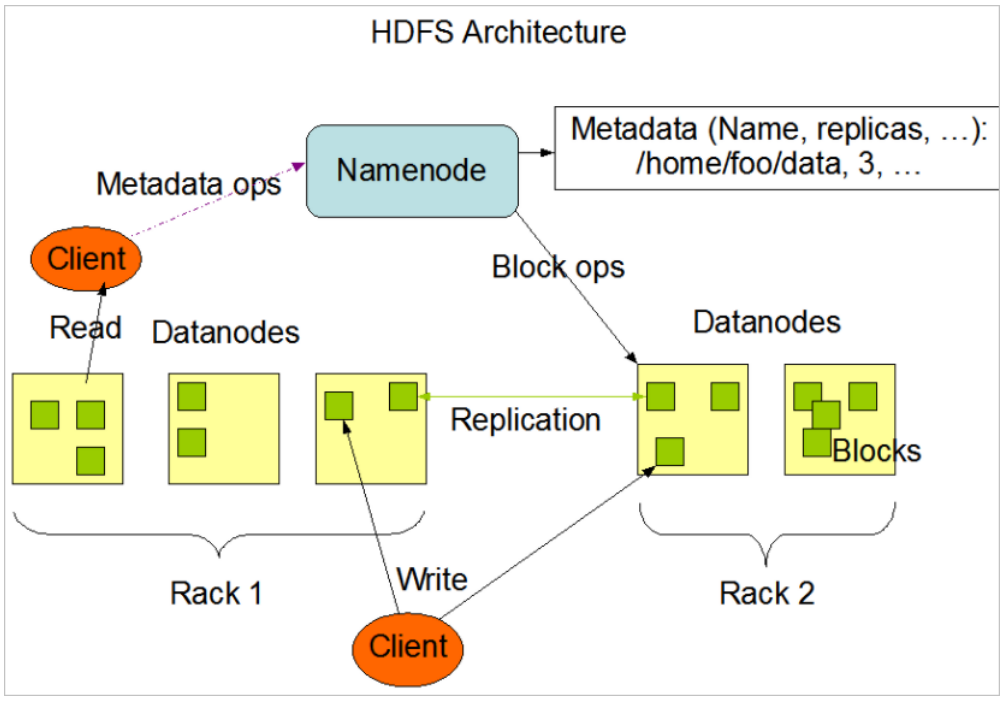
- 具有高容错性和高可扩展性。
- 提供Shell命令接口。
- 提供Web可视化组件管理界面，方便管理。
- 拥有类似于Linux的文件权限管理。
- Locality感知，在分配存储空间时会考虑计算节点的位置。
- 当DataNode之间数据不平衡时，可以进行Rebalance操作。
- 支持滚动重启和升级操作。

架构

HDFS是经典的Master和Slave架构，每一个HDFS集群包括一个NameNode和多个DataNode。

NameNode管理所有文件的元数据信息，并且负责与客户端交互。DataNode负责管理存储在该节点上的文件。每一个上传到HDFS的文件都会被划分为一个或多个数据块，这些数据块根据HDFS集群的数据备份策略被分配到不同的DataNode上，位置信息交由NameNode统一管理。

HDFS架构图如下。



基本概念

名称	描述
NameNode	用于管理文件系统的命名空间、维护文件系统的目录结构树以及元数据信息，记录写入的每个数据块（Block）与其归属文件的对应关系。 此信息以命名空间镜像（FSImage）和编辑日志（EditsLog）两种形式持久化在本地磁盘中。
DataNode	DataNode是文件的实际存放位置。 DataNode会根据NameNode或Client的指令来存储或者提供数据块，并且定期的向NameNode汇报该DataNode存储的数据块信息。
Client	通过Client来访问文件系统，然后由Client与NameNode和DataNode进行通信。Client对外作为文件系统的接口，类似于POSIX。
Blocks	HDFS将文件拆分成128 MB大小的数据块进行存储，这些Block可能存储在不同的节点上。HDFS可以存储更大的单个文件，甚至超过任何一个磁盘所能容纳的大小。一个Block默认存储3个副本（EMR Core节点如果使用云盘，则为2副本），以Block为粒度将副本存储在多个节点上。此方式不仅提高了数据的安全性，而且对于分布式作业可以更好地利用本地的数据进行计算，减少网络传输。
Secondary NameNode	对于非高可用集群，默认会启动一个Secondary NameNode进程。Secondary NameNode的作用是消费EditsLog，定期地合并FsImage和EditsLog，生成新的FsImage文件，降低了NameNode的压力。
高可用	对于高可用集群，默认会启动两个NameNode，一个是Active NameNode，另一个是Standby NameNode，两个NameNode承担不同角色。 Active NameNode负责处理DataNode和Client的请求，Standby NameNode跟Active NameNode一样拥有最新的元数据信息，随时准备在Active NameNode出现异常时接管其服务。如果Active NameNode异常，Standby NameNode会感知到并切换成Active NameNode的角色处理DataNode和Client请求。

6.2.16.2. 基础使用

6.2.16.2.1. 常见命令

您可以在已经创建好的E-MapReduce（简称EMR）集群中，直接使用hadoop fs命令来对HDFS中的文件进行操作。本文为您介绍HDFS的常见命令。

前提条件

- 已创建集群，详情请参见创建集群。
- 已登录集群，详情请参见登录集群。

背景信息

HDFS常见命令如下表所示。

命令	功能
mkdir	在HDFS文件系统中创建目录。
touchz	在HDFS文件系统中新建一个空文件。
ls	在文件或者目录创建完之后，您可以查看指定路径下的文件或目录信息。在查看文件或目录信息的时候需要给出绝对路径。
put	上传本地文件到HDFS的指定路径。
du	显示文件的大小或者目录中所有文件的大小。
cat	查看HDFS上文件的内容。
cp	在HDFS系统中，复制文件或目录到目标路径，并且保持源文件内容或目录结构不变。此命令允许有多个源路径，此时目标路径必须是一个目录。
mv	在HDFS系统中，移动文件或目录到目标路径，并且不保留源文件内容或目录结构。此命令允许有多个源路径，此时目标路径必须是一个目录。
get	下载HDFS指定路径的文件到本地路径。
rm	删除HDFS系统中指定的文件。
rmr	递归删除HDFS系统中指定的目录及其文件。

关于Apache Hadoop的详细介绍，请参见Apache Hadoop官网。

mkdir

在HDFS文件系统中创建目录。

- 语法

```
hadoop fs -mkdir <path1> [path2] ... [pathn]
```

- 示例：

- 在HDFS文件系统中，创建*dir*目录。

  ```
  hadoop fs -mkdir dir
  ```

  您可以通过ls命令查看创建的目录。

  ```
  [root@emr-header-1 ~]# hadoop fs -mkdir dir
  [root@emr-header-1 ~]# hadoop fs -ls
  Found 7 items
  drwx------   - root hadoop          0 2021-05-28 18:43 .Trash
  drwxr-x--x   - root hadoop          0 2021-05-24 12:19 .sparkStaging
  drwxr-x--x   - root hadoop          0 2021-05-28 18:43 dir
  drwxr-x--x   - root hadoop          0 2021-05-28 11:28 dir2
  -rw-r-----   2 root hadoop          8 2021-05-28 14:29 hello.txt
  drwxr-x--x   - root hadoop          0 2021-05-28 11:27 test
  drwxr-x--x   - root hadoop          0 2021-05-28 13:51 tmp
  [root@emr-header-1 ~]#
  ```

- 在*dir*目录下，创建*sub-dir*目录。

  ```
  hadoop fs -mkdir /dir/sub-dir
  ```

  您可以通过ls命令查看创建的目录。

  ```
  [root@emr-header-1 ~]# hadoop fs -mkdir /dir/sub-dir
  [root@emr-header-1 ~]# hadoop fs -ls /dir
  Found 1 items
  drwxr-x--x   - root hadoop          0 2021-05-28 11:40 /dir/sub-dir
  ```

touchz

在HDFS文件系统中新建一个空文件。

- 语法

  ```
  hadoop fs -touchz URI [URI ...]
  ```

- 示例：在HDFS文件系统中新建*/dir/*目录下的*emptyfile.txt*文件。

  ```
  hadoop fs -touchz /dir/emptyfile.txt
  ```

  您可以通过ls命令查看新建的文件。

  ```
  [root@emr-header-1 ~]# hadoop fs -touchz /dir/emptyfile.txt
  [root@emr-header-1 ~]# hadoop fs -ls /dir/
  Found 2 items
  -rw-r-----   2 root hadoop          0 2021-05-28 18:32 /dir/emptyfile.txt
  drwxr-x--x   - root hadoop          0 2021-05-28 14:25 /dir/hello2.txt
  [root@emr-header-1 ~]#
  ```

ls

在文件或者目录创建完之后，您可以查看指定路径下的文件或目录信息。在查看文件或目录信息的时候需要给出绝对路径。

> **说明** hadoop fs没有进入某个目录下的概念。

- 语法

  ```
  hadoop fs -ls <path>
  ```

- 示例：
  - 查看文件*hello.txt*的信息。

    ```
    hadoop fs -ls hello.txt
    ```

    ```
    [root@emr-header-1 ~]# hadoop fs -ls hello.txt
    -rw-r-----   2 root hadoop          0 2021-05-28 11:54 hello.txt
    ```

  - 查看*/dir/sub-dir*目录的信息。

    ```
    hadoop fs -ls /dir/sub-dir
    ```

    ```
    [root@emr-header-1 ~]# hadoop fs -ls /dir/sub-dir
    Found 1 items
    -rw-r-----   2 root hadoop         25 2021-05-28 13:44 /dir/sub-dir/hello.txt
    ```

put

上传本地文件到HDFS的指定路径。

- 语法

  ```
  hadoop fs -put <path1> <path2>
  ```

- 示例：上传本地文件*hello.txt*至HDFS的*/dir/sub-dir*路径下。

```
hadoop fs -put hello.txt /dir/sub-dir
```

您可以通过ls命令查看文件上传的情况。

```
[root@emr-header-1 ~]# hadoop fs -put hello.txt /dir/sub-dir
[root@emr-header-1 ~]# hadoop fs -ls /dir/sub-dir
Found 1 items
-rw-r-----   2 root hadoop         25 2021-05-28 13:44 /dir/sub-dir/hello.txt
```

du

显示文件的大小或者目录中所有文件的大小。

- 语法

```
hadoop fs -du <path>
```

- 示例
  - 查看文件的大小。

```
hadoop fs -du hello.txt
```

```
[root@emr-header-1 ~]# hadoop fs -du hello.txt
61  hello.txt
```

  - 查看目录下所有文件的大小。

```
hadoop fs -du /dir
```

```
[root@emr-header-1 ~]# hadoop fs -du /dir
0   /dir/hello2.txt
25  /dir/sub-dir
```

cat

查看HDFS上文件的内容。

- 语法

```
hadoop fs -cat <path>
```

- 示例：
  - 查看*hello.txt*文件的内容。

```
hadoop fs -cat hello.txt
```

```
[root@emr-header-1 ~]# hadoop fs -cat hello.txt
welcome
```

  - 查看*/dir/sub-dir/*目录下*hello_world.txt*文件的内容。

```
hadoop fs -cat /dir/sub-dir/hello_world.txt
```

```
[root@emr-header-1 ~]# hadoop fs -cat /dir/sub-dir/hello_world.txt
hello world!
```

cp

在HDFS系统中，复制文件或目录到目标路径，并且保持源文件内容或目录结构不变。此命令允许有多个源路径，此时目标路径必须是一个目录。

> **说明** 您也可以使用此命令对文件重命名，以保留源文件或目录。

- 语法

```
hadoop fs -cp <path1> <path2>
```

- 示例：复制*/dir/sub-dir/*目录下的文件*hello_world.txt*至*/tmp*目录下。

```
hadoop fs -cp /dir/sub-dir/hello_world.txt /tmp
```

您可以通过ls命令查看文件复制的情况。

```
[root@emr-header-1 ~]# hadoop fs -cp /dir/sub-dir/hello_world.txt /tmp
[root@emr-header-1 ~]# hadoop fs -ls /tmp
Found 4 items
drwxrwxrwx   - root   hadoop          0 2021-05-24 12:18 /tmp/hadoop-yarn
-rw-r-----   2 root   hadoop         13 2021-05-28 14:56 /tmp/hello_world.txt
drwx-wx-wx   - hadoop hadoop          0 2021-05-24 12:24 /tmp/hive
drwxrwxrwt   - hadoop hadoop          0 2021-05-24 12:20 /tmp/logs
```

mv

在HDFS系统中，移动文件或目录到目标路径，并且不保留源文件内容或目录结构。此命令允许有多个源路径，此时目标路径必须是一个目录。

- 语法

```
hadoop fs -mv <path1> <path2>
```

- 示例
  - 移动*/tmp/*目录下的文件*hello_world2.txt*至*/dir/sub-dir/*目录下。

    ```
    hadoop fs -mv /tmp/hello_world2.txt /dir/sub-dir/
    ```

    您可以通过ls命令查看文件移动的情况。

    ```
    [root@emr-header-1 ~]# hadoop fs -mv /tmp/hello_world2.txt /dir/sub-dir/
    [root@emr-header-1 ~]#  hadoop fs -ls /dir/sub-dir/
    Found 2 items
    -rw-r-----   2 root hadoop         13 2021-05-28 14:48 /dir/sub-dir/hello_world.txt
    -rw-r-----   2 root hadoop         13 2021-05-28 15:04 /dir/sub-dir/hello_world2.txt
    [root@emr-header-1 ~]#
    ```

  - 移动*/tmp/*路径下的*test*目录至*/dir/sub-dir/*目录下。

    ```
    hadoop fs -mv /tmp/test /dir/sub-dir/
    ```

    您可以通过ls命令查看目录移动的情况。

    ```
    [root@emr-header-1 ~]# hadoop fs -mv /tmp/test /dir/sub-dir/
    [root@emr-header-1 ~]#  hadoop fs -ls /tmp/
    Found 4 items
    drwxrwxrwx   - root      hadoop          0 2021-05-24 12:18 /tmp/hadoop-yarn
    -rw-r-----   2 root      hadoop         13 2021-05-28 14:56 /tmp/hello_world.txt
    drwx-wx-wx   - anonymous hadoop          0 2021-05-28 15:50 /tmp/hive
    drwxrwxrwt   - hadoop    hadoop          0 2021-05-24 12:20 /tmp/logs
    [root@emr-header-1 ~]#  hadoop fs -ls /dir/sub-dir/
    Found 4 items
    -rw-r-----   2 root   hadoop         13 2021-05-28 14:48 /dir/sub-dir/hello_world.txt
    -rw-r-----   2 root   hadoop         13 2021-05-28 15:04 /dir/sub-dir/hello_world2.txt
    drwx-wx-wx   - hadoop hadoop          0 2021-05-24 12:24 /dir/sub-dir/hive
    drwxr-x--x   - root   hadoop          0 2021-05-28 15:54 /dir/sub-dir/test
    [root@emr-header-1 ~]#
    ```

get

下载HDFS指定路径的文件到本地路径。

- 语法

```
hadoop fs -get <path1> <path2>
```

- 示例：下载HDFS系统中*/dir/sub-dir/*目录下的文件*hello_world2.txt*至本地的*/emr*路径下。

```
hadoop fs -get /dir/sub-dir/hello_world2.txt /emr
```

您可以通过ls命令查看文件下载的情况。

```
[root@emr-header-1 ~]# hadoop fs -get /dir/sub-dir/hello_world2.txt /root/emr
[root@emr-header-1 ~]# cd emr
[root@emr-header-1 emr]# ls
hello_world2.txt
[root@emr-header-1 emr]#
```

rm

删除HDFS系统中指定的文件。

- 语法

```
hadoop fs -rm <path>
```

- 示例：删除HDFS系统中*/dir/sub-dir/*目录下的文件*hello_world2.txt*。

```
hadoop fs -rm /dir/sub-dir/hello_world2.txt
```

您可以通过ls命令查看文件删除的情况。

```
[root@emr-header-1 ~]# hadoop fs -rm /dir/sub-dir/hello_world2.txt
21/05/28 16:48:27 INFO fs.TrashPolicyDefault: Moved: 'hdfs://emr-header-1.cluster-230659:9000/dir/sub-dir/hello_wo
rld2.txt' to trash at: hdfs://emr-header-1.cluster-230659:9000/user/root/.Trash/Current/dir/sub-dir/hello_world2.t
xt
[root@emr-header-1 ~]# hadoop fs -ls /dir/sub-dir/
Found 3 items
-rw-r-----   2 root   hadoop         13 2021-05-28 14:48 /dir/sub-dir/hello_world.txt
drwx-wx-wx   - hadoop hadoop          0 2021-05-24 12:24 /dir/sub-dir/hive
drwxr-x--x   - root   hadoop          0 2021-05-28 15:54 /dir/sub-dir/test
```

rmr

递归删除HDFS系统中指定的目录及其文件。

- 语法

```
hadoop fs -rmr <path>
```

- 示例：递归删除HDFS系统中*/dir/*下的*sub-dir*目录及其文件。

```
 hadoop fs -rmr /dir/sub-dir/
```

您可以通过ls命令查看目录删除的情况。

```
[root@emr-header-1 ~]# hadoop fs -rmr /dir/sub-dir/
rmr: DEPRECATED: Please use '-rm -r' instead.
rmr: `/dir/sub-dir/': No such file or directory
[root@emr-header-1 ~]# hadoop fs -ls /dir/sub-dir/
ls: `/dir/sub-dir/': No such file or directory
[root@emr-header-1 ~]# hadoop fs -ls /dir/
Found 2 items
-rw-r-----   2 root hadoop          0 2021-05-28 16:07 /dir/emptyfile.txt
drwxr-x--x   - root hadoop          0 2021-05-28 14:25 /dir/hello2.txt
[root@emr-header-1 ~]#
```

6.2.16.3. 高阶使用

6.2.16.3.1. 开启权限认证

HDFS开启了权限控制后，当您访问HDFS需要有合法的权限才能正常操作HDFS，例如读取数据和创建文件夹等。本文为您介绍如何开启HDFS的权限控制。

前提条件

已创建集群，详情请参见创建集群。

背景信息

Hadoop提供了以下两种用于决定用户身份的操作模式：

- 简单模式（Simple）：用户的身份由与HDFS建立链接的客户端操作系统决定。在类Unix系统中，等同于`whoami`命令。
- Kerberos集群模式：客户端的身份由他自己的Kerberos证书决定。

您可以在创建EMR集群时，开启Kerberos集群模式。Kerberos详情信息，请参见概述。

注意事项

- umask值可以根据实际需求修改。
- HDFS是一个基础的服务，Hive和HBase等都是基于HDFS，所以在配置其他上层服务时，需要提前配置好HDFS的权限控制。
- 在HDFS开启权限后，需要设置好服务的日志路径。例如，Spark的日志路径为*/spark-history*、YARN的日志路径*/tmp/$user/*等。
- 您可以对文件夹设置Stickybit，防止除了superuser、file owner或dir owner之外的用户，删除该文件夹中的文件或文件夹（即使用户对该文件夹拥有rwx权限）。

  例如，在HDFS客户端中，使用具有HDFS管理员权限的用户，执行如下命令，修改*/user*的目录权限。

```
hdfs dfs -chmod 1777 </user>
```

此处将权限修改为1777，即在权限处增加1，表示增加目录的粘性，即只有创建的用户才可以删除此目录。

操作步骤

> 注意 对于开启Kerberos模式的集群，已经默认设置了HDFS的权限（umask为027），无需配置和重启服务。未开启Kerberos模式的集群需要按照以下方式添加配置并重启服务。

1. 进入集群详情页面。
   i. 登录阿里云E-MapReduce控制台。

ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
iii. 单击上方的**集群管理**页签。
iv. 在**集群管理**页面，单击相应集群所在行的**详情**。

2. 在左侧导航栏，选择**集群服务 > HDFS**。
3. 修改配置。
   i. 在HDFS服务页面，单击**配置**页签。
   ii. 在**配置搜索**中，输入待修改的配置项，单击图标。
   iii. 在**服务配置**区域，根据实际需求修改配置项。

参数	描述
dfs.permissions.enabled	默认值为false。开启权限检查，修改为true。
dfs.datanode.data.dir.perm	默认值为755。DataNode使用的本地文件夹路径的权限。
fs.permissions.umask-mode	权限掩码，在新建文件或文件夹的时候的默认权限值。 ■ 简单模式：默认值为022，对应新建文件权限为644（0666&^022=644），新建文件夹权限为755（0777&^022=755）。 ■ Kerberos集群模式：默认值为027，对应新建文件权限为640，新建文件夹权限为750。
dfs.namenode.acls.enabled	默认值为false。打开ACL控制，不仅可以对用户或用户组进行权限控制，还可以对其他用户进行设置。 设置ACL相关命令有 `hadoop fs -getfacl [-R] <path>` 和 `hadoop fs -setfacl [-R] [-b |-k -m |-x <acl_spec> <path>] |[--set <acl_spec> <path>]` 。
dfs.permissions.superusergroup	默认值为hadoop。超级用户组的名称。属于该组的用户都具有超级用户的权限。

4. 重启服务。
   i. 在HDFS服务页面的右上角，选择**操作 > 重启All Components**。
   ii. 在**执行集群操作**对话框中，输入**执行原因**，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。
   您可以单击右上角的**查看操作历史**查看任务进度，等待任务完成。

示例

1. 使用SSH方式登录集群，详情请参见登录集群。
2. 执行以下命令，切换为已创建的test用户。

```
su test
```

3. 执行以下命令，使用test用户创建目录。

```
hadoop fs -mkdir /tmp/test
```

4. 执行以下命令，查看创建的目录的权限。

```
hadoop fs -ls /tmp
```

返回如下类似信息。

```
Found 4 items
drwxrwxrwx   - root   hadoop          0 2021-06-08 13:14 /tmp/hadoop-yarn
drwx-wx-wx   - hadoop hadoop          0 2021-06-16 15:54 /tmp/hive
drwxrwxrwt   - hadoop hadoop          0 2021-06-08 13:16 /tmp/logs
drwxr-x--x   - test   hadoop          0 2021-06-16 17:15 /tmp/test
```

5. 执行以下命令，给目录设置ACL权限并授权给foo用户rwx权限。

```
hadoop fs -setfacl -m user:foo:rwx /tmp/test
```

6. 执行以下命令，查看目录权限。

```
hadoop fs -ls /tmp/
```

返回如下类似信息。

```
Found 4 items
drwxrwxrwx   - root   hadoop          0 2021-06-08 13:14 /tmp/hadoop-yarn
drwx-wx-wx   - hadoop hadoop          0 2021-06-16 15:54 /tmp/hive
drwxrwxrwt   - hadoop hadoop          0 2021-06-08 13:16 /tmp/logs
drwxrwx--x+  - test   hadoop          0 2021-06-16 17:15 /tmp/test
```

说明 权限后面带了+号的表示设置了ACL权限，例如drwxrwx--x+。

7. 执行以下命令，查看ACL权限。

```
hadoop fs -getfacl /tmp/test
```

返回如下类似信息。

```
# file: /tmp/test
# owner: test
# group: hadoop
user::rwx
user:foo:rwx
group::r-x
mask::rwx
other::--x
```

6.2.16.3.2. 配额（Quotas）

HDFS允许管理员可以对所使用的名称数量和单个目录使用的空间量设置配额，名称配额与空间配额分别独立运作，但是两种配额的管理和实现方式非常相似。本文为您介绍HDFS的名称配额和空间配额两种方式以及如何进行配额操作。

名称配额（Name Quotas）

名称配额是对当前目录树中的文件和目录名称数量的硬性限制。创建文件或目录时，如果超出配额，则文件和目录创建失败。配额是一个目录的属性，不仅在创建时会检查配额，在重命名时也会检查配额。如果一个目录已经设置了配额，则该目录执行重命名操作后，该目录的配额亦然生效。

即使目录违反了新的配额，对目录设置新的配额仍然可以成功。新建的目录初始状态是没有设置配额的。最大配额值为Long.Max_Value。例如，设置配额为1会使得一个目录为空，因为目录本身占用一个额度。

空间配额（Space Quotas）

空间配额是对当前目录树中的文件所使用的字节数量的硬性限制。如果额度不允许写入一个完整的块，则块分配会失败。一个块的所有副本都会计入额度中。例如，一个文件大小为N，有三个副本，则占用的额度为3N。配额是一个目录的属性，不仅在创建时会检查配额，在重命名时也会检查配额。对于重命名的目录，配额不变。如果重命名操作与设置的配置冲突，则重命名操作失败。

即使目录违反了新的配额，对目录设置新的配额仍然可以成功。新建的目录初始状态是没有设置配额的。最大配额值为Long.Max_Value。当配额为0时仍然允许创建文件，但是不能向该文件中写内容。目录不占用磁盘空间，因此不计入空间配额中。

命令示例

- 管理员命令

  管理员可以通过以下命令来管理额度：

  - 设置目录的名称配额为N。

    ```
    hdfs dfsadmin -setQuota <N> <directory>...<directory>
    ```

    当N是非正数、目录不存在或目录马上超过新配额时设置失败。

  - 删除目录的名称配额。

    ```
    hdfs dfsadmin -clrQuota <directory>...<directory>
    ```

  - 设置目录的空间配额为N字节。

    ```
    hdfs dfsadmin -setSpaceQuota <N> <directory>...<directory>
    ```

  - 删除目录的空间配额。

    ```
    hdfs dfsadmin -clrSpaceQuota <directory>...<directory>
    ```

- 报告命令

  报告配额值和当前使用的名称总数以及总的字节数。

  ```
  hadoop fs -count -q [-h] [-v] [-t [comma-separated list of storagetypes]] <directory>...<directory>
  ```

6.2.16.3.3. 快照

HDFS快照（Snapshots）是文件系统在某一时刻的只读副本。快照可以在文件系统的一个分支或者整个文件系统上生成。快照常用来备份数据，防止错误性的操作。本文为您介绍HDFS快照的目录、路径以及相关的快照操作。

背景信息

HDFS快照的特征如下：

- 快照的创建是瞬时的：时间复杂度为O(1)，不包括INode查找时间。
- 仅当修改快照相关的数据时才会使用额外的内存：内存使用复杂度为O(M)，其中M是修改的文件或目录的数量。
- DataNode中的数据块不会被复制：快照文件仅记录块列表和文件大小，不涉及数据复制。
- 快照不会对常规的HDFS操作造成不利影响：修改操作按照时间倒序记录，以便可以直接访问当前最新的数据。快照数据是通过当前数据减去修改的部分计算得到的。

快照目录

当目录被设置为可被快照时才会生成快照。一个快照目录可以同时容纳65536个快照。快照目录的数量是没有限制的。管理员可以将任何目录设置为快照。如果一个目录中有快照，则在删除所有快照之前，既不能删除也不能重命名该目录。

不允许及联的快照目录。如果一个目录的父目录或者子目录是快照目录，则不能将该目录设置为快照。

快照路径

对于一个快照目录，访问时需要添加*/.snapshot*后缀。例如，如果*/foo*是一个快照目录，*/foo/bar*是*/foo*下的文件或者目录，*/foo*有一个快照s0，则*/foo/.snapshot/s0/bar*是*/foo/bar*的快照副本。常用的API和CLI可以通过在*.snapshot*目录下完成，操作快照的命令示例如下：

- 列出快照目录下的所有快照

```
hdfs dfs -ls /foo/.snapshot
```

- 列出快照s0中的所有文件

```
hdfs dfs -ls /foo/.snapshot/s0
```

- 从快照s0中复制文件

```
hdfs dfs -cp -ptopax /foo/.snapshot/s0/bar /tmp
```

> **说明** 该示例使用 `-ptopax` 参数来保留时间戳、所有权、权限、ACL和XAttrs。

快照操作

- 管理员操作

> **说明** 以下管理员操作需要超级用户权限。

  - 允许快照

    允许创建目录的快照。如果操作成功完成，该目录将变为可快照目录。

```
hdfs dfsadmin -allowSnapshot <path>
```

  - 禁止快照

    禁止创建目录的快照。在禁止快照之前，必须删除该目录的所有快照。

```
hdfs dfsadmin -disallowSnapshot <path>
```

- 用户操作

> **说明** HDFS超级用户可以执行以下所有操作。

  - 创建快照

    创建快照目录的快照。该操作需要快照目录的所有者权限。

```
hdfs dfs -createSnapshot <path> [snapshotName]
```

> **说明** 本文代码示例中的 `[snapshotName]` 表示快照名称，是一个可选参数。当其省略时，默认的名称是使用时间戳syyyyMMdd-HHmmss.SSSS的格式表示，例如，s20130412-151029.033。

- 删除快照

  从快照目录中删除快照。该操作需要快照目录的所有者权限。

  ```
  hdfs dfs -deleteSnapshot <path> <snapshotName>
  ```

- 重命名快照

  重命名一个快照。该操作需要快照目录的所有者权限。

  ```
  hdfs dfs -renameSnapshot <path> <oldName> <newName>
  ```

  > **说明** 本文代码示例中的 `<oldName>` 表示原快照名称，`<newName>` 表示新快照名称。

- 获取快照目录列表

  获取当前用户拥有快照权限的所有快照目录。

  ```
  hdfs lsSnapshottableDir
  ```

- 获取快照区别报告

  获取两个快照之间的区别。该操作需要快照内所有文件或目录的读权限。

  ```
  hdfs snapshotDiff <path> <fromSnapshot> <toSnapshot>
  ```

  > **说明** 本文代码示例中的 `<fromSnapshot>` 表示原快照，`<toSnapshot>` 表示待对比的快照。

6.2.16.3.4. Balancer工具

Balancer工具可以用来分析块的分布情况，并且可以重新分配DataNode中的数据。本文通过为您介绍如何使用Balancer工具以及Balancer的主要调优参数。

前提条件

- 已创建集群，详情请参见创建集群。
- 已登录集群，详情请参见登录集群。

使用Balancer

HDFS命令语句如下。

```
hdfs balancer
[-threshold <threshold>]
[-policy <policy>]
[-exclude [-f <hosts-file> | <comma-separated list of hosts>]]
[-include [-f <hosts-file> | <comma-separated list of hosts>]]
[-source [-f <hosts-file> | <comma-separated list of hosts>]]
[-blockpools <comma-separated list of blockpool ids>]
[-idleiterations <idleiterations>]
```

Balancer主要参数如下表。

参数	说明
threshold	磁盘容量的百分数。 默认值为10%，表示上下浮动10%。 当集群总使用率较高时，需要调小Threshold，避免阈值过高。 当集群新增节点较多时，您可以适当增加Threshold，使数据从高使用率节点移向低使用率节点。
policy	平衡策略。支持以下策略： • datanode（默认）：当每一个DataNode是平衡的时候，集群就是平衡的。 • blockpool：当每一个DataNode中的blockpool是平衡的，集群就是平衡的。
exclude	Balancer排除特定的DataNode。

参数	说明
include	Balancer仅对特定的DataNode进行平衡操作。
source	仅选择特定的DataNode作为源节点。
blockpools	Balancer仅在指定的blockpools中运行。
idleterations	最多允许的空闲循环次数，否则退出。覆盖默认的5次。

示例

本示例以阿里云E-MapReduce集群为例。

1. 登录待配置集群的任意节点。

   本示例登录Master节点，详情请参见登录集群。

2. 切换到hdfs用户并执行Balancer参数。

   ```
   /usr/lib/hadoop-current/sbin/start-balancer.sh -threshold 10
   ```

3. 执行以下命令，进入*hadoop-hdfs*目录。

   ```
   cd /var/log/hadoop-hdfs
   ```

4. 执行 `ll` 命令，查看Balancer日志。

   返回信息类似如下截图。

   ```
   [hdfs@emr-header-1 hadoop-hdfs]$ ll
   total 266440
   -rw-r--r-- 1 hdfs hadoop      2519 Jun 30 10:53 hadoop-hdfs-balancer-emr-header-1.cluster-23[redacted].log
   -rw-r--r-- 1 hdfs hadoop       975 Jun 30 10:53 hadoop-hdfs-balancer-emr-header-1.cluster-23[redacted].out
   -rw-r--r-- 1 hdfs hadoop  21824254 Jun 30 10:53 hadoop-hdfs-namenode-emr-header-1.cluster-23[redacted].log
   -rw-r--r-- 1 hdfs hadoop     10723 Jun 16 17:17 hadoop-hdfs-namenode-emr-header-1.cluster-23[redacted].out
   -rw-r--r-- 1 hdfs hadoop     10723 Jun  8 13:14 hadoop-hdfs-namenode-emr-header-1.cluster-23[redacted].out.1
   ```

5. 执行以下命令，查看Balancer运行情况。

   ```
   tailf /var/log/hadoop-hdfs/hadoop-hdfs-balancer-emr-header-xx.cluster-xxx.log
   ```

   当提示信息包含 `Successfully` 字样时，表示执行成功。

   > **说明** 代码中的 `hadoop-hdfs-balancer-emr-header-xx.cluster-xxx.log` 为步骤4获取到的日志名称。

Balancer调优参数

执行Balancer会占用一定的系统资源，建议在业务空闲期执行。默认情况下，不需要对Balancer参数进行额外调整。当需要时，可以对客户端和DataNode两类配置进行调整。

- 客户端配置，配置入口在HDFS服务的**配置**页面，您可以单击**hdfs-site**页签，然后单击**自定义配置**，添加以下参数，配置完成后重新执行Balancer即可生效。添加参数详情，请参见添加组件参数。

参数	描述
dfs.balancer.dispatcherThreads	Balancer在移动Block之前，每次迭代时查询出一个Block列表，分发给Mover线程使用。 **说明** dispatcherThreads是该分发线程的个数，默认为200。
dfs.balancer.rpc.per.sec	默认值为20，即每秒发送的RPC数量为20。 因为分发线程调用大量getBlocks的RPC查询，所以为了避免NameNode由于分发线程压力过大，需要控制分发线程RPC的发送速度。 例如，您可以在负载高的集群调整参数值，减小10或者5，对整体移动进度不会产生特别大的影响。
dfs.balancer.getBlocks.size	Balancer会在移动Block前，每次迭代时查询出一个Block列表，给Mover线程使用，默认Block列表中Block的大小为2 GB。因为getBlocks过程会对RPC进行加锁，所以您可以根据NameNode压力进行调整。

参数	描述
dfs.balancer.moverThreads	默认值为1000。 Balancer处理移动Block的线程数，每个Block移动时会使用一个线程。

- DataNode配置，配置入口在HDFS服务的**配置**页面，您可以单击**hdfs-site**页签，然后单击**自定义配置**，添加以下参数，配置完成后重启DataNode，再次执行Balancer即可生效。添加参数详情，请参见添加组件参数。

参数	描述
dfs.datanode.balance.bandwidthPerSec	指定DataNode用于Balancer的带宽，通常推荐设置为100 MB/s，您也可以通过dfsadmin -setBalancerBandwidth 参数进行适当调整，无需重启DataNode。 例如，在负载低时，增加Balancer的带宽。在负载高时，减少Balancer的带宽。
dfs.datanode.balance.max.concurrent.moves	默认值为5。 指定DataNode节点并发移动的最大个数。通常考虑和磁盘数匹配，推荐在DataNode端设置为 `4 * 磁盘数` 作为上限，可以使用Balancer的值进行调节。 例如：一个DataNode有28块盘，在Balancer端设置为28，DataNode端设置为 `28 * 4` 。具体使用时根据集群负载适当调整。在负载较低时，增加concurrent数；在负载较高时，减少concurrent数。

6.2.16.3.5. HaAdmin工具

集群启动高可用特性后，您可以使用HaAdmin工具来管理HDFS集群。本文为您介绍如何使用HaAdmin工具。

前提条件

- 已创建集群，并开启了高可用，详情请参见创建集群。

- 已登录集群，详情请参见登录集群。

背景信息

在Hadoop 2.0.0之前，NameNode在HDFS集群中都是以单节点的形式存在。每个集群只有一个NameNode，如果此NameNode不可用，整个集群都会变成不可用的状态，直到NameNode重新与集群建立连接。

单一NameNode主要从两个方面影响HDFS集群的可用性：

- 当发生一个计划之外的事件，例如机器宕机，集群将会处于不可用状态，直到手动重启NameNode。
- 有计划的维护事件，例如软件或硬件升级，也会使得集群存在一个不可用的窗口期。

HDFS高可用特性解决了上述问题，通过提供了两个冗余的NameNode以主动或被动的方式用于热备，使得集群既可以从机器宕机中快速恢复，也可以优雅的在有计划的维护时快速恢复。

使用HaAdmin

HaAdmin工具的常用命令如下所示：

- 查看所有NameNode的状态。

```
hdfs haadmin -getAllServiceState
```

- 检查指定NameNode的健康情况。

```
hdfs haadmin -checkHealth <serviceId>
```

- 在两个NameNode中初始化一个故障转移操作。

```
hdfs haadmin -failover [--forcefence] [--forceactive] <serviceId>
```

- 获取serviceId的状态，判断指定的NameNode是Active或者Standby。

```
hdfs haadmin -getServiceStat <serviceId>
```

- 将指定的NameNode状态转换成Active。

```
hdfs haadmin -transitionToActive <serviceId> [--forceactive]
```

- 将指定的NameNode状态转换成Standby。

```
hdfs haadmin -transitionToStandby <serviceId>
```

6.2.16.3.6. HDFS授权

HDFS开启了权限控制后，用户访问HDFS需要有合法的权限才能正常操作HDFS，如读取数据和创建文件夹等。

进入配置页面

1. 登录阿里云E-MapReduce控制台。
2. 在顶部菜单栏处，根据实际情况选择地域和资源组。
3. 单击上方的**集群管理**页签。
4. 在**集群管理**页面，单击相应集群所在行的**详情**。
5. 在左侧导航栏单击**集群服务 > HDFS**。
6. 单击**配置**页签。

> **说明**
> - 对于Kerberos安全集群，已经默认设置了HDFS的权限（umask 设置为 027），无需配置和重启服务。
> - 对于非Kerberos安全集群需要添加配置并重启服务。

添加配置

HDFS权限相关的配置如下：

- **dfs.permissions.enabled**

  开启权限检查，即使该值为**false**， chmod/chgrp/chown/setfacl操作还是会进行权限检查。

- **dfs.datanode.data.dir.perm**

  datanode使用的本地文件夹路径的权限，默认**755**。

- **fs.permissions.umask-mode**
  - 权限掩码，在新建文件/文件夹的时候的默认权限设置。
  - 新建文件：0666 & ^umask。
  - 新建文件夹：0777 & ^umask。
  - 默认umask值为022，即新建文件权限为644（666&^022=644），新建文件夹权限为755（777&^022=755）。
  - EMR 的 Kerberos 安全集群默认设置为027，对应新建文件权限为640，新建文件夹权限为750。

- **dfs.namenode.acls.enabled**
  - 打开 ACL 控制，打开后除了可以对owner/group进行权限控制外，还可以对其它用户进行设置。

- 设置 ACL 相关命令：

```
hadoop fs -getfacl [-R] <path>
hadoop fs -setfacl [-R] [-b |-k -m |-x <acl_spec> <path>] |[--set <acl_spec>   <path>]
```

如：

```
su test
 #test用户创建文件夹
 hadoop fs -mkdir /tmp/test
 #查看创建的文件夹的权限
 hadoop fs -ls /tmp
 drwxr-x---   - test   hadoop          0 2017-11-26 21:18 /tmp/test
 #设置acl，授权给foo用户rwx
 hadoop fs -setfacl -m user:foo:rwx /tmp/test
 #查看文件权限(+号表示设置了ACL)
 hadoop fs -ls  /tmp/
 drwxrwx---+  - test   hadoop          0 2017-11-26 21:18 /tmp/test
 #查看acl
  hadoop fs -getfacl  /tmp/test
  # file: /tmp/test
# owner: test
# group: hadoop
user::rwx
user:foo:rwx
group::r-x
mask::rwx
other::---
```

- dfs.permissions.superusergroup

  超级用户组，属于该组的用户都具有超级用户的权限。

重启HDFS服务

1. 在**集群服务 > HDFS**页面，单击右上角的**操作 > 重启 All Components**。
2. 执行集群操作。
   i. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
   ii. 在**确认**对话框，单击**确定**。
   iii. 单击右上角**查看操作历史**查看任务进度，等待任务完成。

其它

- umask值可以根据需求自行修改。
- HDFS是一个基础的服务，Hive/HBase等都是基于HDFS，所以在配置其它上层服务时，需要提前配置好HDFS的权限控制。
- 在HDFS开启权限后，需要设置好服务的日志路径（如Spark 的 /spark-history、YARN 的 /tmp/$user/ 等）。
- sticky bit

  针对文件夹可设置sticky bit，可以防止除了superuser/file owner/dir owner之外的其它用户删除该文件夹中的文件/文件夹（即使其它用户对该文件夹有rwx 权限）。

```
#即在第一位添加数字1
hadoop fs -chmod 1777 /tmp
hadoop fs -chmod 1777 /spark-history
hadoop fs -chmod 1777 /user/hive/warehouse
```

6.2.16.4. 开发指南

本文为您介绍如何通过Java API接口方式进行HDFS的相关操作。

背景信息

HDFS相关的操作如下所示：

- HDFS初始化
- 创建目录
- 写文件
- 追加文件内容
- 读文件
- 删除目录

- 删除文件
- 移动或重命名文件
- 移动或重命名目录

HDFS初始化

在使用HDFS提供的API之前，需要先进行HDFS初始化操作。初始化HDFS时会加载HDFS的配置文件，HDFS使用到的配置文件主要为*core-site.xml*和*hdfs-site.xml*两个文件。

初始化代码样例如下。

```
private void init() throws IOException {
  conf = new Configuration();
  // conf path for core-site.xml and hdfs-site.xml
  conf.addResource(new Path(PATH_TO_HDFS_SITE_XML));
  conf.addResource(new Path(PATH_TO_CORE_SITE_XML));
  fSystem = FileSystem.get(conf);
}
```

在HDFS文件系统初始化之后，就可以调用HDFS提供的各种API进行开发。

创建目录

如果要在HDFS文件系统中创建目录，需要FileSystem实例的exists方法判断该目录是否已经存在：

- 如果存在，则直接返回。
- 如果不存在，则调用FileSystem实例的mkdirs方法创建该目录。

创建目录代码样例如下。

```
/**
 * create directory path
 *
 * @param dirPath
 * @return
 * @throws java.io.IOException
 */
private boolean createPath(final Path dirPath) throws IOException {
    if (!fSystem.exists(dirPath)) {
        fSystem.mkdirs(dirPath);
    }
    return true;
}
```

写文件

通过调用FileSystem实例的create方法获取写文件的输出流。通常获得输出流之后，可以直接对这个输出流进行写入操作，将内容写入HDFS的指定文件中。写完文件后，需要调用close方法关闭输出流。

写文件代码样例如下。

```
/**
 * 创建文件，写文件
 *
 * @throws java.io.IOException
 */
private void createAndWrite() throws IOException {
   final String content = "Hello HDFS!";
   FSDataOutputStream out = null;
   try {
       out = fSystem.create(new Path(DEST_PATH + File.separator + FILE_NAME));
       out.write(content.getBytes());
       out.hsync();
       LOG.info("success to write.");
   } finally {
    // make sure the stream is closed finally.
       out.close();
  }
}
```

追加文件内容

对于已经在HDFS中存在的文件，可以追加指定的内容，以增量的形式在该文件现有内容的后面追加。通过调用FileSystem实例的append方法获取追加写入的输出流。然后使用该输出流将待追加内容添加到HDFS的指定文件后面。追加完指定的内容后，需要调用close方法关闭输出流。

> **注意** 需要确保待追加的文件已经存在，并且没有正在被写入内容，否则追加内容会失败抛出异常。

追加文件内容代码样例如下。

```
/**
 * 追加文件内容
 *
 * @throws java.io.IOException
 */
private void appendContents() throws IOException {
    final String content = "Hello Hello";
    FSDataOutputStream out = null;
    try {
        out = fSystem.append(new Path(DEST_PATH + File.separator + FILE_NAME));
        out.write(content.getBytes());
        out.hsync();
        LOG.info("success to append.");
    } finally {
        // make sure the stream is closed finally.
        out.close();
    }
}
```

读文件

读文件即为获取HDFS上某个指定文件的内容。通过调用FileSystem实例的open方法获取读取文件的输入流。然后使用该输入流读取HDFS的指定文件的内容。读完文件后，需要调用close方法关闭输入流。

读文件代码样例如下。

```
private void read() throws IOException {
    String strPath = DEST_PATH + File.separator + FILE_NAME;
    Path path = new Path(strPath);
    FSDataInputStream in = null;
    BufferedReader reader = null;
    StringBuffer strBuffer = new StringBuffer();
    try {
        in = fSystem.open(path);
        reader = new BufferedReader(new InputStreamReader(in));
        String sTempOneLine;
        // write file
        while ((sTempOneLine = reader.readLine()) != null) {
            strBuffer.append(sTempOneLine);
        }
        LOG.info("result is : " + strBuffer.toString());
        LOG.info("success to read.");
    } finally {
        // make sure the streams are closed finally.
        IOUtils.closeStream(reader);
        IOUtils.closeStream(in);
    }
}
```

删除目录

通过调用delete方法删除HDFS上某个指定目录。delete方法第二个参数代表是否递归删除目录下面的所有目录。如果该参数为false，而目录下还存在文件或者子目录，则删除目录操作会失败。

> **注意** 待删除的目录会被直接删除，且无法恢复，因此，请谨慎使用删除目录操作。

删除目录代码样例如下。

```
private boolean deletePath(final Path dirPath) throws IOException {
    if (!fSystem.exists(dirPath)) {
        return false;
    }
    // fSystem.delete(dirPath, true);
    return fSystem.delete(dirPath, true);
}
```

删除文件

通过调用delete方法删除HDFS上某个指定文件。

> 注意 待删除的文件会被直接删除，且无法恢复，因此，请谨慎使用删除文件操作。

删除文件代码样例如下。

```
private void deleteFile() throws IOException {
    Path beDeletedPath = new Path(DEST_PATH + File.separator + FILE_NAME);
    if (fSystem.delete(beDeletedPath, true)) {
        LOG.info("success to delete the file " + DEST_PATH + File.separator + FILE_NAME);
    } else {
        LOG.warn("failed to delete the file " + DEST_PATH + File.separator + FILE_NAME);
    }
}
```

移动或重命名文件

对于HDFS来说，文件的重命名和移动是一个操作。调用FileSystem的rename方法对HDFS文件系统的文件进行重命名操作。

> 注意 rename方法的原文件必须存在，并且目标文件不能存在，否则重命名操作会失败。

移动或重命名文件代码样例如下。

```
private void renameFile() throws IOException {
    Path srcFilePath = new Path(SRC_PATH + File.separator + SRC_FILE_NAME);
    Path destFilePath = new Path(DEST_PATH + File.separator + DEST_FILE_NAME);
        fs.rename(new Path(srcFilePath), new Path(destFilePath));
}
```

移动或重命名目录

对于HDFS来说，目录的重命名和移动是一个操作。调用FileSystem的rename方法对HDFS文件系统的目录进行重命名操作。

> 注意 rename方法的原目录必须存在，并且目标目录不能存在，否则重命名操作会失败。

移动或重命名目录代码样例如下。

```
private void renameDir() throws IOException {
    Path srcDirPath = new Path(SRC_PATH + File.separator + SRC_DIR_NAME);
    Path destDirPath = new Path(DEST_PATH + File.separator + DEST_DIR_NAME);
        fs.rename(new Path(srcDirPath), new Path(destDirPath));
}
```

6.2.16.5. 最佳实践

6.2.16.5.1. JVM内存调优

本文为您介绍如何调整NameNode JVM和DataNode JVM内存大小，以便优化HDFS的稳定性。

前提条件

已创建集群，详情请参见创建集群。

调整NameNode JVM内存大小

- 背景：在HDFS中，每个文件对象都需要在NameNode中记录元数据信息，并占用一定的内存空间。默认的JVM配置可以满足部分普通的HDFS使用。部分Workload会向HDFS中写入更多地文件， 或者随着一段时间的积累HDFS保存的文件数不断增加，当增加的文件数超过默认的内存空间配置时，则默认的内存空间无法存储相应的信息，您需要修改内存大小的设置。

- 建议：修改NameNode JVM内存大小。修改方式如下：
  - HA集群

    您可以在EMR控制台的HDFS服务的**配置**页面，在搜索区域，搜索参数**hadoop_namenode_heapsize**，参数值根据实际需求进行调整。
  - 非HA集群

    您可以在EMR控制台的HDFS服务的**配置**页面，在搜索区域，搜索参数**hadoop_namenode_heapsize**和**hadoop_secondary_namenode_heapsize**，参数值根据实际需求进行调整。

    > **说明** 配置完成后，需要重启相应的NameNode或SecondaryNamenode服务，使得配置生效。

  您可以通过访问HDFS UI页面，查看文件数Files和文件块数Blocks，访问Web UI的详情信息，请参见访问链接与端口。您可以根据以下计算方法调整NameNode JVM内存大小。

  ```
  建议值=（文件数（以百万为单位）+块数（以百万为单位））×512 MB
  ```

  例如，您有1000万个文件，都是中小文件且不超过1个Block，Blocks数量也为1000万，则内存大小建议值为10240 MB，即（10+10）×512 MB。

  当您的大多数文件不超过1个Block时，建议值如下。

文件数量	建议值（MB）
10,000,000	10240
20,000,000	20480
50,000,000	51200
100,000,000	102400

调整DataNode JVM内存大小

- 背景：在HDFS中，每个文件Block对象都需要在DataNode中记录Block元数据信息，并占用一定的内存空间。默认的JVM配置可以满足部分简单的、压力不大的作业需求。而部分作业会向HDFS写入更多的文件，或者随着一段时间的积累HDFS保存的文件数不断增加。当文件数增加，DataNode上的Block数也会增加，而默认的内存空间无法存储相应的信息时，则需要修改内存大小的设置。
- 建议：您可以在EMR控制台的HDFS服务的**配置**页面，在搜索区域，搜索参数hadoop_datanode_heapsize，参数值根据实际需求进行调整，以修改DataNode JVM内存大小。

  您可以根据以下计算方法调整DataNode JVM内存大小。

  ```
  集群中每个DataNode实例平均保存的副本数Replicas=文件块数Blocks×3÷DataNode节点数
  建议值=单个DataNode副本数Replicas（百万单位）×2048 MB
  ```

  > **说明** 配置完成后，需重启DataNode服务，使配置生效。

  例如，大数据机型为3副本，Core节点数量为6，如果您有1000万个文件且都是中小文件，Blocks数量也为1000万，则单个DataNode副本数Replicas为 500万（1000万×3÷6），内存大小建议值为10240 MB（5×2048 MB）。

  当您的大多数文件不超过1个Block时，建议值如下。

单个DataNode实例平均Replicas数量	建议值（MB）
1,000,000	2048
2,000,000	4096
5,000,000	10240

> **说明** 该建议值已为JVM内核预留了部分空间，以及为作业高峰压力预留了部分空间，因此通常情况下可以直接换算使用。

6.2.16.5.2. 实时计算场景优化

本文为您介绍在E-MapReduce（简称EMR）上使用HDFS进行实时计算场景化配置的一些建议，以便优化HDFS的稳定性。

前提条件

已创建集群，详情请参见创建集群。

调整DataNode Xceiver连接数

- 背景：通常实时计算框架会打开较多的HDFS文件写入流（Stream），方便不断地向HDFS写入新的数据。HDFS允许同时打开的文件数量是有限的，受限于DataNode参数**dfs.datanode.max.transfer.threads**。
- 建议：您可以在EMR控制台HDFS服务的**配置**页面，在**配置搜索**区域，搜索参数**dfs.datanode.max.transfer.threads**，该参数表示DataNode处理读或写流的线程池大小，默认值为4096。当您在日志目录下或者客户端运行日志中发现如下报错时，可以适当地调大该参数值：
  - 在日志目录*/var/log/hadoop-hdfs/*下观察DataNode服务端日志，发现如下报错。

    ```
    java.io.IOException: Xceiver count 4097 exceeds the limit of concurrent xcievers: 4096
            at org.apache.hadoop.hdfs.server.datanode.DataXceiverServer.run(DataXceiverServer.java:150)
    ```

  - 在客户端运行日志中发现如下报错。

    ```
    DataXceiver error processing WRITE_BLOCK operation  src: /10.*.*.*:35692 dst: /10.*.*.*:50010
    java.io.IOException: Premature EOF from inputStream
    ```

配置预留磁盘空间

- 背景：HDFS对于打开的文件写入流，会预先保留128 MB Blocksize的磁盘剩余空间，从而确保该文件可以正常写入。如果该文件实际大小很小，例如仅为8 MB，则当文件调用close方法关闭输入流时只会占用8 MB的磁盘空间。

  通常实时计算框架会打开较多的HDFS文件写入流，如果同时打开很多文件， 则HDFS会预先保留较多的磁盘空间。如果磁盘剩余空间不够，则会导致创建文件失败。
- 建议：如果同时打开的文件数为N，则集群至少需要预留的磁盘空间为`N * 128 MB * 副本数`。

6.2.16.5.3. HDFS使用优化

本文为您介绍在E-MapReduce（简称EMR）上使用HDFS进行场景化配置的一些建议，以便优化HDFS的使用性能或稳定性等。

背景信息

本文为您介绍一些HDFS使用的优化建议：

- 配置回收站机制
- 控制小文件个数
- 配置HDFS单目录文件数量
- 在网络不稳定的情况下，降低客户端运行异常概率
- 配置可容忍的磁盘坏卷
- 防止目录被误删
- 使用Balancer进行容量均衡

配置回收站机制

- 背景：在HDFS中，删除的文件将被移动到回收站（trash）中，以便在误操作的情况下恢复被删除的数据。

  您可以设置文件保留在回收站中的时间阈值，一旦文件保存时间超过此阈值，系统将自动从回收站中永久地删除该文件。您也可以手动删除回收站里面的文件。
- 建议：您可以在EMR控制台HDFS服务的**配置**页面，在**配置搜索**区域，搜索参数fs.trash.interval，参数描述如下表。

参数	描述	默认值
fs.trash.interval	以分钟为单位的垃圾回收时间，垃圾站中数据超过此时间会被删除。 如果设置为0，表示禁用回收站机制。	1440

> **说明** 建议您使用默认值，以便于在误操作的情况下恢复被删除的文件。不建议您将此参数设置过大，避免回收站中过多的文件占用集群的可用空间。

控制小文件个数

- 背景：HDFS NameNode将所有文件元数据加载在内存中，在集群磁盘容量一定的情况下，如果小文件个数过多，则会造成NameNode的内存容量瓶颈。
- 建议：尽量控制小文件的个数。对于存量的小文件，建议合并为大文件。

配置HDFS单目录文件数量

- 背景：当集群运行时，不同组件（例如Spark和YARN）或客户端可能会向同一个HDFS目录不断写入文件。但HDFS系统支持的单目录文件数目是有上限的，因此需要您提前做好规划，防止单个目录下的文件数目超过阈值，导致任务出错。
- 建议：您可以在EMR控制台HDFS服务的**配置**页面，单击**hdfs-site**页签，然后单击**自定义配置**，新增参数dfs.namenode.fs-limits.max-directory-items，以设置单个目录下可以存储的文件数目，最后保存配置。添加参数详情，请参见添加组件参数。

> **说明** 您需要将数据做好存储规划，可以按时间、业务类型等分类，不要单个目录下直属的文件过多，建议单个目录下约100万条。

在网络不稳定的情况下，降低客户端运行异常概率

- 背景：在网络不稳定的情况下，调整ipc.client.connect.max.retries.on.timeouts和ipc.client.connect.timeout参数，适当提高客户端的重试次数和超时时间，可以降低客户端运行异常的概率。
- 建议：您可以在EMR控制台HDFS服务的**配置**页面，在**配置搜索**区域，搜索参数ipc.client.connect.max.retries.on.timeouts，您可以增大该参数值，增加连接的最大重试次数。在**配置搜索**区域，搜索参数ipc.client.connect.timeout，您可以增大该参数值，增加建立连接的超时时间。

参数	描述	默认值
ipc.client.connect.max.retries.on.timeouts	客户端同服务端建立Socket连接时，客户端的最大重试次数。	45
ipc.client.connect.timeout	客户端与服务端建立Socket连接的超时时间。 单位：毫秒。	20000

配置可容忍的磁盘坏卷

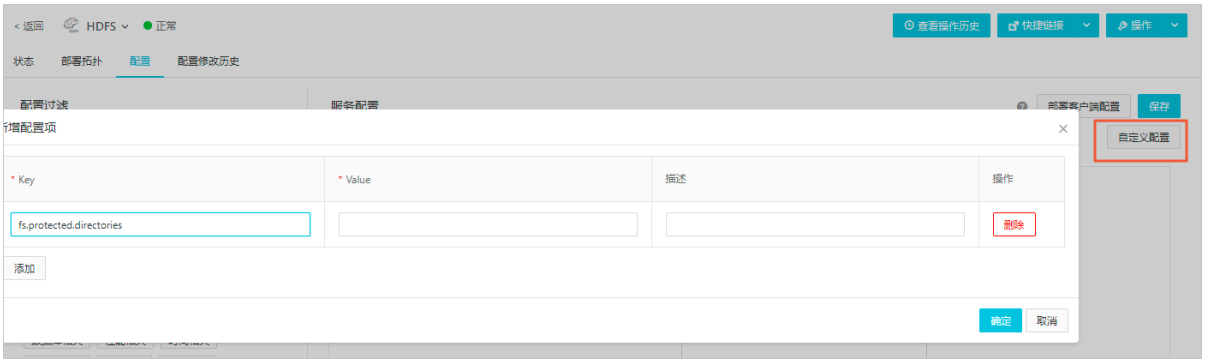
- 背景：如果为DataNode配置多个数据存放卷，默认情况下其中一个卷损坏，则DataNode将不再提供服务。
- 建议：您可以在EMR控制台HDFS服务的**配置**页面，在**配置搜索**区域，搜索参数dfs.datanode.failed.volumes.tolerated，您可以修改此参数，指定失败的个数，小于该个数，DataNode可以继续提供服务。

参数	描述	默认值
dfs.datanode.failed.volumes.tolerated	DataNode停止提供服务前允许失败的卷数。默认情况下，必须至少有一个有效卷。	0

> **说明** 当DataNode存在坏盘，而又没有其他足够的节点可以提供服务时，可以临时将该值调大，先将DataNode启动起来。

防止目录被误删

- 背景：HDFS允许将一些目录配置为受保护的，避免这些目录被误删除，但是依然可以将目录挪到回收站。
- 建议：您可以在EMR控制台HDFS服务的**配置**页面，单击**core-site**页签，然后单击**自定义配置**，新增参数**fs.protected.directories**，参数值为您待保护的目录，多个目录时使用逗号（,）分隔，并保存配置。



使用Balancer进行容量均衡

- 背景：HDFS集群可能出现DataNode节点间磁盘利用率不平衡的情况，例如集群中添加新DataNode的场景。如果HDFS出现数据不平衡的状况，则可能导致个别DataNode压力过大。

- 建议：您可以使用Balancer操作进行容量均衡。

  > **说明** 执行Balancer操作时会占用DataNode的网络带宽资源，请根据业务需求在业务空闲时期执行Balancer任务。

  i. 登录待配置集群任意节点。

  本示例登录Master节点，详情请参见登录集群。

  ii. （可选）执行以下命令，修改Balancer的最大带宽。

  ```
  hdfs dfsadmin -setBalancerBandwidth <bandwidth in bytes per second>
  ```

  > **说明** 代码示例中的 `<bandwidth in bytes per second>` 为设置的最大带宽，例如，如果需要设置带宽控制为20 MB/s，对应值为20971520，则完整代码示例为 `hdfs dfsadmin -setBalancerBandwidth 20971520` 。如果集群负载较高，可以改为209715200（200 MB/s）；如果集群空闲，可以改为1073741824（1 GB/s）。

  iii. 执行以下命令，切换到hdfs用户并执行Balancer参数。

  ```
  su hdfs
  /usr/lib/hadoop-current/sbin/start-balancer.sh -threshold 10
  ```

  iv. 执行以下命令，进入*hadoop-hdfs*目录。

  ```
  cd /var/log/hadoop-hdfs
  ```

  v. 执行 `ll` 命令，查看Balancer日志。

  返回信息类似如下截图。

  ```
  [hdfs@emr-header-1 hadoop-hdfs]$ ll
  total 266440
  -rw-r---r-- 1 hdfs hadoop      2519 Jun 30 10:53 hadoop-hdfs-balancer-emr-header-1.cluster-23[redacted].log
  -rw-r---r-- 1 hdfs hadoop       975 Jun 30 10:53 hadoop-hdfs-balancer-emr-header-1.cluster-23[redacted].out
  -rw-r---r-- 1 hdfs hadoop  21824254 Jun 30 10:53 hadoop-hdfs-namenode-emr-header-1.cluster-23[redacted].log
  -rw-r---r-- 1 hdfs hadoop     10723 Jun 16 17:17 hadoop-hdfs-namenode-emr-header-1.cluster-23[redacted].out
  -rw-r---r-- 1 hdfs hadoop     10723 Jun  8 13:14 hadoop-hdfs-namenode-emr-header-1.cluster-23[redacted].out.1
  ```

  vi. 执行以下命令，查看Balancer运行情况。

  ```
  tailf /var/log/hadoop-hdfs/hadoop-hdfs-balancer-emr-header-xx.cluster-xxx.log
  ```

  当提示信息包含 `Successfully` 字样时，表示执行成功。

  > **说明** 代码中的 `hadoop-hdfs-balancer-emr-header-xx.cluster-xxx.log` 为前面步骤中获取到的日志名称。

6.2.16.6. 常见问题

本文汇总了HDFS使用时的常见问题。

- 为什么NameNode重启特别慢？
- 为什么NameNode无法响应？
- 为什么会有大量的Editslog文件？
- 为什么有大量的Under Replicated Blocks？
- 如何处理Missing Blocks或Corrupted Blocks问题？
- 如何处理EditsLog不连续导致NameNode启动失败的问题？

为什么NameNode重启特别慢？

- 问题现象：NameNode原先正常，重启NameNode过程中非常慢，并且NameNode重启未完成，十几分钟后自动重启了。观察日志时发现正在加载FsImage和EditsLog。
- 问题原因：因为NameNode启动过程中加载FsImage和EditsLog会消耗较多的内存。
- 解决方法：建议调大NameNode HeapSize，详情请参见调整NameNode JVM内存大小。

为什么NameNode无法响应？

- 问题现象：NameNode节点长时间满负载，所在节点CPU达到100%，NameNode无法响应。
- 问题原因：因为NameNode的内存容量已经无法承担太多的文件，进程在频繁发生FULL GC。
- 解决方法：建议调大NameNode HeapSize，详情请参见调整NameNode JVM内存大小。

为什么会有大量的Editslog文件？

- 问题现象：NameNode节点数据目录占用磁盘空间大，发现有大量的Editslog文件。

- 问题原因：查看Secondary NameNode（非HA集群）或Standby NameNode（HA集群）的健康状态，发现Secondary NameNode或Standby NameNode服务不正常，导致了Editslog文件没有及时合并。服务不正常很可能是内存不够导致的。
- 解决方法：适当调节NameNode的HeapSize，使其正常启动，详情请参见调整NameNode JVM内存大小。

为什么有大量的Under Replicated Blocks?

- 问题现象：使用 `fsck` 命令查看，发现有大量Under Replicated Blocks。
- 问题原因：由于Decommission或节点（磁盘）异常下线后，副本数恢复较慢。
- 解决方法：需要恢复副本数，您可以在EMR控制台的HDFS服务的**配置**页面，在搜索区域，搜索下表参数并调大参数值。

参数	描述
dfs.namenode.replication.work.multiplier.per.iteration	默认值100。建议调大为200，但不超过500。 该参数影响NameNode下发给每个DataNode进行副本复制作业任务的并发度，即任务调度速度。 该参数是系数值，实际下发任务数为该系数值乘以集群节点个数。
dfs.namenode.replication.max-streams	建议设置为100。 该参数负责调节低优先级的块的复制任务的执行并发度。
dfs.namenode.replication.max-streams-hard-limit	默认值100。建议调大为200，但不超过500。 该参数负责调节所有优先级的块的复制任务的执行并发度，包含最高优先级的块。

如何处理Missing Blocks或Corrupted Blocks问题?

- 问题现象：使用 `fsck` 命令查看，提示Missing Blocks或Corrupted Blocks。
- 问题原因：可能是DataNode停止了服务，或者是磁盘损坏或异常操作导致数据丢失。
- 解决方法：如果之前DataNode停止了服务，请将DataNode重新启动下。 如果是磁盘损坏或异常操作导致数据丢失，需要人工恢复，您可以通过 `hdfs fsck / -files` 命令扫描损坏的文件，导出文件列表，删除后重新上传。

如何处理EditsLog不连续导致NameNode启动失败的问题?

- 问题现象：在JournalNode节点断电，数据目录磁盘占满，网络异常时，重启NameNode失败。
- 问题原因：可能是JournalNode上的EditsLog不连续。
- 解决方法：某台NameNode EditsLog损坏的情况下，需要手工恢复。

  操作方法如下：

  i. 备份NameNode节点元数据的整个目录*/mnt/disk1/hdfs*，以防误操作的风险。
  ii. 观察NameNode启动日志，记录加载失败的EditsLog的txid。
  iii. 登录另外一台NameNode节点，找到并复制相同txid的EditsLog文件，覆盖本节点的同名文件。
  iv. 重启NameNode，观察是否成功。

  > **说明** 如果还是失败，请提交工单。

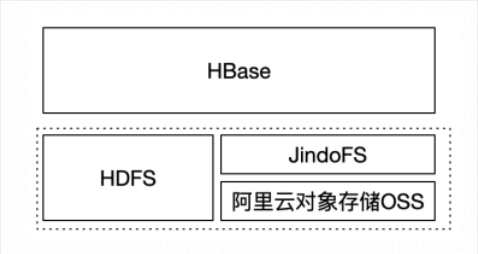
6.2.17. HBase

6.2.17.1. HBase概述

HBase是一个高可靠性、高性能、面向列和可伸缩的分布式存储系统。本文为您介绍如何在E-MapReduce中使用HBase。

背景信息

E-MapReduce中HBase架构如下所示。



E-MapReduce（简称EMR）中的HBase也可以构建在JindoFS（OSS）上。

HBase特点如下：

- 处理海量数据（TB或PB级别以上）。
- 具有高吞吐量。
- 在海量数据中实现高效的随机读取。
- 具有很好的伸缩能力。

EMR中HBase的主要组件如下：

- HMaster：部署EMR的Master节点。

  > **说明** 集群开启高可用时会启动两个HMaster。

- RegionServer：部署EMR的Worker节点。

关于Apache HBase的更多介绍，请参见Apache HBase官网。

使用HBase

- 访问HBase详情，请参见访问HBase。
- HBase on JindoFS详情，请参见使用JindoFS作为HBase的底层存储。

6.2.17.2. HBase授权

HBase在不开启授权的情况下，任何账号对HBase集群可以进行任何操作，例如disable table、drop table、major compact等。

背景信息

对于没有Kerberos认证的集群，即使开启了HBase授权，用户也可以伪造身份访问集群服务。所以建议创建高安全模式（即支持 Kerberos）的集群，详情请参见概述。

进入配置页面

1.
2. 在顶部菜单栏处，根据实际情况选择地域和资源组。
3. 单击上方的**集群管理**页签。
4. 在**集群管理**页面，单击相应集群所在行的**详情**。
5. 在左侧导航栏单击**集群服务 > HBase**。
6. 单击**配置**页签。

添加配置

1. 在**服务配置**区域，单击**hbase-site**。
2. 单击右侧的**自定义配置**，设置以下配置。

Key	Value
hbase.security.authorization	true
hbase.coprocessor.master.classes	org.apache.hadoop.hbase.security.access.AccessController
hbase.coprocessor.region.classes	org.apache.hadoop.hbase.security.token.TokenProvider,org.apache.hadoop.hbase.security.access.AccessController
hbase.coprocessor.regionserver.classes	org.apache.hadoop.hbase.security.access.AccessController,org.apache.hadoop.hbase.security.token.TokenProvider

重启HBase集群

1. 在**集群服务 > HBase**页面，单击右上角的**操作 > 重启 All Components**。
2. 执行集群操作。
   i. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
   ii. 在**确认**对话框，单击**确定**。
   iii. 单击右上角**查看操作历史**查看任务进度，等待任务完成。

授权（ACL）

- 基本概念

  授权就是将对 [某个范围的资源] 的 [操作权限] 授予[某个实体]。

  在 HBase 中，上述对应的三个概念分别为：

  - 某个范围（Scope）的资源

名称	描述
Superuser	超级账号可以进行任何操作，运行HBase服务的账号默认是 Superuser。也可以通过在*hbase-site.xml*中配置hbase.superuser的值可以添加超级账号。
Global	Global Scope拥有集群所有table的Admin权限。
Namespace	在Namespace Scope进行相关权限控制。
Table	在Table Scope进行相关权限控制。
ColumnFamily	在ColumnFamily Scope进行相关权限控制。
Cell	在Cell Scope进行相关权限控制。

  - 操作权限

名称	描述
Read（R）	读取某个Scope资源的数据。
Write （W）	写数据到某个Scope的资源。
Execute （X）	在某个Scope执行协处理器。
Create （C）	在某个Scope创建或删除表等操作。
Admin（A）	在某个Scope进行集群相关操作，如balance、assign等。

  - 某个实体

名称	描述
User	对某个用户授权。
Group	对某个用户组授权。

- 授权命令
  - grant授权

```
grant <user> <permissions> [<@namespace> [<table> [<column family> [<column qualifier>]]]
```

    - user和group的授权方式一样，但group需要加一个前缀@

```
grant 'test','R','tbl1'   #给用户test授予表tbl1的读权限。
grant '@testgrp','R','tbl1' #给用户组testgrp授予表tbl1的读权限。
```

    - namespace需要加一个前缀@

```
grant 'test','C','@ns_1'  #给用户test授予namespace ns_1的CREATE权限。
```

- revoke回收

```
revoke 'trafodion'  #回收trafodion用户的所有权限。
```

- user_permission查看权限

```
user_permission 'TABLE_A'  #查看TABLE_A表的所有权限。
```

6.2.18. Hudi

6.2.18.1. Hudi概述

Apache Hudi是一种数据湖的存储格式，在Hadoop文件系统之上提供了更新数据和删除数据的能力以及消费变化数据的能力。

Hudi表类型

Hudi支持如下两种表类型：

- Copy On Write

  使用Parquet格式存储数据。Copy On Write表的更新操作需要通过重写实现。

- Merge On Read

  使用列式文件格式（Parquet）和行式文件格式（Avro）混合的方式来存储数据。Merge On Read使用列式格式存放Base数据，同时使用行式格式存放增量数据。最新写入的增量数据存放至行式文件中，根据可配置的策略执行COMPACTION操作合并增量数据至列式文件中。

针对不同表类型的差异点如下表所示。

Trade-off	Copy On Write	Merge On Read
Data Latency	高	低
Query Latency	低	高
Update cost (I/O)	高（重写整个Parquet）	Lower（追加到Delta Log）
Parquet File Size	小 （高更新(I/O)开销）	大（低更新开销）
Write Amplification	高（写放大较高）	低（取决于合并策略）

Hudi查询类型

Hudi支持如下三种查询类型：

- Snapshot Queries

  可以查询最新COMMIT的快照数据。针对Merge On Read类型的表，查询时需要在线合并列存中的Base数据和日志中的实时数据；针对Copy On Write表，可以查询最新版本的Parquet数据。

  Copy On Write和Merge On Read表支持该类型的查询。

- Incremental Queries

  支持增量查询的能力，可以查询给定COMMIT之后的最新数据。

  Copy On Write和Merge On Read表支持该类型的查询。

- Read Optimized Queries

  只能查询到给定COMMIT之前所限定范围的最新数据。Read Optimized Queries是对Merge On Read表类型快照查询的优化，通过牺牲查询数据的时效性，来减少在线合并日志数据产生的查询延迟。

针对不同查询类型的差异点如下表所示。

Trade-off	Snapshort Queries	Read Optimized Queries
Data Latency	低	高
Query Latency	对于MOR类型，高	低

应用场景

- 近实时数据摄取

  Hudi支持插入、更新和删除数据的能力。您可以实时摄取消息队列（Kafka）和日志服务SLS等日志数据至Hudi中，同时也支持实时同步数据库Binlog产生的变更数据。

  Hudi优化了数据写入过程中产生的小文件。因此，相比其他传统的文件格式，Hudi对HDFS文件系统更加的友好。

- 近实时数据分析

  Hudi支持多种数据分析引擎，包括Hive、Spark、Presto和Impala。Hudi作为一种文件格式，不需要依赖额外的服务进程，在使用上也更加的轻量化。

- 增量数据处理

  Hudi支持Incremental Query查询类型，您可以通过Spark Streaming查询给定COMMIT后发生变更的数据。Hudi提供了一种消费HDFS变化数据的能力，可以用来优化现有的系统架构。

6.2.18.2. 基础使用

本文为您介绍如何在E-MapReduce Hudi中写数据以及查询数据。

写数据

EMR-3.32.0以及后续版本中，已经将Hudi相关依赖集成到各个开源组件中，包括Spark、Hive和Presto，因此运行时不需要引入额外的Hudi依赖，只需要在pom文件中添加Hudi依赖即可。不同的EMR版本使用的Hudi版本不同，详细信息请参见下表。

Hudi版本	EMR版本
0.6	EMR 3.32~EMR 3.35，EMR 4.5~EMR 4.9
0.8	EMR 3.36~EMR 3.37，EMR 5.2
0.9	EMR 3.38，EMR 5.4
0.10	EMR 3.39，EMR 4.10，EMR 5.5.0

```
<dependency>
   <groupId>org.apache.hudi</groupId>
   <artifactId>hudi-spark_2.11</artifactId>
   <!-- for spark3 <artifactId>hudi-spark_2.12</artifactId> -->
   <version>${hudi_version}</version>
  <scope>provided</scope>
</dependency>
```

示例如下。

```
 val spark = SparkSession
      .builder()
      .master("local[*]")
      .appName("hudi test")
      .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
      .getOrCreate()
import spark.implicits._
    val df = (for (i <- 0 until 10) yield (i, s"a$i", 30 + i * 0.2, 100 * i + 10000, s"p${i % 5}"))
      .toDF("id", "name", "price", "version", "dt")
    df.write.format("hudi")
      .option(TABLE_NAME, "hudi_test_0")
      // .option(OPERATION_OPT_KEY, UPSERT_OPERATION_OPT_VAL) for update
      .option(OPERATION_OPT_KEY, INSERT_OPERATION_OPT_VAL) // for insert
      .option(RECORDKEY_FIELD_OPT_KEY, "id")
      .option(PRECOMBINE_FIELD_OPT_KEY, "version")
      .option(KEYGENERATOR_CLASS_OPT_KEY, classOf[SimpleKeyGenerator].getName)
      .option(HIVE_PARTITION_EXTRACTOR_CLASS_OPT_KEY, classOf[MultiPartKeysValueExtractor].getCanonicalName)
      .option(PARTITIONPATH_FIELD_OPT_KEY, "dt")
      .option(HIVE_PARTITION_FIELDS_OPT_KEY, "ds")
      .option(META_SYNC_ENABLED_OPT_KEY, "true")
      .option(HIVE_USE_JDBC_OPT_KEY, "false")
      .option(HIVE_DATABASE_OPT_KEY, "default")
      .option(HIVE_TABLE_OPT_KEY, "hudi_test_0")
      .option(INSERT_PARALLELISM, "8")
      .option(UPSERT_PARALLELISM, "8")
      .mode(Overwrite)
      .save("/tmp/hudi/h0")
```

示例如下。

```
df.write.format("hudi")
      .option(TABLE_NAME, "hudi_test_0")
      .option(OPERATION_OPT_KEY, DELETE_OPERATION_OPT_VAL) // for delete
      .option(RECORDKEY_FIELD_OPT_KEY, "id")
      .option(PRECOMBINE_FIELD_OPT_KEY, "version")
      .option(KEYGENERATOR_CLASS_OPT_KEY, classOf[SimpleKeyGenerator].getName)
      .option(HIVE_PARTITION_EXTRACTOR_CLASS_OPT_KEY, classOf[MultiPartKeysValueExtractor].getCanonicalName)
      .option(PARTITIONPATH_FIELD_OPT_KEY, "dt")
      .option(DELETE_PARALLELISM, "8")
      .mode(Append)
      .save("/tmp/hudi/h0")
```

环境配置

Insert和Update

Delete

查询数据

EMR引擎环境中已集成Hudi相关的软件包，您无需在Spark、Presto和Hive查询引擎中额外引入相关依赖。

Hive和Presto查询Hudi表，需要在写入阶段开启元数据同步功能，即设置META_SYNC_ENABLED_OPT_KEY为true。

对于社区版Hudi， COW和MOR表需要设置hive.input.format为org.apache.hudi.hadoop.hive.HoodieCombineHiveInputFormat。EMR版本对于COW类型表，可以不用设置input format，支持自动适配Hudi的input format功能。

6.2.18.3. 高阶使用

6.2.18.3.1. Hudi与Spark SQL集成

E-MapReduce的Hudi 0.8.0版本支持Spark SQL对Hudi进行读写操作，可以极大的简化Hudi的使用成本。本文为您介绍如何通过Spark SQL对Hudi进行读写操作。

前提条件

已创建Hadoop集群，详情请参见创建集群。

使用限制

EMR-3.36.0及后续版本和EMR-5.2.0及后续版本，支持Spark SQL对Hudi进行读写操作。

进入spark-sql命令行

1. 使用SSH方式登录到集群，详情信息请参见登录集群。
2. 执行以下命令，进入*spark-sql*命令行。

   ```
   spark-sql --conf 'spark.serializer=org.apache.spark.serializer.KryoSerializer' \
   --conf 'spark.sql.extensions=org.apache.spark.sql.hudi.HoodieSparkSessionExtension'
   ```

   当返回信息中包含如下信息时，表示已进入spark-sql命令行。

   ```
   spark-sql>
   ```

SQL操作

- 新建表

  ```
  create table h0(
     id bigint,
     name string,
     price double
   ) using hudi
   options (
     primaryKey = 'id',
     preCombineField = 'id'
  );
  ```

  返回信息中包含如下类似信息，表示创建表格成功。

  ```
  Time taken: 1.258 seconds
  ```

- 插入数据

```
insert into h0 select 1, 'a1', 10;
```

返回信息中包含如下类似信息，表示插入数据成功。

```
Time taken: 7.294 seconds
```

- 查询数据

```
select id,name,price from h0;
```

返回信息中包含如下类似信息，表示查询数据成功。

```
1  a1  10.0
Time taken: 1.219 seconds, Fetched 1 row(s)
```

6.2.18.4. 开发指南

6.2.18.4.1. DDL语句

本文为您介绍Hudi与Spark SQl集成后，支持的建表语句。

背景信息

Spark SQL创建Hudi表时，可以通过options设置表配置信息，options参数如下表所示。

> 注意 0.10版本之后options被替换为tblproperties。

参数	描述	是否必选
primaryKey	指定主键列，多个主键时使用逗号（,）隔开。	必选
type	表类型，支持以下两种类型： • cow（默认值）：表示Copy-On-Write类型表。 • mor：表示Merge-On-Read类型表。	可选
preCombineField	版本字段。 对应Hudi的DataSourceWriteOptions.PRECOMBINE_FIELD_OPT_KEY字段。	建议设置，否则upsert场景无法支持
payloadClass	默认值为DefaultHoodieRecordPayload。 对应Hudi的DataSourceWriteOptions.PAYLOAD_CLASS_OPT_KEY字段。	可选

前提条件

已创建Hadoop集群，详情请参见创建集群。

使用限制

EMR-3.36.0及后续版本和EMR-5.2.0及后续版本，支持Spark SQL对Hudi进行读写操作。

进入spark-sql命令行

1. 使用SSH方式登录到集群，详情信息请参见登录集群。
2. 执行以下命令，进入*spark-sql*命令行。

```
spark-sql --conf 'spark.serializer=org.apache.spark.serializer.KryoSerializer' \
--conf 'spark.sql.extensions=org.apache.spark.sql.hudi.HoodieSparkSessionExtension'
```

当返回信息中包含如下信息时，表示已进入spark-sql命令行。

```
spark-sql>
```

创建非分区表

options通过primaryKey指定主键列，多个字段时使用逗号（,）隔开。创建非分区表的示例如下所示：

- 创建表类型为cow，主键为id的非分区表。

```
create table if not exists h0(
  id bigint,
  name string,
  price double
) using hudi
options (
  type = 'cow',
  primaryKey = 'id'
);
```

- 创建表类型为mor，主键为id和name的非分区表。

```
create table if not exists h0(
  id bigint,
  name string,
  price double
) using hudi
options (
  type = 'mor',
  primaryKey = 'id,name'
);
```

- 创建表类型为cow的非分区表。

```
create table if not exists h0(
  id bigint,
  name string,
  price double
) using hudi
options (
  type = 'cow'
);
```

创建分区表

创建分区表的示例如下所示。

```
create table if not exists h_p0 (
id bigint,
name string,
dt string,
hh string
) using hudi
location 'oss://xxx/h_p0'
options (
  type = 'cow',
  primaryKey = 'id',
  preCombineField = 'id'
 )
partitioned by (dt, hh);
```

> **说明** 本文代码示例中的location为表所在的路径，可以是OSS路径，也可以是HDFS路径。主键为id，分区字段为dt和hh，版本字段为ts。

创建外表

支持在已经存在的Hudi表之上创建外表。创建外表示例如下所示。

```
create table h0
using hudi
location '/xx/xx/h0';
```

CTAS语法

通过以下示例为您介绍如何使用CTAS语法。

- 示例1：

```
create table if not exists h1 using hudi
as
select 1 as id, 'a1' as name, 10 as price;
```

- 示例2:

```
create table if not exists h2 using hudi
partitioned by (dt)
location '/xx/xx/h2'
options (
  type = 'mor',
  primaryKey = 'id,name'
) as
select 1 as id, 'a1' as name, 20 as price, '2021-01-03' as dt;
```

6.2.18.4.2. DML语句

本文为您介绍Hudi与Spark SQl集成后，支持的DML语句。

前提条件

已创建Hadoop集群，详情请参见创建集群。

使用限制

EMR-3.36.0及后续版本和EMR-5.2.0及后续版本，支持Spark SQL对Hudi进行读写操作。

进入spark-sql命令行

1. 使用SSH方式登录到集群，详情信息请参见登录集群。
2. 执行以下命令，进入*spark-sql*命令行。

   ```
   spark-sql --conf 'spark.serializer=org.apache.spark.serializer.KryoSerializer' \
   --conf 'spark.sql.extensions=org.apache.spark.sql.hudi.HoodieSparkSessionExtension'
   ```

   当返回信息中包含如下信息时，表示已进入spark-sql命令行。

   ```
   spark-sql>
   ```

MERGE INTO

表示执行插入、更新或删除操作。

- 语法

```
MERGE INTO tableIdentifier AS target_alias
USING (sub_query | tableIdentifier) AS source_alias
ON <merge_condition>
 WHEN MATCHED [ AND <condition> ] THEN <matched_action>
[ WHEN MATCHED [ AND <condition> ] THEN <matched_action> ]
[ WHEN NOT MATCHED [ AND <condition> ]  THEN <not_matched_action> ]
<merge_condition> =A equal bool condition
<matched_action>  =
  DELETE  |
  UPDATE SET *  |
  UPDATE SET column1 = value1 [, column2 = value2 ...]
<not_matched_action>  =
  INSERT *  |
  INSERT (column1 [, column2 ...]) VALUES (value1 [, value2 ...])
```

- 示例

```
-- without delete
merge into h0 as target
using (
  select 1 as id, 'a1' as name, 10.0 as price
) source
on target.id = source.id
when matched then update set *
when not matched then insert *;
-- with delete
merge into h0 as target
using (
  select 1 as id, 'a1' as name, 10.0 as price
) source
on target.id = source.id
when matched then update set id = source.id, name = source.name, price = source.price
when matched and name = 'delete' then delete
when not matched then insert (id,name,price) values(id, name, price);
```

INSERT INTO

表示向分区表或非分区表插入数据。

代码示例如下所示：

- 向非分区表h0中插入数据。

```
insert into h0 select 1, 'a1', 20;
```

- 向静态分区表h_p0中插入数据。

```
insert into h_p0 partition(dt='2021-01-02') select 1, 'a1';
```

- 向动态分区表h_p0中插入数据。

```
insert into h_p0 partition(dt) select 1, 'a1', dt from s;
```

- 向动态分区（分区字段放在select表达式最后面）h_p1表中插入数据。

```
insert into h_p1 select 1 as id, 'a1', '2021-01-03' as dt, '19' as hh;
```

- 重写表h0中的数据。

```
insert overwrite table h0 select 1, 'a1', 20;
```

UPDATE语句

表示更新分区表或非分区表中行对应的单列或多列数据。

- 语法

```
UPDATE tableIdentifier SET column = EXPRESSION(,column = EXPRESSION);
```

- 示例：将表h0中id为1的price字段值更新为20。

```
update h0 set price=20 where id=1;
```

DELETE语句

表示删除分区表或非分区表中满足指定条件的单行或多行数据。

- 语法

```
DELETE FROM tableIdentifier [WHERE BOOL_EXPRESSION];
```

- 示例：删除表h0中id大于100的数据。

```
delete from h0 where id>100;
```

6.2.18.4.3. 设置Hudi参数

本文通过两种方式为您介绍，如何设置Hudi参数。

Set方式

此方式通过Set设置全局参数。

```
set hoodie.insert.shuffle.parallelism = 100;
set hoodie.upsert.shuffle.parallelism = 100;
set hoodie.delete.shuffle.parallelism = 100;
```

options方式

此方式是在建表语句options中指定参数来设置全局参数。

```
create table if not exists h0(
  id bigint,
  name string,
  price double
) using hudi
options (
  primaryKey = 'id',
  type = 'mor',
  hoodie.index.type = 'GLOBAL_BLOOM'
);
```

6.2.18.5. 常见问题

本文汇总了Hudi使用时的常见问题。

- Spark查询Hudi数据重复，如何处理？
- Hive查询Hudi数据重复，如何处理？
- Spark查询Hudi表分区裁剪不生效？

Spark查询Hudi数据重复，如何处理？

- 问题原因：出现Spark查询hudi数据重复，通常是因为Hudi不支持Spark DataSource方式读取导致的。
- 解决方法：您需要在执行查询Hudi表的命令时，添加上 `spark.sql.hive.convertMetastoreParquet=false` 。

Hive查询Hudi数据重复，如何处理？

- 问题原因：Hive默认使用HiveCombineInputFormat不会调用表自定义的 `input format` 。
- 解决方法：您需要在执行查询Hudi表的命令时，添加上 `set hive.input.format = org.apache.hudi.hadoop.hive.HoodieCombineHiveInputFormat` 。

Spark查询Hudi表分区裁剪不生效？

- 问题原因：可能是在分区字段包含/（正斜线）的情况下，分区字段个数和实际分区目录级数不一致，导致Spark分区裁剪失效。
- 解决方法：您在使用Spark DataFrame API写Hudi表时，需要加上 `hoodie.datasource.write.partitionpath.urlencode= true` 。

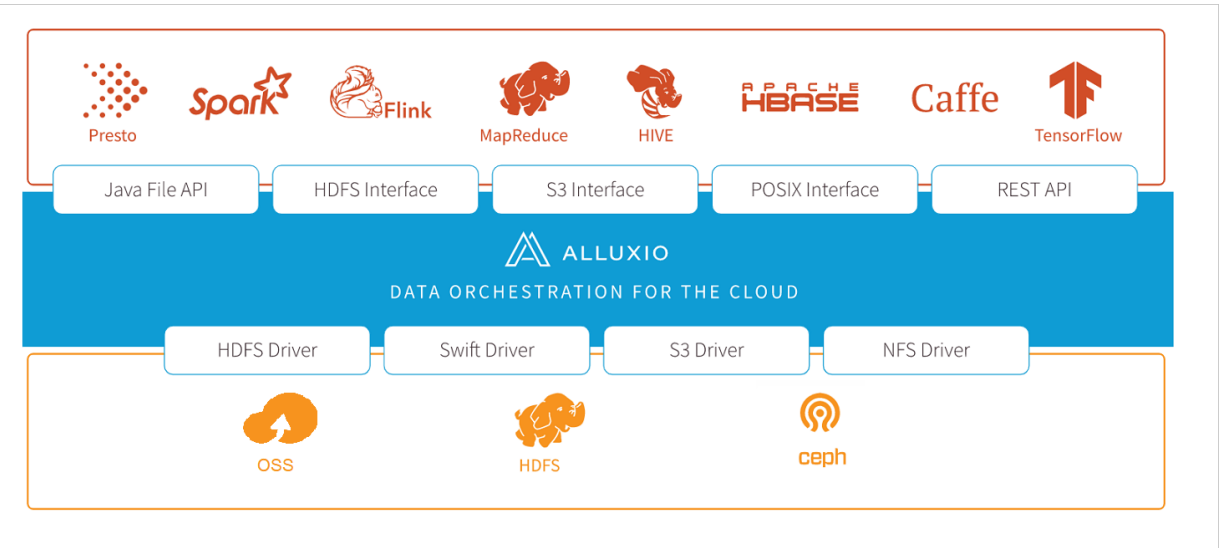
6.2.19. Alluxio

6.2.19.1. 概述

Alluxio是一个面向基于云的数据分析和人工智能的开源的数据编排技术。Alluxio为数据驱动型应用和存储系统构建了桥梁，将数据从存储层移动到距离数据驱动型应用更近的位置，从而能够更容易被访问， 同时使得应用程序能够通过一个公共接口连接到许多存储系统。

背景信息

在大数据生态系统中，Alluxio位于数据驱动框架或应用（例如Apache Spark、Presto、TensorFlow、Apache Flink和Apache Hive等）和各种持久化存储系统（例如HDFS和阿里云OSS）之间，使得上层的计算应用可以通过统一的客户端API和全局命名空间访问包括HDFS和OSS在内的持久化存储系统。



优势

- 提供内存级I/O吞吐率，同时降低具有弹性扩张特性的数据驱动型应用的成本开销。
- 简化云存储和对象存储接入。
- 简化数据管理，提供对多数据源的单点访问。
- 应用程序部署简易。

Alluxio的详细信息，请参见Alluxio。

6.2.19.2. 基础使用

6.2.19.2.1. 常见命令

您可以在已经创建好的E-MapReduce（简称EMR）集群中，直接使用Alluxio Shell命令来对Alluxio中的文件进行操作，也可以使用Hadoop Shell命令操作Alluxio中的文件。本文为您介绍Alluxio的常见命令。

前提条件

- 已创建集群，并选择了Alluxio服务，详情请参见创建集群。
- 已登录集群，详情请参见登录集群。

背景信息

Alluxio常见命令如下表所示。

命令	功能
mkdir	在Alluxio文件系统中创建目录。
cat	查看Alluxio上文件的内容。
ls	在文件或者目录创建完之后，您可以查看指定路径下的文件或目录信息。在查看文件或目录信息的时候需要给出绝对路径。
mv	移动文件或目录到目标路径。
copyFromLocal	上传本地文件到Alluxio的指定路径。
copyToLocal	下载Alluxio指定路径的文件到本地路径。

命令	功能
rm	删除Alluxio系统中指定的文件。

关于Alluxio的更多命令介绍，请参见Alluxio。

mkdir

在Alluxio文件系统中创建目录。

- 语法
  - Alluxio Shell用法：

    ```
    alluxio fs mkdir <path1> [path2] ... [pathn]
    ```

  - Hadoop Shell用法：

    ```
    hadoop dfs -mkdir alluxio://<path1> [path2] ... [pathn]
    ```

- 示例：
  - 在Alluxio文件系统中，创建*/dir*目录。

    ```
    alluxio fs mkdir /dir
    ```

    返回如下信息表示创建目录成功。

    ```
    Successfully created directory /dir
    ```

  - 在*dir*目录下，创建*logs*目录。

    ```
    alluxio fs mkdir /dir/logs
    ```

    返回如下信息表示创建目录成功。

    ```
    Successfully created directory /dir/logs
    ```

ls

在文件或者目录创建完之后，您可以查看指定路径下的文件或目录信息。在查看文件或目录信息的时候需要给出绝对路径。

- 语法
  - Alluxio Shell用法：

    ```
    alluxio fs ls <path>
    ```

  - Hadoop Shell用法：

    ```
    hadoop dfs -ls alluxio://<path>
    ```

- 示例：
  - 查看*/tmp*目录下*hello.txt*文件的信息。

    ```
    alluxio fs ls /tmp/hello.txt
    ```

    ```
    [root@emr-header-1 ~]# alluxio fs ls /tmp/hello.txt
                8       PERSISTED 06-01-2021 14:41:27:793 100% /tmp/hello.txt
    [root@emr-header-1 ~]# hadoop dfs -ls alluxio:///tmp/hello.txt
    DEPRECATED: Use of this script to execute hdfs command is deprecated.
    Instead use the hdfs command for it.

    21/06/01 15:07:49 INFO hadoop.AbstractFileSystem: Creating Alluxio configuration from Hadoop configuration {}, uri configuration {}
    21/06/01 15:07:49 INFO hadoop.AbstractFileSystem: Initializing filesystem with connect details emr-header-1:19998
    21/06/01 15:07:49 INFO network.TieredIdentityFactory: Initialized tiered identity TieredIdentity(node=emr-header-1.cluster-230659, rack=null)
    21/06/01 15:07:49 INFO network.NettyUtils: EPOLL_MODE is available
    -rw-r--r--   0 root root          8 2021-06-01 14:41 alluxio:///tmp/hello.txt
    [root@emr-header-1 ~]#
    ```

- 查看*/dir/logs*目录的信息。

```
alluxio fs ls /dir/logs
```

```
[root@emr-header-1 ~]# alluxio fs ls /dir/logs/
            13       PERSISTED 06-01-2021 15:14:02:744 100% /dir/logs/hello_world.txt
[root@emr-header-1 ~]# hadoop dfs -ls alluxio:///dir/logs
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

21/06/01 15:30:30 INFO hadoop.AbstractFileSystem: Creating Alluxio configuration from Hadoop configuration {}, uri configuration {}
21/06/01 15:30:30 INFO hadoop.AbstractFileSystem: Initializing filesystem with connect details emr-header-1:19998
21/06/01 15:30:30 INFO network.TieredIdentityFactory: Initialized tiered identity TieredIdentity(node=emr-header-1.cluster-230659, rack=null)
21/06/01 15:30:30 INFO network.NettyUtils: EPOLL_MODE is available
Found 1 items
-rw-r--r--   0 root root         13 2021-06-01 15:14 alluxio:///dir/logs/hello_world.txt
[root@emr-header-1 ~]#
```

cat

查看Alluxio上文件的内容。

- 语法
  - Alluxio Shell用法：

```
alluxio fs cat <path>
```

  - Hadoop Shell用法：

```
hadoop dfs -cat alluxio://<path>
```

- 示例：查看*/tmp3*目录下*hello.txt*文件的内容。

```
alluxio fs cat hello.txt
```

返回*hello.txt*文件的内容。

```
[root@emr-header-1 ~]# alluxio fs cat /tmp3/hello.txt
welcome
Hello
[root@emr-header-1 ~]# hadoop dfs -cat alluxio:///tmp3/hello.txt
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

21/06/01 18:52:36 INFO hadoop.AbstractFileSystem: Creating Alluxio configuration from Hadoop configuration {}, uri configuration {}
21/06/01 18:52:37 INFO hadoop.AbstractFileSystem: Initializing filesystem with connect details emr-header-1:19998
21/06/01 18:52:37 INFO network.TieredIdentityFactory: Initialized tiered identity TieredIdentity(node=emr-header-1.cluster-230659, rack=null)
21/06/01 18:52:37 INFO network.NettyUtils: EPOLL_MODE is available
welcome
Hello
[root@emr-header-1 ~]#
```

mv

移动文件或目录到目标路径。

- 语法
  - Alluxio Shell用法：

```
alluxio fs mv <path>
```

  - Hadoop Shell用法：

```
hadoop dfs -mv alluxio://<path>
```

- 示例
  - 移动*/tmp/*目录下的文件*hello.txt*至*/tmp3/logs/*目录下。

```
alluxio fs mv /tmp/hello.txt /tmp3/logs/hello.txt
```

您可以通过ls命令查看文件移动的情况。

```
[root@emr-header-1 ~]# hadoop fs -mv /tmp/hello_world2.txt /dir/sub-dir/
[root@emr-header-1 ~]#  hadoop fs -ls /dir/sub-dir/
Found 2 items
-rw-r-----   2 root hadoop         13 2021-05-28 14:48 /dir/sub-dir/hello_world.txt
-rw-r-----   2 root hadoop         13 2021-05-28 15:04 /dir/sub-dir/hello_world2.txt
[root@emr-header-1 ~]#
```

- 移动*/tmp/*路径下的*test*目录至*/dir/sub-dir/*目录下。

  ```
  hadoop fs -mv /tmp/test /dir/sub-dir/
  ```

  您可以通过ls命令查看目录移动的情况。

  ```
  [root@emr-header-1 ~]# hadoop fs -mv /tmp/test /dir/sub-dir/
  [root@emr-header-1 ~]#  hadoop fs -ls /tmp/
  Found 4 items
  drwxrwxrwx   - root      hadoop          0 2021-05-24 12:18 /tmp/hadoop-yarn
  -rw-r-----   2 root      hadoop         13 2021-05-28 14:56 /tmp/hello_world.txt
  drwx-wx-wx   - anonymous hadoop          0 2021-05-28 15:50 /tmp/hive
  drwxrwxrwt   - hadoop    hadoop          0 2021-05-24 12:20 /tmp/logs
  [root@emr-header-1 ~]#  hadoop fs -ls /dir/sub-dir/
  Found 4 items
  -rw-r-----   2 root   hadoop         13 2021-05-28 14:48 /dir/sub-dir/hello_world.txt
  -rw-r-----   2 root   hadoop         13 2021-05-28 15:04 /dir/sub-dir/hello_world2.txt
  drwx-wx-wx   - hadoop hadoop          0 2021-05-24 12:24 /dir/sub-dir/hive
  drwxr-x--x   - root   hadoop          0 2021-05-28 15:54 /dir/sub-dir/test
  [root@emr-header-1 ~]#
  ```

copyFromLocal

> 注意 上传本地文件到Alluxio的指定路径时，Alluxio Shell使用的是 `copyFromLocal` 命令，Hadoop Shell使用的是 `put` 命令。

上传本地文件到Alluxio的指定路径。

- 语法
  - Alluxio Shell用法：

    ```
    alluxio fs copyFromLocal <src> <remoteDst>
    ```

  - Hadoop Shell用法：

    ```
    hadoop dfs -put <src> alluxio://<remoteDst>
    ```

- 示例：上传本地文件*hello_world.txt*至Alluxio的*/dir/logs2*路径下。

  ```
  alluxio fs copyFromLocal hello_world.txt /dir/logs2
  ```

  返回如下信息表示上传成功。

  ```
  Copied file:///root/hello_world.txt to /dir/logs2
  ```

copyToLocal

> 注意 下载Alluxio指定路径的文件到本地路径时，Alluxio Shell使用的是 `copyToLocal` 命令，Hadoop Shell使用的是 `get` 命令。

下载Alluxio指定路径的文件到本地路径。

- 语法
  - Alluxio Shell用法：

    ```
    alluxio fs copyToLocal <src> <localDst>
    ```

  - Hadoop Shell用法：

    ```
    hadoop dfs -get <src> alluxio://<localDst>
    ```

- 示例：下载Alluxio系统中*/dir/sub-dir/*目录下的文件*hello_world2.txt*至本地的*/emr*路径下。

  ```
  alluxio fs copyToLocal /dir/sub-dir/hello_world2.txt /emr
  ```

  返回如下信息表示下载成功。

  ```
  Copied /dir/sub-dir/hello_world2.txt to file:///emr
  ```

rm

删除Alluxio系统中指定的文件。

- 语法
  - Alluxio Shell用法：

    ```
    alluxio fs rm <path>
    ```

- Hadoop Shell用法：

```
hadoop dfs -rm alluxio://<path>
```

- 示例：删除Alluxio系统中*/dir/logs2*目录下的文件*hello_world.txt*。

```
alluxio fs rm /dir/logs2/hello_world.txt
```

返回如下信息表示删除成功。

```
/dir/logs2/hello_world.txt has been removed
```

6.2.19.3. 高阶使用

6.2.19.3.1. 管理员常见命令

您可以在已经创建好的E-MapReduce（简称EMR）集群中，使用Alluxio Admin Shell命令来对Alluxio中的文件进行管理和运维。本文为您介绍Alluxio管理员的常见命令。

前提条件

- 已创建集群，并选择了Alluxio服务，详情请参见创建集群。
- 已登录集群，详情请参见登录集群。

背景信息

管理员的常见命令如下表所示。

命令	功能
backup	创建Alluxio元数据的备份。
doctor	显示Alluxio错误和警告。
report	显示Alluxio运行中的集群信息。
ufs	更新挂载的底层存储的属性。

关于管理员的更多命令介绍，请参见管理员命令行接口。

backup

创建Alluxio元数据的备份。

- 语法

```
alluxio fsadmin backup [directory]
```

- 示例：
  - 备份元数据至默认目录。

```
alluxio fsadmin backup
```

返回如下信息表示备份成功。

```
Backup Host        : emr-header-1.cluster-23****
Backup URI         : hdfs://emr-header-1:9000/alluxio_backups/alluxio-backup-2021-06-01-1622547059762.gz
Backup Entry Count : 25
```

  - 备份元数据至*/tmp*目录。

```
alluxio fsadmin backup /tmp
```

返回如下信息表示备份成功。

```
Backup Host        : emr-header-1.cluster-23****
Backup URI         : hdfs://emr-header-1:9000/tmp/alluxio-backup-2021-06-01-1622547072114.gz
Backup Entry Count : 25
```

doctor

显示Alluxio错误和警告。

- 语法

```
alluxio fsadmin doctor [category]
```

> **说明** `[category]` 为可选参数，不传入任何参数，则打印出所有类别的错误和警告。`[category]` 可以为configuration或storage参数。

- 示例：显示Alluxio错误和警告。

```
alluxio fsadmin doctor
```

返回如下信息，表示没有错误和告警信息。

```
All worker storage paths are in working state.
```

report

显示Alluxio运行中的集群信息。

- 语法

```
alluxio fsadmin report [category]
```

> **说明** `[category]` 为可选参数，不传入任何参数，则只打印摘要信息。`[category]` 可以为capacity、metrics、summary、ufs或jobservice参数。

- 示例：显示Alluxio运行中的集群信息。

```
alluxio fsadmin report
```

返回集群信息类似如下所示。

```
Alluxio cluster summary:
    Master Address: emr-header-1:1****
    Web Port: 1****
    Rpc Port: 1****
    Started: 06-01-2021 14:07:08:420
    Uptime: 0 day(s), 20 hour(s), 42 minute(s), and 57 second(s)
    Version: 2.5.0
    Safe Mode: false
    Zookeeper Enabled: false
    Live Workers: 2
    Lost Workers: 0
    Total Capacity: 88.00GB
        Tier: MEM  Size: 8.00GB
        Tier: SSD  Size: 80.00GB
    Used Capacity: 320.00MB
        Tier: MEM  Size: 64.00MB
        Tier: SSD  Size: 256.00MB
    Free Capacity: 87.69GB
```

ufs

更新挂载的底层存储的属性。

- 语法

```
alluxio fsadmin --mode <noAccess/readOnly/readWrite> <ufsPath>
```

- 示例：设置底层存储为readOnly模式来禁止写入操作。

```
alluxio fsadmin ufs --mode readOnly hdfs://ns
```

6.2.19.3.2. 管理缓存

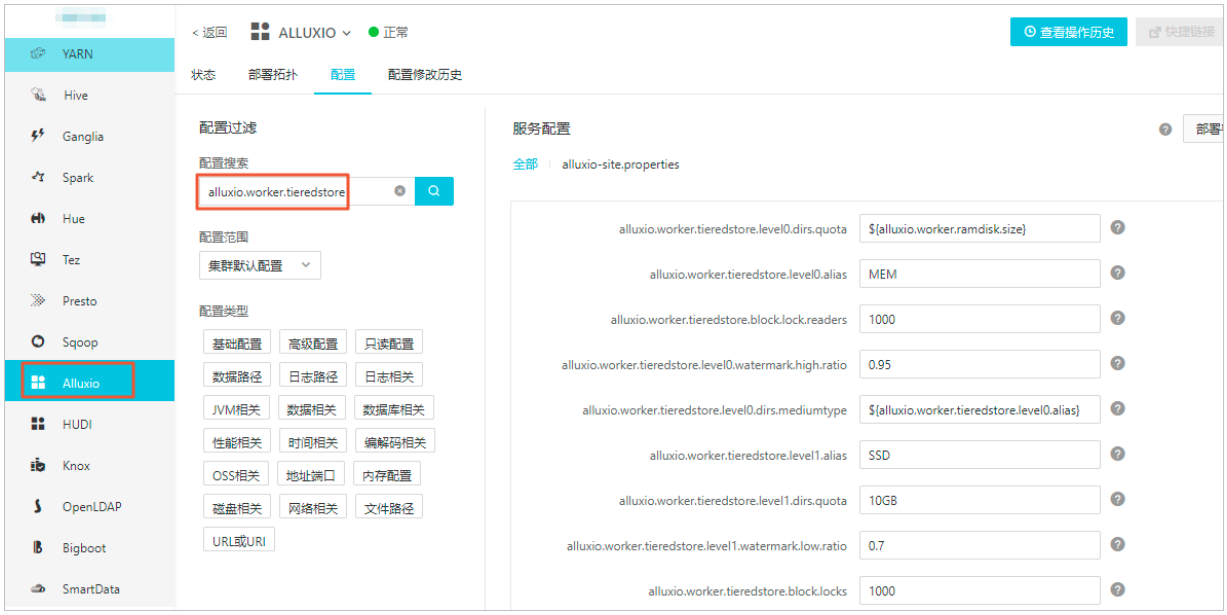
Alluxio利用E-MapReduce（简称EMR）集群的本地节点的内存和磁盘对数据进行分布式缓存。本文为您介绍缓存相关的内容。

前提条件

- 已创建集群，并选择了Alluxio服务，详情请参见创建集群。
- 已登录集群，详情请参见登录集群。

背景信息

EMR默认使用双层缓存存储，即同时使用内存和磁盘进行缓存，内存默认分配了当前节点的10%，磁盘默认分配当前节点的30%。如果您需要修改当前的分配情况，可以在Alluxio服务的配置页面，搜索以**alluxio.worker.tieredstore**开头的配置项并修改。



缓存详细信息，请参见缓存。

缓存策略

客户端写新的数据块时，默认情况下会将其写入level0层存储。如果level0没有足够的可用空间， 则会尝试下一层存储。如果在所有层上均未找到存储空间，Alluxio会释放空间来存储新写入的数据块。默认的释放策略是LRUAnnotator，按照最近最少使用的顺序释放数据块。

客户端读取数据块时，如果数据已经存在于Alluxio中，则客户端将直接读取对应Worker节点；如果Alluxio中不存在数据，则会先缓存数据至Alluxio中，以便下次从Worker节点读取。

管理数据生命周期

数据生命周期管理常见命令如下表所示。

命令	功能
free	释放缓存中的数据。
load	加载数据到Alluxio缓存中。
persist	将Alluxio中的文件或目录持久化到底层文件系统中。
setTtl	设置文件或目录的生存时间（TTL），单位为毫秒。

free

释放缓存中的数据。

释放数据是指从Alluxio缓存中删除数据，而不是从底层UFS中删除数据。释放操作后，数据仍然可供用户使用，但对Alluxio释放文件后尝试访问该文件的客户端来讲性能可能会降低。

- 语法

```
alluxio fs free <path>
```

- 示例：将*tmp*目录中的所有数据从缓存中释放。

```
alluxio fs free /tmp
```

返回如下信息。

```
/tmp was successfully freed from Alluxio space.
```

load

加载数据到Alluxio缓存中。

- 语法

```
alluxio fs load <path>
```

- 示例：加载*/tmp3/logs*目录中的所有数据到Alluxio缓存中。

```
alluxio fs load /tmp3/logs
```

返回如下信息。

```
/tmp3/logs loaded
```

persist

将Alluxio中的文件或目录持久化到底层文件系统中。

持久化数据是指将Alluxio存储中可能被修改过或未被修改过的数据写回UFS。 通过将数据写回到UFS，可以保证如果Alluxio发生故障数据还是可恢复的。

- 语法

```
alluxio fs persist <path>
```

- 示例：将Alluxio中*tmp*目录持久化到底层文件系统中。

```
alluxio fs persist /tmp
```

返回如下信息。

```
persisted file /tmp with size 46
```

setTtl

设置文件或目录的生存时间（TTL），单位为毫秒。

如果当前时间大于该文件的创建时间与TTL时间之和时，行动参数将指示要执行的操作。delete操作（默认）将同时删除Alluxio和底层文件系统中的文件，而free操作仅仅删除Alluxio中的文件。

- 语法

```
alluxio fs setTtl [--action delete|free] <path> <time to live>
```

- 示例：
  - 一分钟后，*tmp*目录将被删除。

    ```
    alluxio fs setTtl /tmp 60000
    ```

    返回如下信息。

    ```
    TTL of path '/tmp' was successfully set to 60000 milliseconds, with expiry action set to DELETE
    ```

  - 一天后，*dir*目录缓存将被驱逐。

    ```
    alluxio fs setTtl --action free /dir 86400000
    ```

    返回如下信息。

    ```
    TTL of path '/dir' was successfully set to 86400000 milliseconds, with expiry action set to FREE
    ```

6.2.19.3.3. 设置权限

本文为您介绍E-MapReduce（简称EMR）中Alluxio服务权限相关的内容，包括认证（Authentication），授权（Authorization）和审计（Audit），并介绍如何开启授权和审计。

前提条件

已创建集群，并选择了Alluxio服务，详情请参见创建集群。

认证

认证用于确认访问者的身份信息。

EMR中的Alluxio服务支持SIMPLE、NOSASL和CUSTOM三种认证方式。默认为SIMPLE认证方式，以便用于日志审计。

在SIMPLE认证方式下，客户端访问Alluxio服务时，会从操作系统获取当前的登录用户，一起发送请求到服务端，供服务端进行身份标识。如果客户端设置了**alluxio.security.login.username**参数，客户端将使用对应的配置作为请求服务端的用户。如果是创建目录或文件请求，该用户信息会保存在元数据中。Alluxio支持修改认证方式，修改后需要重启服务生效，详细信息请参见Alluxio文档。

开启授权

授权是验证用户是否有权限对文件和目录进行操作。Alluxio为POSIX权限模型，Alluxio服务会根据认证中提供的用户信息判断允许或拒绝用户的访问请求。

EMR中的Alluxio默认不开启授权，您可以通过以下方式开启授权。

1. 进入Alluxio页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务 > Alluxio**。
2. 在Alluxio服务页面，单击**配置**页签。
3. 开启授权。
   i. 在配置搜索中，输入配置项**alluxio.security.authorization.permission.enabled**，单击 图标。
   ii. 设置参数**alluxio.security.authorization.permission.enabled**的值为**true**。

   iii. 单击**保存**。
   iv. 在**确认修改**页面，输入**执行原因**，单击**确定**。
4. 重启服务。
   i. 单击右上角的**操作 > 重启All Components**。
   ii. 在**执行集群操作**页面，输入**执行原因**，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

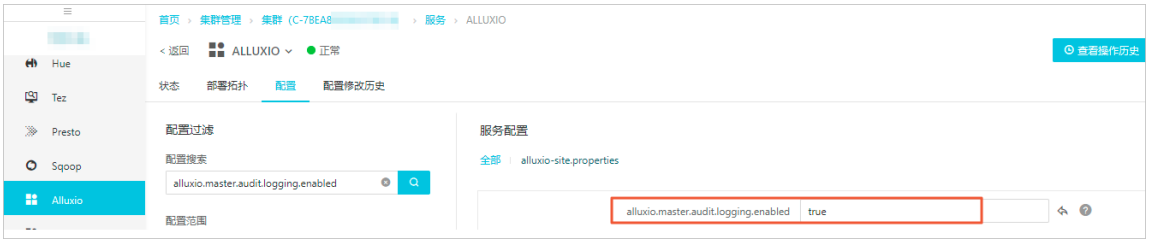
开启审计

Alluxio服务支持通过审计日志，查看和跟踪用户对文件元数据的访问操作。Alluxio服务的审计日志存储在*/mnt/disk1/log/alluxio/master_audit.log*中。

EMR中的Alluxio默认不开启审计功能，您可以通过以下方式开启审计功能。

1. 进入Alluxio页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务 > Alluxio**。
2. 在Alluxio服务页面，单击**配置**页签。
3. 开启审计功能。
   i. 在配置搜索中，输入配置项**alluxio.master.audit.logging.enabled**，单击 图标。

ii. 设置参数alluxio.master.audit.logging.enabled的值为true。



iii. 单击**保存**。

iv. 在**确认修改**页面，输入**执行原因**，单击**确定**。

4. 重启服务。

i. 单击右上角的**操作 > 重启All Components**。

ii. 在**执行集群操作**页面，输入**执行原因**，单击**确定**。

iii. 在**确认**对话框中，单击**确定**。

6.2.19.4. 常见问题

本文汇总了Alluxio使用时的常见问题。

- 错误提示：No FileSystem for scheme: alluxio
- 如何排查Alluxio服务状态异常的问题？
- 如何对OSS进行缓存加速？
- 如何配置Alluxio参数？

错误提示：No FileSystem for scheme: alluxio

如果您是在创建E-MapReduce集群时选择的Alluxio服务，是不会出现此类问题的。通常情况下，此类问题是由于您在E-MapReduce集群创建后，通过添加服务的方式添加的Alluxio服务，添加服务后没有重启服务。

添加完Alluxio服务后，您需要重启服务来加载Hadoop配置。重启服务的具体操作，请参见重启服务。

如何排查Alluxio服务状态异常的问题？

您需要报错信息先查找Alluxio服务异常的节点，找到异常信息，然后查看Alluxio服务的异常日志来确认问题。

通常情况下，Alluxio服务的日志位于*/mnt/disk1/log/alluxio/*目录下。

如何对OSS进行缓存加速？

E-MapReduce默认使用HDFS作为Alluxio的UFS，对于OSS目录建议作为挂载点来使用。

您可以使用如下代码，对OSS上的数据进行缓存加速。命令示例如下所示。

```
alluxio fs mount --option fs.oss.accessKeyId=<OSS_ACCESS_KEY_ID> \
  --option fs.oss.accessKeySecret=<OSS_ACCESS_KEY_SECRET> \
  --option fs.oss.endpoint=<OSS_ENDPOINT> \
  /oss_dir <path>/
```

代码示例中的参数描述如下：

- `<OSS_ACCESS_KEY_ID>`：阿里云账号的AccessKey ID。
- `<OSS_ACCESS_KEY_SECRET>`：阿里云账号的AccessKey Secret。
- `<OSS_ENDPOINT>`：OSS的地域。您可以在OSS的控制台查看，通常为*oss-xxxx-internal.aliyuncs.com*形式。EMR集群需要和OSS在同一Region下，建议使用内网Endpoint，例如*oss-cn-shanghai-internal.aliyuncs.com*。
- `<path>`：OSS上文件的路径。例如，`oss://<OSS_YOURBUCKETNAME>/<OSS_DIRECTORY>`，`<OSS_YOURBUCKETNAME>`为您OSS Bucket的名称。

如何配置Alluxio参数？

您可以通过配置全局参数和动态传入配置两种方式修改Alluxio参数。

- 配置全局参数

  您可以在E-MapReduce控制台的Alluxio服务页面，修改配置项，详细信息请参见管理组件参数。

- 动态传入配置

- Alluxio Shell

  您可以登录集群，通过 `-Dproperty=value` 格式的命令，添加自定义配置，代码示例如下所示。

  ```
  alluxio fs copyFromLocal hello.txt /dir/tmp -Dalluxio.user.file.writetype.default=CACHE_THROUGH
  ```

  > **说明** 代码中的 `hello.txt` 为您本地的文件， `/dir/tmp` 为Alluxio文件系统中的目录。 `copyFromLocal` 命令的用法，请参见常见命令。

- Spark作业

  您可以通过对Spark executor的spark.executor.extraJavaOptions和Spark drivers的spark.driver.extraJavaOptions添加 `-Dproperty=value` 格式的命令，向Spark作业传递JVM环境参数。

  例如，提交Spark作业时，将Alluxio写类型设置为CACHE_THROUGH，代码片段示例如下所示。

  ```
  spark-submit \
  --conf 'spark.driver.extraJavaOptions=-Dalluxio.user.file.writetype.default=CACHE_THROUGH' \
  --conf 'spark.executor.extraJavaOptions=-Dalluxio.user.file.writetype.default=CACHE_THROUGH' \
  ```

- MapReduce作业

  您可以在 `hadoop jar` 或 `yarn jar` 命令后添加 `-Dproperty=value` 格式的命令，向MapReduce作业传递属性。

  例如，在MapReduce作业中，将Alluxio写类型设置为CACHE_THROUGH，代码片段示例如下所示。

  ```
  hadoop jar <HADOOP_HOME>/share/hadoop/mapreduce/hadoop-mapreduce-examples-x.x.x.jar wordcount \
  -Dalluxio.user.file.writetype.default=CACHE_THROUGH \
  -libjars /<PATH_TO_ALLUXIO>/client/alluxio-x.x.x.-client.jar \
  <path1> <path2>
  ```

  > **说明** 代码示例中的 `<path1>` 为输入文件的路径， `<path2>` 为输出文件的目录， `x.x.x` 为JAR包的具体版本号。 `<HADOOP_HOME>/share/hadoop/mapreduce/hadoop-mapreduce-examples-x.x.x.jar` 和 `<PATH_TO_ALLUXIO>/client/alluxio-x.x.x.-client.jar` 均为示例。

6.2.20. Kudu

6.2.20.1. 概述

Kudu是一个分布式的，具有可扩展性的列式存储管理器，可以对快速变化的数据进行快速分析。

使用场景

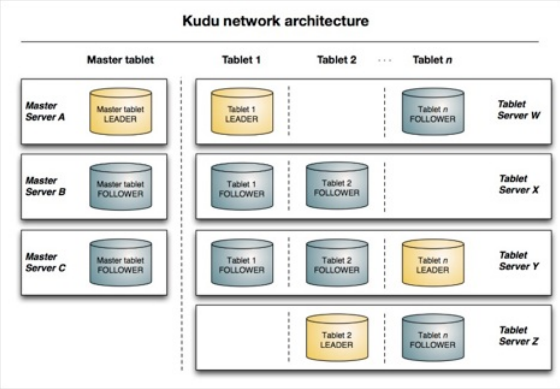
典型的应用场景如下：

- 近实时计算场景
- 时间序列数据的场景
- 预测建模
- 与存量数据共存

  通常生产环境中会有大量的存量数据，数据可能存储在HDFS、RDBMS或Kudu中。如果您只是想访问和查询这些存量数据，可以使用Impala访问和查询，而无需迁移存量数据至Kudu。

架构

Kudu集群架构图如下。



Kudu包含如下两种类型的组件：

- Master Server：负责管理元数据。

  元数据包括Tablet Server的服务器的信息以及Tablet的信息，Master Server通过Raft协议提供高可用性。
- Tablet Server：用来存储Tablets。

  每个Tablet存在多个副本，副本之间通过Raft协议提供高可用性。

基本概念

名称	描述
Master服务	主要负责管理整个集群的元数据。元数据包括TabletServer信息、表的信息、Tablet的信息以及其他相关元数据相关的信息。
Tserver服务	负责为客户端储存和提供Tablets，仅Leader Tablet可以写入请求，其他的Tablet只能执行请求。
列式存储	Kudu是一种列数据储存结构，同一列的数据被存储在底层存储的相邻位置。
表（Table）	数据存储在Kudu的位置。Table有Schema和局有序的Primary Key属性，且可以划分为多个Tablet。
分片（Tablet）	一个表可以被分到若干个分片中，称为Tablet。 一个Tablet是指表上一段连续的Segment。一个特定的Tablet会被复制到多个Tablet服务器上，其中一个会被认为是Leader Tablet。每一个备份Tablet都可以支持读取和写入请求。
Raft	一致性协议，可以保证Kudu Master服务的高可用以及Tablet多副本之间数据一致性。
Catalog Table	Kudu的MetaData的中心位置，存储Table和Tablet的信息。

6.2.20.2. 基础使用

6.2.20.2.1. 常见命令

您可以在已经创建好的E-MapReduce（简称EMR）集群中，直接使用Kudu命令来操作。本文为您介绍Kudu的常见命令。

前提条件

- 已创建集群，并且选择了Kudu服务，详情请参见创建集群。
- 已登录集群，详情请参见登录集群。

背景信息

Kudu的常见命令如下所示：

- 查看Master列表
- 查看Tserver列表
- 查看集群健康信息
- 查看所有Table
- 查看表内容

- 检查集群Metrics

使用限制

创建的集群选择了Kudu服务。

查看Master列表

- 语法

```
kudu master list <your_Intranet_IP>
```

> **说明** 本文代码中的 `<your_Intranet_IP>` 为Kudu集群的内网IP地址，多个IP地址之间使用英文逗号（,）隔开。例如 `192.168.10.59,192.168.10.60,192.168.10.61` 。

- 示例：您可以执行以下命令，查看Master列表。

```
kudu master list 192.168.10.59,192.168.10.60,192.168.10.61
```

返回如下类似信息。

```
               uuid               |          rpc-addresses           |   role
----------------------------------+----------------------------------+----------
 3ebef6e84e0d45b2b6b5d24a2911**** | emr-header-1.cluster-23****:7051 | LEADER
 a4e0160acd804e8d83b4448183f6**** | emr-header-3.cluster-23****:7051 | FOLLOWER
 0d9fdf59efce48e58f18cf212c2c**** | emr-header-2.cluster-23****:7051 | FOLLOWER
```

查看Tserver列表

- 语法

```
kudu tserver list <your_Intranet_IP>
```

- 示例：您可以执行以下命令，查看Tserver列表。

```
kudu tserver list 192.168.10.59,192.168.10.60,192.168.10.61
```

返回如下类似信息。

```
               uuid               |          rpc-addresses
----------------------------------+----------------------------------
 be173a301ea24997a4a7a0f78815**** | emr-worker-2.cluster-23****:7050
 2a8257107d0048728707e783035e**** | emr-worker-1.cluster-23****:7050
```

查看集群健康信息

- 语法

```
kudu cluster ksck <your_Intranet_IP>
```

- 示例：您可以执行以下命令，查看集群健康信息。

```
kudu cluster ksck 192.168.10.59,192.168.10.60,192.168.10.61
```

返回如下类似信息。

```
Master Summary
              UUID               |    Address    | Status
-----------------------------------+---------------+---------
 0d9fdf59efce48e58f18cf212c2c**** | 192.168.10.60 | HEALTHY
 3ebef6e84e0d45b2b6b5d24a2911**** | 192.168.10.61 | HEALTHY
 a4e0160acd804e8d83b4448183f6**** | 192.168.10.59 | HEALTHY
              Flag                | Value  |  Tags  |         Master
-----------------------------------+--------+--------+-------------------------
 raft_get_node_instance_timeout_ms | 300000 | hidden | all 3 server(s) checked
Tablet Server Summary
              UUID               |             Address              | Status  | Location
-----------------------------------+----------------------------------+---------+----------
 2a8257107d0048728707e783035e**** | emr-worker-1.cluster-234593:7050 | HEALTHY | <none>
 be173a301ea24997a4a7a0f78815**** | emr-worker-2.cluster-234593:7050 | HEALTHY | <none>
Tablet Server Location Summary
 Location |  Count
----------+---------
 <none>   |       2
              Flag                | Value  |  Tags  |      Tablet Server
-----------------------------------+--------+--------+-------------------------
 raft_get_node_instance_timeout_ms | 300000 | hidden | all 2 server(s) checked
Version Summary
 Version |         Servers
---------+-------------------------
 1.10.0  | all 5 server(s) checked
The cluster doesn't have any matching tablets
The cluster doesn't have any matching tables
Tablet Replica Count Summary
   Statistic    | Replica Count
----------------+---------------
 Minimum        | 0
 First Quartile | 0
 Median         | 0
 Third Quartile | 0
 Maximum        | 0
==================
Warnings:
==================
Some masters have unsafe, experimental, or hidden flags set
Some tablet servers have unsafe, experimental, or hidden flags set
OK
```

查看所有Table

- 语法

```
kudu table list <your_Intranet_IP>
```

- 示例：您可以执行以下命令，查看所有Table。

```
kudu table list 192.168.10.59,192.168.10.60,192.168.10.61
```

返回如下类似信息。

```
impala::default.my_first_table
```

查看表内容

- 语法

```
kudu table describe <your_Intranet_IP> <your_tablename>
```

> **说明** 本文代码中的 `<your_tablename>` 为表名称，您可以通过查看所有Table获取表名称。例如 `impala::default.my_first_table` 。

- 示例：您可以执行以下命令，查看表内容。

```
kudu table describe 192.168.10.59,192.168.10.60,192.168.10.61 impala::default.my_first_table
```

返回如下类似信息。

```
ABLE impala::default.my_first_table (
    id INT64 NOT NULL,
    name STRING NULLABLE,
    PRIMARY KEY (id)
)
HASH (id) PARTITIONS 16
REPLICAS 1
```

检查集群Metrics

- 查看Master服务的Metrics

```
kudu-master --dump_metrics_json
```

- 查看Tserver服务的Metrics

```
kudu-tserver --dump_metrics_json
```

6.2.20.2.2. Impala集成Kudu

Impala集成Kudu后，您可以使用impala访问kudu的数据表。本文为您介绍Impala如何集成Kudu。

前提条件

已创建集群，并且选择了Impala和Kudu服务，详情请参见创建集群。

Impala集成Kudu

Impala集成Kudu的方式如下：

- 方式一：命令行方式
  i. 连接Impala，详情请参见Impala命令行工具。
  ii. 执行以下命令，新建表格。

  代码中添加了 `kudu.master_addresses` 来指定Kudu集群。代码示例如下。

```
create table my_first_table
    (
      id bigint,
      name string,
      primary key(id)
    )
    partition by hash partitions 16
    stored as kudu
    tblproperties(
      'kudu.master_addresses' = 'emr-header-1:7051,emr-header-2:7051,emr-header-3:7051',
      'kudu.num_tablet_replicas' = '1');
```

> **说明** 本文代码示例中的 `my_first_table` 为表名称，您可以自定义。

返回如下提示信息时，表示成功创建表。

```
[emr-header-1.cluster-234660:21000] default> create table my_first_table
                                         >     (
                                         >       id bigint,
                                         >       name string,
                                         >       primary key(id)
                                         >     )
                                         >     partition by hash partitions 16
                                         >     stored as kudu
                                         >     tblproperties(
                                         >       'kudu.master_addresses' = 'emr-header-1:7051,emr-header-2:7051,emr-header-3:7051',
                                         >       'kudu.num_tablet_replicas' = '1');
Query: create table my_first_table
    (
      id bigint,
      name string,
      primary key(id)
    )
    partition by hash partitions 16
    stored as kudu
    tblproperties(
      'kudu.master_addresses' = 'emr-header-1:7051,emr-header-2:7051,emr-header-3:7051',
      'kudu.num_tablet_replicas' = '1')
+-------------------------+
| summary                 |
+-------------------------+
| Table has been created. |
+-------------------------+
Fetched 1 row(s) in 0.06s
```

iii. （可选）您可以执行以下命令，向表中插入数据。

```
insert into my_first_table values(1,"ss");
```

iv. （可选）您可以执行以下命令，查询表数据。

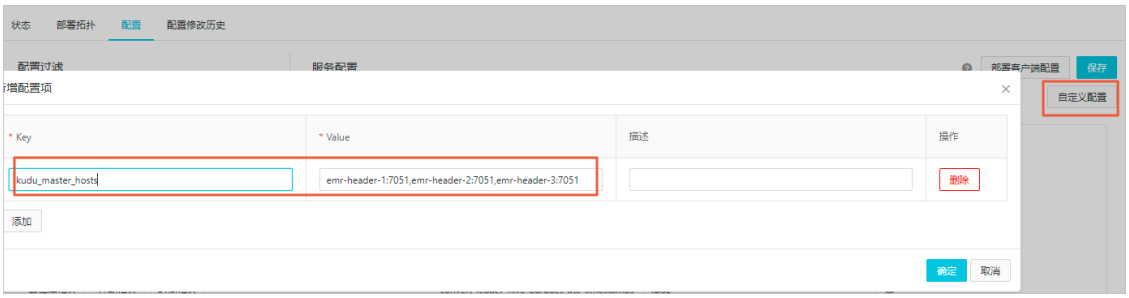
```
select * from my_first_table;
```

返回如下提示信息。

```
[emr-header-1.cluster-234660:21000] default> insert into my_first_table values(1,"ss");
Query: insert into my_first_table values(1,"ss")
Query submitted at: 2021-07-02 16:54:58 (Coordinator: http://emr-worker-1.cluster-234660:25000)
Query progress can be monitored at: http://emr-worker-1.cluster-234660:25000/query_plan?query_id=be481a9f8bf1df52:55586b8100000000
Modified 1 row(s), 0 row error(s) in 3.62s
[emr-header-1.cluster-234660:21000] default> select * from my_first_table;
Query: select * from my_first_table
Query submitted at: 2021-07-02 16:55:06 (Coordinator: http://emr-worker-1.cluster-234660:25000)
Query progress can be monitored at: http://emr-worker-1.cluster-234660:25000/query_plan?query_id=c64cdcf000867523:73c40be200000000
+----+------+
| id | name |
+----+------+
| 1  | ss   |
+----+------+
Fetched 1 row(s) in 0.15s
[emr-header-1.cluster-234660:21000] default>
```

> **说明** 您可以使用命令 `drop table my_first_table;` 删除表。

- 方式二：控制台方式
  i. 在EMR控制台新增配置。
     a. 在Impala服务的**配置**页面，单击**impalad.flgs**页签。
     b. 单击**自定义配置**。
     c. 在**新增配置项**对话框中，添加参数名为**kudu_master_hosts**，参数值为**emr-header-1:7051,emr-header-2:7051,emr-header-3:7051**的配置项来指定Kudu集群。



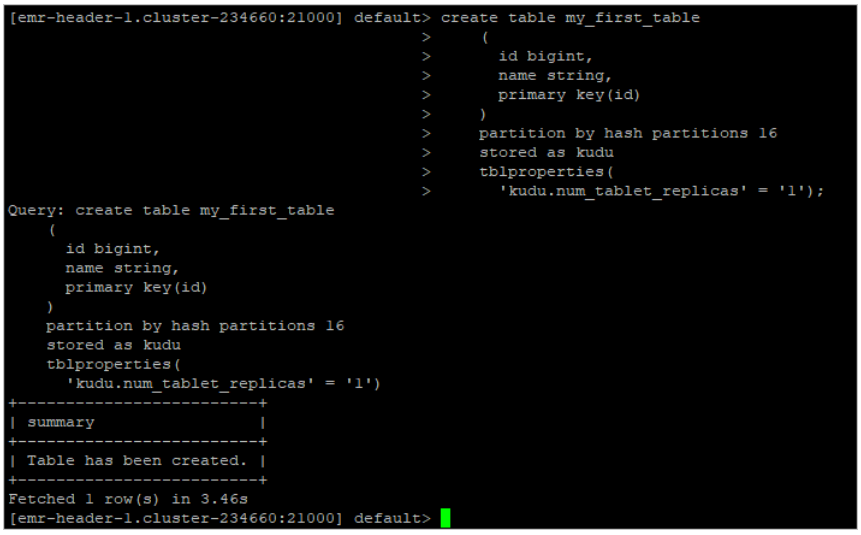
     d. 单击**确定**。
     e. 重复步骤 i ~步骤iv，在**catalogd.flgs**页签也添加参数名为**kudu_master_hosts**，参数值为**emr-header-1:7051,emr-header-2:7051,emr-header-3:7051**的配置项。
  ii. 保存配置。
     a. 单击**保存**。
     b. 在**确认修改**对话框中，输入**执行原因**，单击**确定**。
  iii. 重启配置。
     a. 在右上角选择**操作 > 重启All Components**。
     b. 在**执行集群操作**对话框中，输入**执行原因**，单击**确定**。
     c. 在**确认**对话框中，单击**确定**。
  iv. （可选）您可以登录集群查看集群连接情况。
     a. 连接Impala，详情请参见Impala命令行工具。

b. 执行以下命令，新建表格。

代码示例如下。

```
create table my_first_table
    (
      id bigint,
      name string,
      primary key(id)
    )
    partition by hash partitions 16
    stored as kudu
    tblproperties(
      'kudu.num_tablet_replicas' = '1');
```

返回如下提示信息时，表示成功创建表。



6.2.20.3. 开发指南

6.2.20.3.1. 操作表

本文为您介绍如何通过Java API接口方式进行Kudu表的相关操作。

前提条件

已创建集群，并且选择了Kudu服务，详情请参见创建集群。

背景信息

Kudu表相关的操作如下所示：

- 创建表
- 修改表
- 写数据
- 读数据
- 删除表

创建表

通过createTable方法创建表对象，其中需要指定表的schema和分区信息。

```
public static void createTable(String masterAddress, String tableName) throws KuduException {
    System.out.println("Connecting to " + masterAddress);
    KuduClient client = new KuduClient.KuduClientBuilder(masterAddress).build();
    List<ColumnSchema> columnSchemas = new ArrayList<>();
    columnSchemas.add(new ColumnSchema.ColumnSchemaBuilder("ID", Type.INT32).key(true).build());
    columnSchemas.add(new ColumnSchema.ColumnSchemaBuilder("score", Type.INT32).nullable(false).build());
    Schema tableSchema = new Schema(columnSchemas);
    CreateTableOptions options = new CreateTableOptions();
    List<String> partitionKey = new ArrayList<>();
    partitionKey.add("ID");
    options.addHashPartitions(partitionKey, 16);
    options.setNumReplicas(1);
    KuduTable personTable = client.createTable(tableName, tableSchema, options);
    System.out.println("Table " + personTable.getName());
}
```

说明 示例代码中，定义了一张表，包含ID和score两列。其中ID是INT32类型的主键字段，score是INT32类型的非主键字段且可以为空；同时该表在主键字段ID上做了16个hash分区，表示数据会分成16个独立的tablet。

修改表

- 通过alterTable方法修改表对象，增加名称为newCol的列。

```
public static void alterTable(String masterAddress, String tableName) throws KuduException {
    System.out.println("Connecting to " + masterAddress);
    KuduClient client = new KuduClient.KuduClientBuilder(masterAddress).build();
    String newColumnName = "newCol";
    AlterTableOptions addCollumnOptions = new AlterTableOptions();
    addCollumnOptions.addColumn(newColumnName, Type.STRING, "");
    AlterTableResponse response = client.alterTable(tableName, addCollumnOptions);
    System.out.println("Add column " + response.getElapsedMillis());
    KuduTable table = client.openTable(tableName);
    List<ColumnSchema> columnSchemas = table.getSchema().getColumns();
    for (int i=0; i<columnSchemas.size(); ++i) {
        ColumnSchema columnSchema = columnSchemas.get(i);
        System.out.println("Column " + i + ":" + columnSchema.getName());;
    }
}
```

- 通过alterTable方法修改表对象，删除名称为newCol的列。

```
public static void alterTable(String masterAddress, String tableName) throws KuduException {
    System.out.println("Connecting to " + masterAddress);
    KuduClient client = new KuduClient.KuduClientBuilder(masterAddress).build();
    String newColumnName = "newCol";
    AlterTableOptions deleteColumnOption = new AlterTableOptions();
    deleteColumnOption.dropColumn(newColumnName);
    response = client.alterTable(tableName, deleteColumnOption);
    System.out.println("Delete column " + response.getElapsedMillis());
    table = client.openTable(tableName);
    columnSchemas = table.getSchema().getColumns();
    for (int i=0; i<columnSchemas.size(); ++i) {
        ColumnSchema columnSchema = columnSchemas.get(i);
        System.out.println("Column " + i + ":" + columnSchema.getName());;
    }
}
```

写数据

通过newSession方法循环插入多条记录到已创建好的Kudu表，并检查返回结果。

```
public static void insertRows(String masterAddress, String tableName, int numRows) throws KuduException {
    System.out.println("Connecting to " + masterAddress);
    KuduClient client = new KuduClient.KuduClientBuilder(masterAddress).build();
    KuduTable table = client.openTable(tableName);
    KuduSession session = client.newSession();
    for (int i=0; i<numRows; ++i) {
        Upsert upsert = table.newUpsert();
        PartialRow row = upsert.getRow();
        row.addInt("ID", i+1);
        row.addInt("score", i+10);
        session.apply(upsert);
    }
    // Call session.close() to end the session and ensure the rows are
    // flushed and errors are returned.
    // You can also call session.flush() to do the same without ending the session.
    // When flushing in AUTO_FLUSH_BACKGROUND mode (the default mode recommended
    // for most workloads, you must check the pending errors as shown below, since
    // write operations are flushed to Kudu in background threads.
    session.close();
    if (session.countPendingErrors() != 0) {
        System.out.println("errors inserting rows");
        org.apache.kudu.client.RowErrorsAndOverflowStatus roStatus = session.getPendingErrors();
        org.apache.kudu.client.RowError[] errs = roStatus.getRowErrors();
        int numErrs = Math.min(errs.length, 5);
        System.out.println("there were errors inserting rows to Kudu");
        System.out.println("the first few errors follow:");
        for (int i = 0; i < numErrs; i++) {
            System.out.println(errs[i]);
        }
        if (roStatus.isOverflowed()) {
            System.out.println("error buffer overflowed: some errors were discarded");
        }
        throw new RuntimeException("error inserting rows to Kudu");
    }
    System.out.println("Inserted " + numRows + " rows");
}
```

说明 示例代码中，`numRows`是要写入的记录条数。

读数据

通过newScannerBuilder方法，依次读取Kudu表的所有记录。

```
public static void scanTable(String masterAddress, String tableName, int numRows) throws KuduException {
    System.out.println("Connecting to " + masterAddress);
    KuduClient client = new KuduClient.KuduClientBuilder(masterAddress).build();
    KuduTable table = client.openTable(tableName);
    Schema schema = table.getSchema();
    // Scan with a predicate on the 'key' column, returning the 'value' and "added" columns.
    List<String> projectColumns = new ArrayList<>(2);
    projectColumns.add("ID");
    projectColumns.add("score");
    int lowerBound = 0;
    KuduPredicate lowerPred = KuduPredicate.newComparisonPredicate(
            schema.getColumn("ID"),
            KuduPredicate.ComparisonOp.GREATER_EQUAL,
            lowerBound);
    int upperBound = numRows / 2;
    KuduPredicate upperPred = KuduPredicate.newComparisonPredicate(
            schema.getColumn("ID"),
            KuduPredicate.ComparisonOp.LESS,
            upperBound);
    KuduScanner scanner = client.newScannerBuilder(table)
            .setProjectedColumnNames(projectColumns)
            .addPredicate(lowerPred)
            .addPredicate(upperPred)
            .build();
    // Check the correct number of values and null values are returned, and
    // that the default value was set for the new column on each row.
    // Note: scanning a hash-partitioned table will not return results in primary key order.
    int resultCount = 0;
    int nullCount = 0;
    while (scanner.hasMoreRows()) {
        RowResultIterator results = scanner.nextRows();
        while (results.hasNext()) {
            RowResult result = results.next();
            resultCount++;
        }
    }
    int expectedResultCount = upperBound - lowerBound -1;
    if (resultCount != expectedResultCount) {
        throw new RuntimeException("scan error: expected " + expectedResultCount +
                " results but got " + resultCount + " results");
    }
    System.out.println("Scanned some rows and checked the results");
}
```

删除表

通过deleteTable方法，删除已存在的Kudu表。

```
public static void deleteTable(String masterAddress, String tableName) throws KuduException {
    System.out.println("Connecting to " + masterAddress);
    KuduClient client = new KuduClient.KuduClientBuilder(masterAddress).build();
    DeleteTableResponse response = client.deleteTable(tableName);
    System.out.println("Table delete " + response.getElapsedMillis());
}
```

6.2.20.4. 最佳实践

6.2.20.4.1. 数据迁移

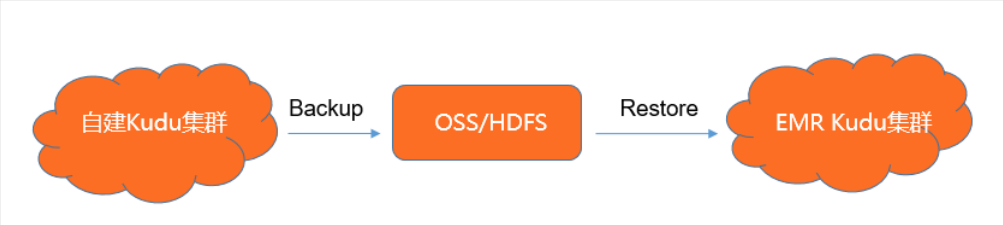
E-MapReduce（简称EMR）支持将您本地自建的Kudu集群迁移至EMR上。本文为您介绍如何迁移自建Kudu集群的数据到E-MapReduce上的Hadoop集群。

前提条件

- 您已自建Kudu集群。
- 已创建E-MapReduce的Hadoop集群，并选择了Kudu服务，详情请参见创建集群。

背景信息

EMR Kudu支持社区1.10以及1.11版本，可以使用社区提供的Backup和Restore的工具进行数据的迁移。基本流程示意图如下所示。



操作步骤

1. 执行以下命令，查看待迁移Kudu表的名称列表。

```
Kudu table list {YourKuduMasterAddress}
```

> **说明** 本文代码示例中的 `{YourKuduMasterAddress}` 为kudu集群的内网IP地址，多个IP地址之间使用英文逗号（,）隔开。

2. 使用Kudu Backup工具备份现有的Kudu集群表。

   默认的Kudu Backup可以根据现有Backup信息来执行增量或者全量备份。

   - 使用OSS作为中间存储

```
spark-submit --class org.apache.kudu.backup.KuduBackup kudu-backup2_2.11-1.10.0.jar \
  --kuduMasterAddresses master1-host,master-2-host,master-3-host \
  --rootPath oss://{your_BucketName}/kudu-backups {YourTableList}
```

> **说明** 本文代码示例中的 `{your_BucketName}` 为您OSS Bucket的名称， `{YourTableList}` 为您需要备份的Kudu表的列表。

   - 使用HDFS作为中间存储

```
spark-submit --class org.apache.kudu.backup.KuduBackup kudu-backup2_2.11-1.10.0.jar \
  --kuduMasterAddresses master1-host,master-2-host,master-3-host \
  --rootPath hdfs://{YourHDFSCluster}/kudu-backups {YourTableList}
```

> **说明** 本文代码示例中的 `{YourHDFSCluster}` 为Hadoop集群的地址。

3. 检查Kudu Backup目录相关数据。
4. 在EMR Kudu集群运行Restore命令， 将数据导入到EMR Kudu集群。

   默认命令如下：

   - 使用OSS作为中间存储

```
spark-submit --class org.apache.kudu.backup.KuduRestore kudu-backup2_2.11-1.10.0.jar \
  --kuduMasterAddresses master1-host,master-2-host,master-3-host \
  --rootPath oss://{your_BucketName}/kudu-backups {YourTableList}
```

   - 使用HDFS作为中间存储

```
spark-submit --class org.apache.kudu.backup.KuduRestore kudu-backup2_2.11-1.10.0.jar \
  --kuduMasterAddresses master1-host,master-2-host,master-3-host \
  --rootPath hdfs://{YourHDFSCluster}/kudu-backups {YourTableList}
```

5. 检查EMR Kudu集群上的数据、表名以及通过计算引擎查找表的数据是否一致。

   如果不一致，请提交工单。

6.2.20.5. 常见问题

本文汇总了Kudu使用时的常见问题。

- 在哪里查看Kudu的日志文件？
- Kudu支持的分区方式有哪些？
- 如何访问Kudu WebUI？
- Kudu客户端连接报错NonRecoverableException

在哪里查看Kudu的日志文件？

Kudu日志文件路径在*/mnt/disk1/log/kudu*下。

Kudu支持的分区方式有哪些?

Kudu支持Range分区方式以及Hash分区方式，两种分区方式可以嵌套使用，详情请参见Apache Kudu Schema Design。

如何访问Kudu WebUI?

因为Kudu WebUI与Knox还没有集成，所以不能通过Knox查看Kudu WebUI。您可以通过隧道的方式访问Kudu的WebUI，详情请参见通过SSH隧道方式访问开源组件Web UI。

Kudu客户端连接报错NonRecoverableException

报错详细信息，如下所示。

```
org.apache.kudu.client.NonRecoverableException: Could not connect to a leader master. Client configured with 1 master(s) (1
92.168.0.10:7051) but cluster indicates it expects 3 master(s) (192.168.0.36:7051,192.168.0.11:7051,192.168.0.10:7051)
```

此问题主要是因为在设置Master节点时，只设置了一个Master节点的信息，程序会找不到主Master节点。因此您在进行此配置时，需要配置所有Master节点的信息。

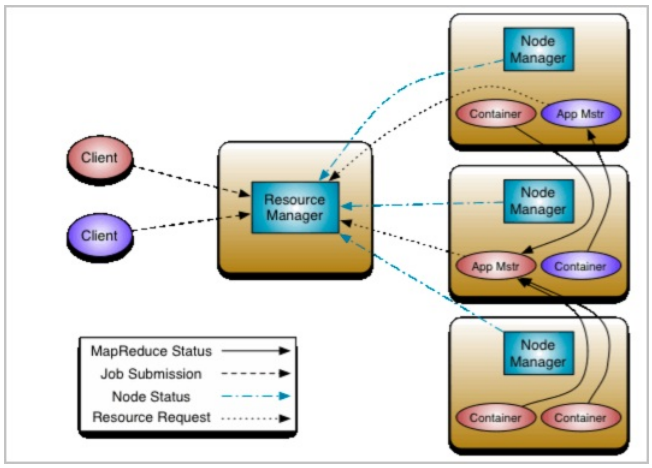
6.2.21. YARN

6.2.21.1. 概述

YARN是一个分布式的资源管理系统。YARN是Hadoop系统的核心组件，主要功能包括负责在Hadoop集群中的资源管理，负责对任务进行调度运行以及监控。

背景信息

YARN架构图如下所示。



YARN组件信息如下：

- ResourceManager：负责集群的资源管理与调度，为运行在YARN上的各种类型任务分配资源。

  非HA集群部署在EMR的Master节点上，HA集群部署在EMR的多个Master节点上，保证了高可用性。
- NodeMananger：负责节点的资源管理、监控和任务运行。

  部署在EMR的Core或Task节点上。
- ApplicationMaster：负责应用程序相关事务。

  例如，ApplicationMaster负责协调来自ResourceManager的资源，并通过NodeManager进行监控和资源管理等。
- YARN Client：负责提交任务。

  部署在EMR的Master、Core和Task节点上。
- JobHistory：解析MapReduce任务的指标，并展示任务执行情况。
- App Timeline Server：收集任务的指标，并展示任务执行情况。
- WebAppProxyServer：负责任务链接跳转，降低基于Web的攻击。

优势

EMR集群中的YARN优势如下：

- 高可用集群可以自动开启YARN HA部署。
- 便捷的运维。

例如，支持通过控制台的方式进行节点扩容，NodeMananger下线和滚动重启等操作。

- 支持监控报警。

  可以对各项指标进行监控和智能报警。

- 弹性伸缩支持优雅下线功能。

  可以在一段时间内等待用户任务执行结束后再下线，而不是直接下线NodeManager导致大量任务重新计算。

6.2.21.2. YARN授权

YARN的授权根据授权实体，可以分为服务级别的授权、队列级别的授权。

进入配置页面

1. 登录阿里云E-MapReduce控制台。
2. 在顶部菜单栏处，根据实际情况选择地域和资源组。
3. 单击上方的**集群管理**页签。
4. 在**集群管理**页面，单击相应集群所在行的**详情**。
5. 在左侧导航栏，选择**集群服务 > YARN**。
6. 单击**配置**页签。

服务级别的授权

详见Hadoop官方文档。

- 控制特定用户访问集群服务，例如提交作业。
- 配置在*hadoop-policy.xml*。
- 服务级别的权限校验在其他权限校验之前（例如，HDFS的permission检查或Yarn提交作业到队列控制）。

> **说明** 通常设置了HDFS permission检查或Yarn队列资源控制，可以不设置服务级别的授权控制，您可以根据自己需求进行相关配置。

队列级别的授权

YARN可以通过队列对资源进行授权管理，有Capacity Scheduler和Fair Scheduler两种队列调度。

这里以Capacity Scheduler为例。

- 添加配置

  队列也有两个级别的授权，一个是提交作业到队列的授权，一个是管理队列的授权。

  > **说明**
  > - 队列的ACL的控制对象为user或group，设置相关参数时，user和group可以同时设置，中间用空格分开，user/group内部可用逗号分开，只有一个空格表示任何人都没有权限。
  > - 队列ACL继承：如果一个user或group可以向某个队列中提交应用程序，则它可以向它的所有子队列中提交应用程序，同理管理队列的ACL也具有继承性。所以如果要防止某个user或group提交作业到某个队列，则需要设置该队列以及该队列的所有父队列的ACL来限制该user/group的提交作业的权限。

  - **yarn.acl.enable**

    ACL开关，设置为**true**。

  - **yarn.admin.acl**
    - YARN的管理员设置。例如，可以执行 `yarn rmadmin/yarn kill` 等命令时，该值必须配置，否则后续的队列相关的ACL管理员设置无法生效。
    - 如上备注，配置值时可以设置user或group。

      ```
      user1,user2 group1,group2 #user和group用空格隔开。
        group1,group2 #只有group情况下，必须在最前面加上空格。
      ```

      EMR集群中需将has配置为admin的ACL权限。

- **yarn.scheduler.capacity.${queue-name}.acl_submit_applications**
  - 设置能够向该队列提交的user或group。
  - 其中${queue-name}为队列的名称，可以是多级队列，注意多级情况下的ACL继承机制。

```
#queue-name=root
  <property>
      <name>yarn.scheduler.capacity.root.acl_submit_applications</name>
      <value> </value> #空格表示任何人都无法往root队列提交作业。
  </property>
 #queue-name=root.testqueue
 <property>
   <name>yarn.scheduler.capacity.root.testqueue.acl_submit_applications</name>
      <value>test testgrp</value> #testqueue只允许test用户或testgrp组提交作业。
  </property>
```

- **yarn.scheduler.capacity.${queue-name}.acl_administer_queue**
  - 设置某些user或group管理队列，例如终止队列中作业等。
  - queue-name可以是多级，注意多级情况下的ACL继承机制。

```
#queue-name=root
  <property>
      <name>yarn.scheduler.capacity.root.acl_administer_queue</name>
      <value> </value>
  </property>
 #queue-name=root.testqueue
 <property>
   <name>yarn.scheduler.capacity.root.testqueue.acl_administer_queue</name>
      <value>test testgrp</value>
  </property>
```

- 重启YARN服务
  - 对于Kerberos安全集群已经默认开启ACL，用户可以根据自己需求配置队列的相关ACL权限控制。
  - 对于非Kerberos安全集群根据上述开启ACL并配置好队列的权限控制，重启YARN服务。
    a. 在**集群服务 > YARN**页面，选择右上角的**操作 > 重启 All Components**。
    b. 执行集群操作。

       单击右上角**查看操作历史**查看任务进度，等待任务完成。
- 配置示例
  - yarn-site.xml

Key	Value
yarn.acl.enable	true
yarn.admin.acl	has

  - capacity-scheduler.xml
  - default队列：禁用default队列，不允许任何用户提交或管理。
  - q1队列：只允许test用户提交作业以及管理队列。
  - q2队列：只允许foo用户提交作业以及管理队列。

```
<configuration>
    <property>
        <name>yarn.scheduler.capacity.maximum-applications</name>
        <value>10000</value>
        <description>Maximum number of applications that can be pending and running.</description>
    </property>
    <property>
        <name>yarn.scheduler.capacity.maximum-am-resource-percent</name>
        <value>0.25</value>
        <description>Maximum percent of resources in the cluster which can be used to run application masters i.e.
            controls number of concurrent running applications.
        </description>
    </property>
    <property>
        <name>yarn.scheduler.capacity.resource-calculator</name>
        <value>org.apache.hadoop.yarn.util.resource.DefaultResourceCalculator</value>
    </property>
```

```
<property>
    <name>yarn.scheduler.capacity.root.queues</name>
    <value>default,q1,q2</value>
    <!-- 3个队列-->
    <description>The queues at the this level (root is the root queue).</description>
</property>
<property>
    <name>yarn.scheduler.capacity.root.default.capacity</name>
    <value>0</value>
    <description>Default queue target capacity.</description>
</property>
<property>
    <name>yarn.scheduler.capacity.root.default.user-limit-factor</name>
    <value>1</value>
    <description>Default queue user limit a percentage from 0.0 to 1.0.</description>
</property>
<property>
    <name>yarn.scheduler.capacity.root.default.maximum-capacity</name>
    <value>100</value>
    <description>The maximum capacity of the default queue.</description>
</property>
<property>
    <name>yarn.scheduler.capacity.root.default.state</name>
    <value>STOPPED</value>
    <!-- default队列状态设置为STOPPED-->
    <description>The state of the default queue. State can be one of RUNNING or STOPPED.</description>
</property>
<property>
    <name>yarn.scheduler.capacity.root.default.acl_submit_applications</name>
    <value> </value>
    <!-- default队列禁止提交作业-->
    <description>The ACL of who can submit jobs to the default queue.</description>
</property>
<property>
    <name>yarn.scheduler.capacity.root.default.acl_administer_queue</name>
    <value> </value>
    <!-- 禁止管理default队列-->
    <description>The ACL of who can administer jobs on the default queue.</description>
</property>
<property>
    <name>yarn.scheduler.capacity.node-locality-delay</name>
    <value>40</value>
</property>
<property>
    <name>yarn.scheduler.capacity.queue-mappings</name>
    <value>u:test:q1,u:foo:q2</value>
    <!-- 队列映射，test用户自动映射到q1队列-->
    <description>A list of mappings that will be used to assign jobs to queues. The syntax for this list is
        [u|g]:[name]:[queue_name][,next mapping]* Typically this list will be used to map users to queues,for
        example, u:%user:%user maps all users to queues with the same name as the user.
    </description>
</property>
<property>
    <name>yarn.scheduler.capacity.queue-mappings-override.enable</name>
    <value>true</value>
    <!-- 上述queue-mappings设置的映射，是否覆盖客户端设置的队列参数-->
    <description>If a queue mapping is present, will it override the value specified by the user? This can be used
        by administrators to place jobs in queues that are different than the one specified by the user. The default
        is false.
    </description>
</property>
<property>
    <name>yarn.scheduler.capacity.root.acl_submit_applications</name>
    <value> </value>
    <!-- ACL继承性，父队列需控制住权限-->
    <description>
        The ACL of who can submit jobs to the root queue.
    </description>
</property>
<property>
    <name>yarn.scheduler.capacity.root.q1.acl_submit_applications</name>
    <value>test</value>
```

```
        <!-- q1只允许test用户提交作业-->
    </property>
    <property>
        <name>yarn.scheduler.capacity.root.q2.acl_submit_applications</name>
        <value>foo</value>
        <!-- q2只允许foo用户提交作业-->
    </property>
    <property>
        <name>yarn.scheduler.capacity.root.q1.maximum-capacity</name>
        <value>100</value>
    </property>
    <property>
        <name>yarn.scheduler.capacity.root.q2.maximum-capacity</name>
        <value>100</value>
    </property>
    <property>
        <name>yarn.scheduler.capacity.root.q1.capacity</name>
        <value>50</value>
    </property>
    <property>
        <name>yarn.scheduler.capacity.root.q2.capacity</name>
        <value>50</value>
    </property>
    <property>
        <name>yarn.scheduler.capacity.root.acl_administer_queue</name>
        <value> </value>
        <!-- ACL继承性，父队列需控制住权限-->
    </property>
    <property>
        <name>yarn.scheduler.capacity.root.q1.acl_administer_queue</name>
        <value>test</value>
        <!-- q1队列只允许test用户管理，如kill作业-->
    </property>
    <property>
        <name>yarn.scheduler.capacity.root.q2.acl_administer_queue</name>
        <value>foo</value>
        <!-- q2队列只允许foo用户管理，如kill作业-->
    </property>
    <property>
        <name>yarn.scheduler.capacity.root.q1.state</name>
        <value>RUNNING</value>
    </property>
    <property>
        <name>yarn.scheduler.capacity.root.q2.state</name>
        <value>RUNNING</value>
    </property>
</configuration>
```

6.2.21.3. 最佳实践

6.2.21.3.1. 使用YARN CGroups功能对CPU进行控制测试

CGroups（Control Groups）是Linux内核提供的一种功能，用来控制、限制与隔离一个进程组群的资源，包括CPU、内存和磁盘输入输出等资源。

概述

YARN中集成了CGroups的功能，使得NodeManager可以对Container的CPU的资源使用进行控制，例如可以对单个Container的CPU使用进行控制，也可以对NodeManager管理的总CPU进行控制。

开启YARN CGroups功能

> **说明** E-MapReduce集群中的YARN默认没有开启CGroups的功能，需要用户根据需求自行开启。

1. 登录阿里云 E-MapReduce 控制台。
2. 进入YARN的**集群服务**页面。
   i. 单击上方的**集群管理**页签。
   ii. 在**集群管理**页面，单击相应集群所在行的**集群ID**。

iii. 在左侧导航栏，单击**集群服务 > YARN**。

3. 启用CGroups。
   i. 单击右上角的**操作 > 启用 CGroups**。
   ii. 在**执行集群操作**页面，根据需求设置相应参数。
   iii. 单击**确定**。
   iv. 在**确认**页面，单击**确定**。
4. 重启NodeManager。
   i. 单击右上角的**操作 > 重启NodeManager**。
   ii. 在**执行集群操作**页面，根据需求设置相应参数。
   iii. 单击**确定**。
   iv. 在**确认**页面，单击**确定**。

控制参数

在开启了CGroups功能的前提下，可以通过调节YARN中的参数来控制CPU的资源使用行为：

1. 登录阿里云 E-MapReduce 控制台。
2. 进入YARN的**集群服务**页面。
   i. 单击上方的**集群管理**页签。
   ii. 在**集群管理**页面，单击相应集群所在行的**集群ID**。
   iii. 在左侧导航栏，单击**集群服务 > YARN**。
3. 调节参数。
   i. 单击**配置**页签。
   ii. 调节如下参数。

参数	描述
yarn.nodemanager.resource.percentage-physical-cpu-limit	NodeManager管理的所有Container使用CPU的阈值，默认100%。 **说明** 任何场景下，NodeManager管理的Container的CPU都不能超过yarn.nodemanager.resource.percentage-physical-cpu-limit设置的比例。
yarn.nodemanager.linux-container-executor.cgroups.strict-resource-usage	对Container的CPU使用资源是否严格按照被分配的比例来进行控制。默认是false，即Container可以使用空闲的CPU。

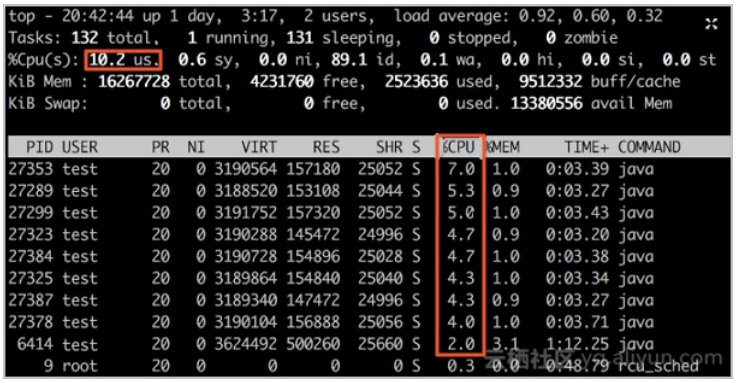
总Container的CPU控制测试

通过调节**yarn.nodemanager.resource.percentage-physical-cpu-limit**参数，以控制NodeManager管理的所有Container的CPU使用率。

下面集群配置为3台4核16GB，其中2台NodeManager，1台ResourceManager，分别设置该值以10、30和50为例，在YARN中运行一个hadoop pi作业，观察NodeManager所在机器的CPU的使用率。

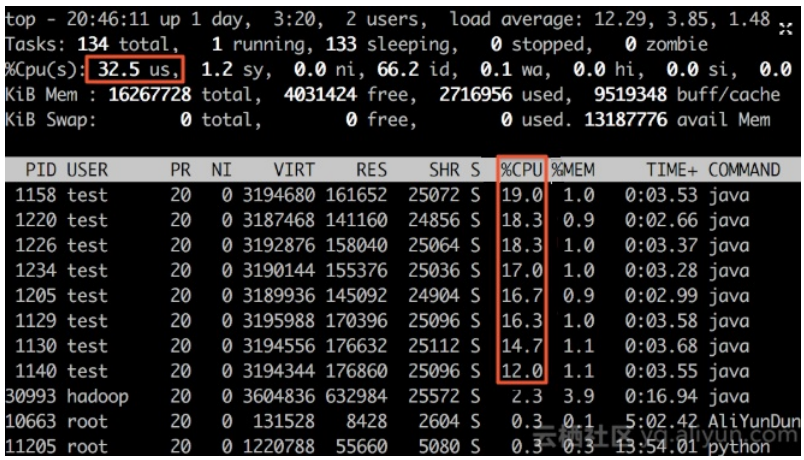
说明 下图中的 `%CPU` 表示进程占用单个核的比例；`%CPU(s)` 表示所有用户进程占总CPU的比例。

- 设置参数为10。



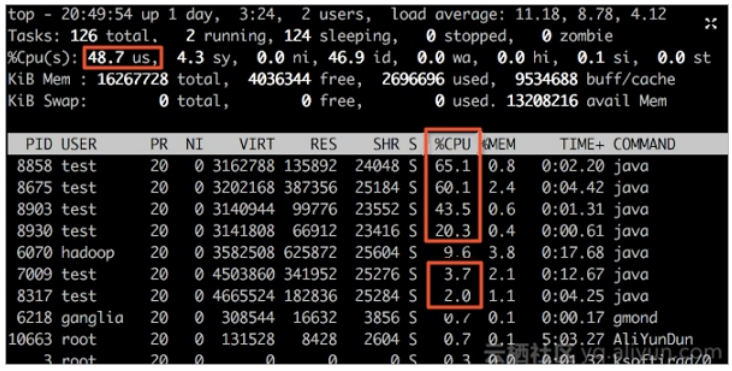
如上图所示，`%CPU(s)` 接近10%；`%CPU` 表示所有的test用户的Container进程加起来（7%+5.3%+5%+4.7%+4.7%+4.3%+4.3%+4%+2%=41.3%=0.413个核，约等于10%*4core=0.4核，即4个核的10%比例）。

- 设置参数为30。



如上图所示，`%CPU(s)` 接近30%；`%CPU` 表示所有的test用户的Container进程加起来（19%+18.3%+18.3%+17%+16.7%+16.3%+14.7%+12%=132.3%=1.323个核，约等于30%*4core=1.2核，即4个核的30%比例）。

- 设置参数为50。



如上图所示，`%CPU(s)` 接近50%；`%CPU` 表示所有的test用户的Container进程加起来（65.1%+60.1%+43.5%+20.3%+3.7%+2%=194.7%=1.947个核，约等于50%*4core=2核，即4个核的50%比例）。

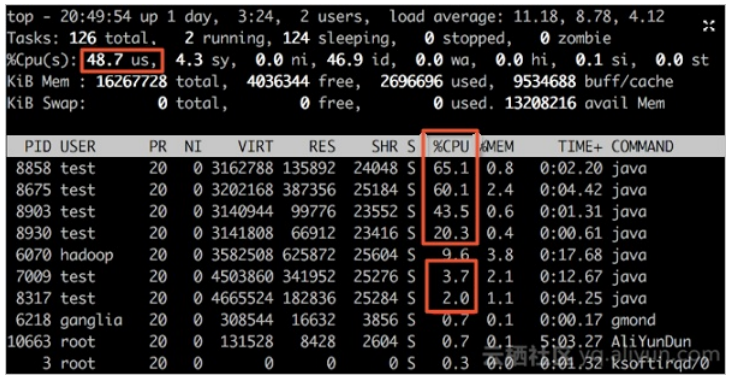
Container间的CPU控制测试

NodeManager上面启动多个Container，所有这些Container对CPU资源的占用不超过总Container的CPU控制测试中设置**yarn.nodemanager.resource.percentage-physical-cpu-limit**的阈值，NodeManager有共享模式（share）和严格模式（strict）两种方式，来管理和控制多个Container之间的CPU使用率，这两种方式通过参数**yarn.nodemanager.linux-container-executor.cgroups.strict-resource-usage**控制。

- 共享模式（share）

  当**yarn.nodemanager.linux-container-executor.cgroups.strict-resource-usage**设置为*false*时，即为共享模式（默认为false）。在这种模式下，Container除了实际被需要分配的CPU资源外，还可以利用空闲的CPU资源。

  例如，设置**yarn.nodemanager.resource.percentage-physical-cpu-limit**为*50*，设置**yarn.nodemanager.linux-container-executor.cgroups.strict-resource-usage**为*false*。NodeManager所在节点是4核，那么该Container申请按比例被分配的cpu资源为（1vcore÷8vcore）*（4core*50%）=0.25核，但是如果CPU有空闲，理论上该Container可以占满NodeManager管理的上限4core*50%=2核。

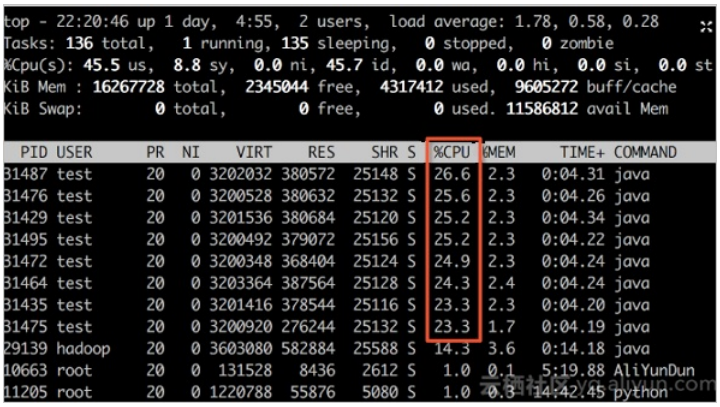


上图可以看出，test用户的多个Container进程占用CPU核数的比例相差很大（65%=0.65核；60.1%=0.61核； 3.7%=0.037核等），即单个Container的CPU使用没有被严格限制在（1vcore÷8vcore）*（4core*50%）=0.25核。

- 严格模式（strict）

  当**yarn.nodemanager.linux-container-executor.cgroups.strict-resource-usage**设置为*true*时，即为严格模式（strict）。在这种模式下，Container只能使用被需要分配的CPU资源，即使CPU有空闲也不能使用。

  例如，设置**yarn.nodemanager.resource.percentage-physical-cpu-limit**为*50*，设置**yarn.nodemanager.linux-container-executor.cgroups.strict-resource-usage**为*true*。



上图可以看出，test用户的多个Container进程占用CPU核数均约在0.25核（26.6%=0.266核；24.9%=0.249核），即该Container实际应该被分配的CPU使用率（1vcore÷8vcore）*（4core*50%）=0.25核。

6.2.21.4. 常见问题

本文汇总了YARN使用时的常见问题。

- 集群问题汇总
  - 集群有状态重启包括哪些内容？
  - 如何启用RM HA？
  - 如何启用CPU CGroups？
  - 如何检查ResourceManager服务是否正常？
  - 如何了解应用运行状况？
  - 应用问题排查流程
  - 资源组或队列（Queue）管理：
    - 如何配置热更新？
    - 如何处理Queue内应用分配不均？
- 应用问题汇总
  - RM

    RM处于Standby状态，无法自动恢复Active状态，该如何处理？
  - NM
    - 为什么节点启动任务时Localize失败或任务日志无法采集与删除？
    - 资源本地化异常，该如何处理？
    - Container启动失败或运行异常，报错提示No space left on device，该如何处理？
    - 节点NM服务或任务运行时无法正常解析域名，该如何处理？
  - UI或REST API
    - 报错提示User [dr.who] is not authorized to view the logs for application ***，该如何处理？
    - 报错提示HTTP ERROR 401 Authentication required或HTTP ERROR 403 Unauthenticated users are not authorized to access this page，该如何处理？
    - 为什么TotalVcore显示值不准确？
    - TEZ UI展示的应用信息不全，该如何处理？

集群有状态重启包括哪些内容？

集群有状态重启包括RM Restart和NM Restart两部分，ResourceManager（简称RM）负责维护应用级基础信息与状态，NodeManager（简称NM）负责维护运行时的Container信息与状态，它们持续将相关状态同步至外部存储（Zookeeper、LevelDB和HDFS等），并在重启后重新加载状态自行恢复，保证集群升级或重启后应用自动恢复，通常情况下可以做到应用无感知。

如何启用RM HA?

您可以在EMR控制台YARN服务的配置页签，检查或者配置以下参数启用RM HA。

参数	描述
yarn.resourcemanager.ha.enabled	设置为true，表示开启HA。 默认值为false。
yarn.resourcemanager.ha.automatic-failover.enabled	使用默认值true，表示启用自动切换。
yarn.resourcemanager.ha.automatic-failover.embedded	使用默认值true，表示使用嵌入的自动切换方式。
yarn.resourcemanager.ha.curator-leader-elector.enabled	设置为true，表示使用curator组件。 默认值为false。
yarn.resourcemanager.ha.automatic-failover.zk-base-path	使用默认值/yarn-leader-electionleader-elector。

如何启用CPU CGroups?

Hadoop官方文档请参见Using CGroups with YARN。启用CPU CGroups的操作步骤如下所示：

1. 在EMR控制台YARN服务页面，选择右上角的**操作 > 启用CGroups**。详情信息，请参见开启YARN CGroups功能。
2. 在EMR控制台YARN服务的**配置**页面，单击**yarn-site**页签，新增参数为yarn.nodemanager.container-executor.class，参数值为org.apache.hadoop.yarn.server.nodemanager.LinuxContainerExecutor的配置项，保存该配置项并重启服务。

   > **说明** 如果您希望在开启了CGroups功能的前提下，可以通过调节YARN中的参数来控制CPU的资源使用行为，则需要更新配置项yarn.nodemanager.linux-container-executor.cgroups.strict-resource-usage为true，默认值是false，即Container可以使用空闲的CPU。

3. 重启NodeManager。

   在EMR控制台YARN服务页面的配置页签，选择右上角的**操作 > 重启NodeManager**。

如何配置热更新?

> **注意** 以下操作内容适用于Hadoop 3.2.0及后续版本。

1. 开启关键配置。

   您可以在EMR控制台YARN服务的配置页签，检查或者配置以下参数。

参数	描述	参数值（推荐）
yarn.scheduler.configuration.store.class	使用的后备存储的类型。例如，设置为fs，表示使用FileSystem类型存储。	fs
yarn.scheduler.configuration.max.version	保留历史版本的最大数量，超出自动清理。	100
yarn.scheduler.configuration.fs.path	*capacity-scheduler.xml*文件存储路径。 不存在存储路径时，则自动创建。不指定前缀时，则默认defaultFs的相对路径。	/yarn/<集群名>/scheduler/conf **注意** 将<集群名>替换为集群名称以便区分，可能有多个YARN集群对应同一分布式存储的情况。

2. 查看*capacity-scheduler.xml*配置。
   - 方式一（REST API）：http://<rm-address>/ws/v1/cluster/scheduler-conf。
   - 方式二（HDFS文件）：${yarn.scheduler.configuration.fs.path}/capacity-scheduler.xml.<timestamp>，其中后缀<timestamp>值最大的文件是最新的配置文件。
3. 更新配置。

   示例：更新一个配置项yarn.scheduler.capacity.maximum-am-resource-percent，并删除配置项yarn.scheduler.capacity.xxx，删除某配置项只需去掉其Value值。

```
curl -X PUT -H "Content-type: application/json" 'http://<rm-address>/ws/v1/cluster/scheduler-conf' -d '
{
  "global-updates": [
    {
      "entry": [{
        "key":"yarn.scheduler.capacity.maximum-am-resource-percent",
        "value":"0.2"
      },{
        "key":"yarn.scheduler.capacity.xxx"
      }]
    }
  ]
}'
```

如何处理Queue内应用分配不均？

注意 以下操作内容适用于Hadoop 2.8.0及后续版本。

Queue内部资源通常会被大作业占据，小作业较难获得资源，希望各作业能公平获得资源。您可以通过以下步骤处理：

1. 修改Queue配置yarn.scheduler.capacity.<queue-path>.ordering-policy，使其调度次序由fifo（默认值）改为fair。

   说明 FIFO Scheduler为先进先出调度器，Fair Scheduler为公平调度器。

   您还可以修改参数yarn.scheduler.capacity.<queue-path>.ordering-policy.fair.enable-size-based-weight，该参数默认值false，表示按used资源值从小到大排序，参数值为true时表示按照used或demand计算值从小到大排序。

2. 开启队列内抢占。

   队列内抢占常用参数如下表所示。

参数	描述	参数值（推荐）
yarn.resourcemanager.scheduler.monitor.enable	抢占功能总开关，在*yarn-site.xml*中配置，其余参数是在*capacity-scheduler.xml*中配置。	true
yarn.resourcemanager.monitor.capacity.preemption.intra-queue-preemption.enabled	队列内抢占开关（队列间抢占无开关，默认开启）。	true
yarn.resourcemanager.monitor.capacity.preemption.intra-queue-preemption.preemption-order-policy	抢占策略优先考虑应用优先级，默认值为userlimit_first。	priority_first
yarn.scheduler.capacity.<queue-path>.disable_preemption	指定Queue不被抢占，默认false。 true表示不允许指定Queue资源被抢占，子Queue未配置则继承父Queue配置值。	true
yarn.scheduler.capacity.<queue-path>.intra-queue-preemption.disable_preemption	指定Queue不开启队列内抢占，默认值false。 true表示禁用Queue内抢占，子Queue未配置则继承父Queue配置值。	true

如何检查ResourceManager服务是否正常？

您可以通过以下方式检查：

- 检查ResourceManager HA状态（如果集群已启用HA，需确保有且只有一个Active ResourceManager），以下方式任选一种，检查字段haState是否为ACTIVE或STANDBY，haZooKeeperConnectionState是否为CONNECTED：
  - 命令行：yarn rmadmin -getAllServiceState
  - REST API：http://<rmAddress>/ws/v1/cluster/info

  示例如下。

```
▼<clusterInfo>
   <id>16225960[redacted]</id>
   <startedOn>1622596090362</startedOn>
   <state>STARTED</state>
   <haState>ACTIVE</haState>
   <rmStateStoreName>org.apache.hadoop.yarn.server.resourcemanager.recovery.ZKRMStateStore</rmStateStoreName>
   <resourceManagerVersion>2.8.5</resourceManagerVersion>
   <resourceManagerBuildVersion>2.8.5 from 8f83e14b1611105177edd1e2d671fa71964d3578 by jenkins source checksum a616fa5ffd7d69e81794b24b615ccd23</resourceManagerBuildVersion>
   <resourceManagerVersionBuiltOn>2020-12-28T08:15Z</resourceManagerVersionBuiltOn>
   <hadoopVersion>2.8.5</hadoopVersion>
   <hadoopBuildVersion>2.8.5 from 8f83e14b1611105177edd1e2d671fa71964d3578 by jenkins source checksum 67b0bf49685455778b11c6a23fb123e</hadoopBuildVersion>
   <hadoopVersionBuiltOn>2020-12-28T08:14Z</hadoopVersionBuiltOn>
   <haZooKeeperConnectionState>CONNECTED</haZooKeeperConnectionState>
 </clusterInfo>
```

- 检查应用状态。

  执行以下命令，查看是否有持续的SUBMITTED或ACCEPTED状态。

  ```
  yarn application -list
  ```

- 查看提交的新应用是否可以正常运行并结束。命令示例如下。

  ```
  hadoop jar <hadoop_home>/share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-*-tests.jar sleep -m 1 -mt 1000 -r 0
  ```

  您可以在 `sleep -m` 之间新增配置项以指定Queue，新增的参数为-Dmapreduce.job.queuename，参数值为default。

如何了解应用运行状况？

您可以查看以下信息。

查询信息	描述
基本信息	包括ID、User、Name、Application Type、State、Queue、App-Priority、StartTime、FinishTime、State、FinalStatus、Running Containers、Allocated CPU VCores、Allocated Memory MB和Diagnostics（诊断信息）等。 • App列表页：http://<rmAddress>/cluster/apps。 • App详情页（在App列表页面，单击App ID进入）：http://<rmAddress>/cluster/app/<applicationId>。 • App Attempt详情页（在App详情页面，单击App Attempt ID进入）：http://<rmAddress>/cluster/appattempt/<appAttemptId>。 • App REST API：http://<rmAddress>/ws/v1/cluster/apps/<applicationId>。 • App Attempts REST API: http://<rmAddress>/ws/v1/cluster/apps/<applicationId>/appattempts。
Queue信息	• Scheduler页面（叶子节点展开）：http://<rmAddress>/cluster/scheduler。 • Scheduler REST API：http://<rmAddress>/ws/v1/cluster/scheduler。

Container日志	• RUNNING Applications
	  ○ 通过NodeManager Log页面查看：http://<nmHost>:8042/node/containerlogs/<containerId>/<user>。
	  ○ 在运行节点${yarn.nodemanager.local-dirs}目录下查找*<containerId>*子目录。
	• Finished Applications
	  ○ 通过 `yarn logs -applicationId <applicationId> -appOwner <user> -containerId <containerId>` 命令查询。
	  ○ 通过HDFS命令 `hadoop fs -ls /logs/<user>/logs/<applicationId>` 查询。

应用问题排查流程

1. 检查App状态，通过App详情页或App REST API检查应用状态。
   - NEW_SAVING：该状态处于Zookeeper State Store写入应用信息阶段。持续该状态可能原因如下：
     - Zookeeper存在问题，检查Zookeeper服务是否正常。
     - Zookeeper读写数据问题，处理方法请参见RM处于Standby状态，无法自动恢复Active状态，该如何处理？。
   - SUBMITTED：该状态极少遇到，可能原因为Node Update请求太多造成Capacity Scheduler内部抢锁堵塞，通常发生在大规模集群，需优化相关流程。
   - ACCEPTED：检查Diagnostics。请根据提示信息，选择相应的处理方式。
     - 报错提示Queue's AM resource limit exceeded。
       - 可能原因：Queue AM已使用资源和AM资源之和超出了Queue AM资源上限，UI条件为${Used Application Master Resources} + ${AM Resource Request} < ${Max Application Master Resources}。
       - 处理方法：上调该Queue的AM资源上限。例如，设置参数yarn.scheduler.capacity.\<queue-path\>.maximum-am-resource-percent为0.5。
     - 报错提示User's AM resource limit exceeded。
       - 可能原因：Queue user AM已使用资源和AM资源之和超出了Queue user AM资源上限。
       - 处理方法：提高user-limit比例。您可以修改参数yarn.scheduler.capacity.\<queue-path\>.user-limit-factor和yarn.scheduler.capacity.\<queue-path\>.minimum-user-limit-percent的参数值。

- 报错提示AM container is launched, waiting for AM container to Register with RM。
  - 可能原因：AM已启动，内部初始化未完成（例如，Zookeeper连接超时等）。
  - 处理方法：需要根据AM日志进一步排查问题。
- 报错提示Application is Activated, waiting for resources to be assigned for AM。

  执行步骤3，检查AM资源分配为何未满足。

- RUNNING：执行步骤2，检查Container资源请求是否完成。

2. 确认YARN资源分配还未完成。
   i. 在apps列表页，单击appID进入AM页面。
   ii. 单击下方列表的AM-ID，进入AM-Attempt页面。
   iii. 查看Total Outstanding Resource Requests列表中是否有Pending资源（也可以通过PendingRequests REST API查询）：
   - 没有Pending资源：说明YARN已分配完毕，退出该检查流程，检查AM情况。

     Total Outstanding Resource Requests: <memory:0, vCores:0, max_memory-mb: 0Mi, guaranteed_memory-mb: 0Mi, guaranteed_vcores: 0, vssd: 0, max_vcores: 0>

Priority	ResourceName	Capability	NumContainers	RelaxLocality	NodeLabelExpression

   - 有Pending资源：说明YARN资源分配未完成，继续下一步检查。
3. 资源限制检查。

   检查集群或Queue资源，查看资源信息，例如，Effective Max Resource和Used Resources。
   i. 检查集群资源或所在Queue资源或其父Queue资源是否已用完。
   ii. 检查叶子队列某维度资源是否接近或达到上限。
   iii. 当集群资源接近用满时（例如85%以上），应用的分配速度可能会变慢，因为大部分机器都没有资源了，分配的机器没有资源就会reserve，reserve到一定个数后分配就会变慢，另外也可能是由于内存资源和CPU资源不成比例，例如，有的机器上内存资源有空闲但CPU资源已经用满，有的机器上CPU资源有空闲但内存已经用满。
4. 检查是否存在Container资源分配成功但启动失败的情况，YARN UI的App Attempt页面可以看到已分配的Container数（间隔一小段时间观察是否变化），如果有Container启动失败，查看对应的NM日志或Container日志进一步排查失败原因。

客户端异常，提示Exception while invoking getClusterNodes of class ApplicationClientProtocolPBClientImpl over rm2 after 1 fail over attempts. Trying to fail over immediately

- 问题现象：无法正常访问Active ResourceManager。ResourceManager日志异常，提示信息：WARN org.apache.hadoop.ipc.Server: Incorrect header or version mismatch from 10.33.**.**:53144 got version 6 expected version 9。
- 问题原因：Hadoop版本太旧，导致客户端RPC版本不兼容。
- 处理方法：使用匹配的Hadoop版本。

RM处于Standby状态，无法自动恢复Active状态，该如何处理？

您可以通过以下方式排查问题：

1. 检查支持自动恢复的必选配置项是否配置正确。

参数	描述
yarn.resourcemanager.ha.enabled	需要配置为true。
yarn.resourcemanager.ha.automatic-failover.enabled	需要配置为true。
yarn.resourcemanager.ha.automatic-failover.embedded	需要配置为true。

2. 修改为上述配置后，问题还未解决，您可以通过以下方式排查问题：
   - 检查Zookeeper服务是否正常。
   - 检查Zookeeper客户端（RM）读取数据是否超出其Buffer上限。
     - 问题现象：RM日志内存在异常，提示Zookeeper error len*** is out of range!或Unreasonable length = ***。
     - 处理方法：在EMR控制台YARN服务的配置页面，单击**yarn-env**页签，更新参数**yarn_resourcemanager_opts**的参数值为-Djute.maxbuffer=4194304，然后重启RM。
   - Zookeeper服务端写入数据是否超出其Buffer上限。
     - 问题现象：Zookeeper日志内存在异常，提示Exception causing close of session 0x1000004d5701b6a: Len error ***。
     - 处理方法：Zookeeper服务各节点新增或更新配置项-Djute.maxbuffer= (单位：bytes)，将Buffer上限调大超过异常显示的长度。
   - 如果RM或Zookeeper日志都找不到异常，可能是因为RM选主Zookeeper Ephemeral Node（${yarn.resourcemanager.zk-state-

store.parent-path}/${yarn.resourcemanager.cluster-id}/ActiveStandbyElectorLock）被其他Session持有未释放，可在zkCli命令窗口中stat该节点确认，默认选主方式可能存在未知问题或触发Zookeeper问题。

建议修改选主方式，在**yarn-site**页签中，新增或更新配置项**yarn.resourcemanager.ha.curator-leader-elector.enabled**为true，然后重启RM。

- 如果还定位不到问题，请提交工单处理。

为什么节点启动任务时Localize失败或任务日志无法采集与删除？

- 问题现象：NM日志异常，提示java.io.IOException: Couldn't create proxy provider class org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider。
- 可能原因：HDFS配置异常。
- 处理方法：
  i. 以上是封装后的异常，非根因，需要打开DEBUG级别日志查看：
     - hadoop客户端命令行环境（例如执行 `hadoop fs -ls /` 命令），开启DEBUG调试信息。

       ```
       export HADOOP_LOGLEVEL=DEBUG
       ```

     - 有Log4j配置的运行环境，Log4j末尾增加 `log4j.logger.org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider=DEBUG` 。

  ii. 以下示例的根因是用户曾修改过NameServices配置，将emr-cluster修改为了hadoop-emr-cluster，但是扩容节点时使用了修改前的配置。

  ```
  21/12/01 18:16:44 DEBUG hdfs.NameNodeProxiesClient: Couldn't create proxy provider class org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider
  java.lang.reflect.InvocationTargetException
          at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
          at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:62)
          at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)
          at java.lang.reflect.Constructor.newInstance(Constructor.java:423)
          at org.apache.hadoop.hdfs.NameNodeProxiesClient.createFailoverProxyProvider(NameNodeProxiesClient.java:245)
          at org.apache.hadoop.hdfs.NameNodeProxiesClient.createFailoverProxyProvider(NameNodeProxiesClient.java:224)
          at org.apache.hadoop.hdfs.NameNodeProxiesClient.createProxyWithClientProtocol(NameNodeProxiesClient.java:134)
          at org.apache.hadoop.hdfs.DFSClient.<init>(DFSClient.java:356)
          at org.apache.hadoop.hdfs.DFSClient.<init>(DFSClient.java:290)
          at org.apache.hadoop.hdfs.DistributedFileSystem.initialize(DistributedFileSystem.java:171)
          at org.apache.hadoop.fs.FileSystem.createFileSystem(FileSystem.java:3303)
          at org.apache.hadoop.fs.FileSystem.access$200(FileSystem.java:124)
          at org.apache.hadoop.fs.FileSystem$Cache.getInternal(FileSystem.java:3352)
          at org.apache.hadoop.fs.FileSystem$Cache.get(FileSystem.java:3320)
          at org.apache.hadoop.fs.FileSystem.get(FileSystem.java:479)
          at org.apache.hadoop.fs.Path.getFileSystem(Path.java:365)
          at org.apache.hadoop.fs.shell.JfsLs.expandAsGlob(JfsLs.java:826)
          at org.apache.hadoop.fs.shell.JfsLs.access$000(JfsLs.java:45)
          at org.apache.hadoop.fs.shell.JfsLs$Ls.expandArgument(JfsLs.java:163)
          at org.apache.hadoop.fs.shell.JindoFsCommandShims.expandArguments(JindoFsCommandShims.java:151)
          at org.apache.hadoop.fs.shell.JindoFsCommandShims.processRawArguments(JindoFsCommandShims.java:56)
          at org.apache.hadoop.fs.shell.JindoFsCommandShims.run(JindoFsCommandShims.java:110)
          at org.apache.hadoop.fs.FsShell.run(FsShell.java:347)
          at org.apache.hadoop.util.ToolRunner.run(ToolRunner.java:76)
          at org.apache.hadoop.util.ToolRunner.run(ToolRunner.java:90)
          at org.apache.hadoop.fs.FsShell.main(FsShell.java:410)
  Caused by: java.lang.RuntimeException: Could not find any configured addresses for URI hdfs://hadoop-emr-cluster/apps/tez-0.9.2-hadoop3.2-1.2.5/tez-0.9.2-hadoop3.2-1.2.5.tar.gz
          at org.apache.hadoop.hdfs.server.namenode.ha.AbstractNNFailoverProxyProvider.getProxyAddresses(AbstractNNFailoverProxyProvider.java:174)
          at org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider.<init>(ConfiguredFailoverProxyProvider.java:51)
          at org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider.<init>(ConfiguredFailoverProxyProvider.java:45)
          ... 26 more
  ls: Couldn't create proxy provider class org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider
  [root@emr-worker-7 hadoop-conf]#
  ```

  iii. 在EMR控制台HDFS服务的配置页面，检查各项配置是否正确。

资源本地化异常，该如何处理？

- 问题现象：
  - Job AM Container启动失败，异常信息如下。

    ```
    Failed job diagnostics信息为：Application application_1412960082388_788293 failed 2 times due to AM Container for appatt
    empt_1412960082388_788293_000002 exited with exitCode: -1000 due to: EPERM: Operation not permitted
    ```

- 在资源本地化过程中（下载后解压时）报错，NodeManager日志信息如下。

```
INFO org.apache.hadoop.yarn.server.nodemanager.containermanager.localizer.ResourceLocalizationService: Failed to downlo
ad rsrc { { hdfs://hadoopnnvip.cm6:9000/user/heyuan.lhy/apv/small_apv_20141128.tar.gz, 1417144849604, ARCHIVE, null },p
ending,[(container_1412960082388_788293_01_000001)],14170282104675332,DOWNLOADING}
EPERM: Operation not permitted
        at org.apache.hadoop.io.nativeio.NativeIO$POSIX.chmodImpl(Native Method)
        at org.apache.hadoop.io.nativeio.NativeIO$POSIX.chmod(NativeIO.java:226)
        at org.apache.hadoop.fs.RawLocalFileSystem.setPermission(RawLocalFileSystem.java:629)
        at org.apache.hadoop.fs.DelegateToFileSystem.setPermission(DelegateToFileSystem.java:186)
        at org.apache.hadoop.fs.FilterFs.setPermission(FilterFs.java:235)
        at org.apache.hadoop.fs.FileContext$10.next(FileContext.java:949)
        at org.apache.hadoop.fs.FileContext$10.next(FileContext.java:945)
        at org.apache.hadoop.fs.FSLinkResolver.resolve(FSLinkResolver.java:90)
        at org.apache.hadoop.fs.FileContext.setPermission(FileContext.java:945)
        at org.apache.hadoop.yarn.util.FSDownload.changePermissions(FSDownload.java:398)
        at org.apache.hadoop.yarn.util.FSDownload.changePermissions(FSDownload.java:412)
        at org.apache.hadoop.yarn.util.FSDownload.changePermissions(FSDownload.java:412)
        at org.apache.hadoop.yarn.util.FSDownload.call(FSDownload.java:352)
        at org.apache.hadoop.yarn.util.FSDownload.call(FSDownload.java:57)
        at java.util.concurrent.FutureTask.run(FutureTask.java:262)
        at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:471)
        at java.util.concurrent.FutureTask.run(FutureTask.java:262)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1145)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615)
        at java.lang.Thread.run(Thread.java:744)
```

- 问题原因：压缩包内含有软链接导致资源本地化异常。
- 处理方法：删除该压缩包内包含的软链接。

Container启动失败或运行异常，报错提示No space left on device，该如何处理？

可能原因及排查思路如下：

- 磁盘已满。
- 检查*/sys/fs/cgroup/cpu/hadoop-yarn/cgroup.clone_children*和*/sys/fs/cgroup/cpu/cgroup.clone_children*的Cgroups配置。
  i. 如果cgroup.clone_children的值为0，请修改为1，开机启动项时，设置命令 `echo 1 > /sys/fs/cgroup/cpu/cgroup.clone_children` 。
  ii. 如果没有上述问题，请检查同级的*cpuset.mems*或*cpuset.cpus*文件，*hadoop-yarn*目录中的值需要和上层一样。
- 可能是Cgroups目录的子目录数超出上限65535造成的，检查YARN配置yarn.nodemanager.linux-container-executor.cgroups.delete-delay-ms或者yarn.nodemanager.linux-container-executor.cgroups.delete-timeout-ms。

节点NM服务或任务运行时无法正常解析域名，该如何处理？

- 问题现象：报错提示java.net.UnknownHostException: Invalid host name: local host is: (unknown)。
- 问题原因及处理方法：
  - 查看是否正确配置了DNS服务器。

    通过以下命令，检查配置信息。

    ```
    cat /etc/resolv.conf
    ```

  - 查看防火墙是否设置了53端口的相关规则。

    如果设置了，请关闭防火墙。

  - 查看是否开启了DNS的NSCD缓存服务。

    通过以下命令，检查服务状态。

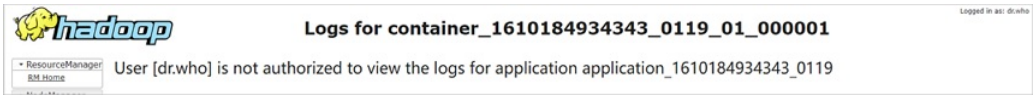
    ```
    systemctl status nscd
    ```

    如果开启了DNS的NSCD缓存服务，可以执行以下命令，停止缓存服务。

    ```
    systemctl stop nscd
    ```

报错提示User [dr.who] is not authorized to view the logs for application ***，该如何处理？

- 问题现象：打开日志页面时，提示信息如下。



- 问题原因：访问NodeManager Log页面时会检查ACL规则。如果启用了ACL规则，远程用户就要符合以下任一个条件：
  - 是admin用户。
  - 是app owner。
  - 满足app自定义ACL规则。
- 处理方法：检查是否满足以上条件中的任一条。

报错提示HTTP ERROR 401 Authentication required或HTTP ERROR 403 Unauthenticated users are not authorized to access this page，该如何处理？

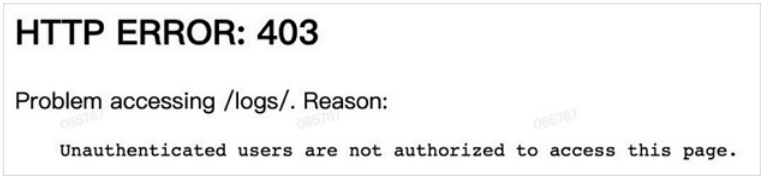
- 问题现象：访问UI或REST API时报错，详细信息如下图。

  HTTP ERROR 401

```
[root@emr-header-1 python_modules]# curl http://localhost:8088/ws/v1/cluster/info
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
<title>Error 401 Authentication required</title>
</head>
<body><h2>HTTP ERROR 401</h2>
<p>Problem accessing /ws/v1/cluster/info. Reason:
<pre>    Authentication required</pre></p><hr /><i><small>Powered by Jetty://</small></i><br/>

<hr/>
```

  HTTP ERROR 403



- 问题原因：YARN启用了Simple认证且不允许匿名访问，详情请参见Authentication for Hadoop HTTP web-consoles。
- 处理方法：
  - 方式一：URL参数中显示指定远程用户，例如，user.name=***。
  - 方式二：您可以在E-MapReduce控制台的HDFS服务的**配置**页签，在搜索区域搜索参数hadoop.http.authentication.simple.anonymous.allowed，修改其参数值为true允许匿名访问，参数含义请参见Authentication for Hadoop HTTP web-consoles。然后重启服务，请参见重启服务。

为什么TotalVcore显示值不准确？

在YARN UI右上方，Cluster或Metrics REST API结果中，TotalVcore显示值不准确。此问题是由于社区2.9.2之前版本TotalVcore计算逻辑有问题，详情请参见https://issues.apache.org/jira/browse/YARN-8443。

EMR-3.37.x和EMR-5.3.x及后续版本已修复此问题。

TEZ UI展示的应用信息不全，该如何处理？

您可以打开浏览器的**开发者工具**，排查并处理遇到的问题：

1. 如果是路径为*http://<rmAddress>/ws/v1/cluster/apps/APPID*的异常，则可能原因是该应用已经被RM清理出去（YARN RM最多保留一定数量的历史应用信息，默认1000个，超出的会将历史应用按启动顺序清理出去）。
2. 如果是路径为*http://<tsAddress>/ws/v1/applicationhistory/...*的异常，直接访问该URL（http://<tsAddress>/ws/v1/applicationhistory/...）返回500异常（提示找不到app），则可能原因是该应用信息未成功存储（RM发起，查看yarn.resourcemanager.system-metrics-publisher.enabled配置项），或已被Timeline Store清理（查看LevelDB TTL相关配置）。
3. 如果是路径为*http://<tsAddress>/ws/v1/timeline/...*的异常，直接访问该URL（http://<tsAddress>/ws/v1/timeline/...）返回正常（code为200），但是内容是NotFound信息。
   - 查看AM日志（syslog）启动时打印信息，正常初始化信息如下：

```
 [INFO] [main] |history.HistoryEventHandler|: Initializing HistoryEventHandler withrecoveryEnabled=true, historyServiceClassName=org.apache.tez.dag.history.logging.ats.ATSHistoryLoggingService
 [INFO] [main] |ats.ATSHistoryLoggingService|: Initializing ATSHistoryLoggingService with maxEventsPerBatch=5, maxPollingTime(ms)=10, waitTimeForShutdown(ms)=-1, TimelineACLManagerClass=org.apache.tez.dag.history.ats.acls.ATSHistoryACLPolicyManager
```

   - 如果有以下异常信息，说明AM运行时yarn.timeline-service.enabled配置异常，可能原因为FlowAgent问题（FlowAgent里对Hive作业有2种实现，一种是Hive命令，另外一种是Beeline命令，此时默认配置yarn.timeline-service.enabled为false。）

```
[WARN] [main] |ats.ATSHistoryLoggingService|: org.apache.tez.dag.history.logging.ats.ATSHistoryLoggingService is disa
bled due to Timeline Service being disabled, yarn.timeline-service.enabled set to false
```

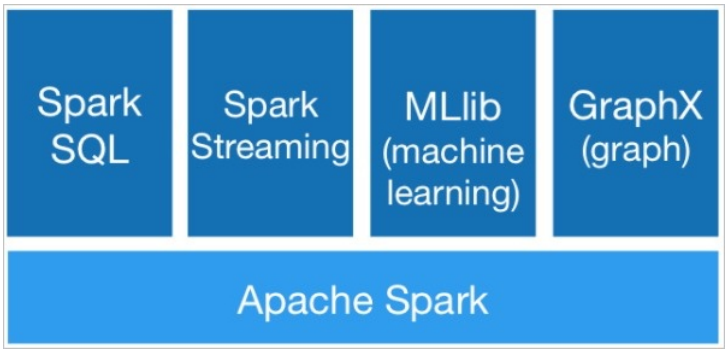
6.2.22. Spark

6.2.22.1. 概述

Spark是一个通用的大数据分析引擎，具有高性能、易用和普遍性等特点。E-MapReduce（简称EMR）支持Spark 1.x和Spark 2.x两个版本。

背景信息

Spark架构如下图所示，基于Spark Core构建了Spark SQL、Spark Streaming、MLlib和Graphx四个主要编程库，分别用于离线ETL（Extract-Transform-Load）、在线数据分析、流计算、机器学习和图计算等场景，详情请参见Apache Spark官网。



使用场景

- 离线ETL

  离线ETL主要应用于数据仓库，对大规模的数据进行抽取（Extract）、转换（Transform）和加载（Load），其特点是数据量大，耗时较长，通常设置为定时任务执行。

- 在线数据分析（OLAP）

  在线数据分析主要应用于BI（Business Intelligence）。分析人员交互式地提交查询作业，Spark可以快速地返回结果。除了Spark，常见的OLAP引擎包括Presto和Impala等。Spark 3.0的主要特性在EMR中的Spark 2.4版本已支持，更多特性详情请参见Spark SQL Guide。

- 流计算

  流计算主要应用于实时大屏、实时风控、实时推荐和实时报警监控等。流计算主要包括Spark Streaming和Flink引擎，Spark Streaming提供DStream和Structured Streaming两种接口，Structured Streaming和Dataframe用法类似，门槛较低。Flink适合低延迟场景，而Spark Streaming更适合高吞吐的场景，详情请参见Structured Streaming Programming Guide。

- 机器学习

  Spark的MLlib提供了较丰富的机器学习库，包括分类、回归、协同过滤、聚合，同时提供了模型选择、自动调参和交叉验证等工具来提高生产力。MLlib主要支持非深度学习的算法模块，详情请参见Machine Learning Library (MLlib) Guide。

- 图计算

  Spark的GraphX支持图计算的库，支持丰富的图计算的算子，包括属性算子、结构算子、Join算子和邻居聚合等。详情请参见GraphX Programming Guide。

入门指导

- 开发Spark作业，详情请参见Spark作业配置、Spark SQL作业配置、Spark Shell作业配置和Spark Streaming作业配置。
- Spark作业开发指南，详情请参见Spark开发指南或Spark Streaming SQL开发指南。
- Spark常用文件的路径，详情请参见常用文件路径。
- 在Zeppelin中使用Spark，详情请参见Zeppelin概述。

6.2.22.2. EMR Spark功能增强

阿里云E-MapReduce产品构建于阿里云云服务器ECS上，基于开源的Apache Hadoop和Apache Spark，做了大量优化。本文为您介绍E-MapReduce（简称EMR）Spark相对开源增强的功能。

背景信息

阿里云EMR 100%采用社区开源组件，随开源版本升级迭代，基于开源组件，优化和增强阿里云部署环境。

功能增强

Spark针对开源功能增强的功能如下表。

EMR版本	组件版本	功能增强
EMR-5.2.1	Spark 3.1.1	• 支持数据湖格式Delta Lake和Hudi。 • 支持Remote Shuffle Service。 • 支持Livy 。 • 优化E-MapReduce控制台上，Spark服务**配置**页面的**spark-defaults**页签的配置项名称。 • 优化CBO（Cost-Based Optimization）、DPP（DynamicPartitionPruning）以及Z-Order等功能，性能比开源Spark 3版本提升50%。 • 支持阿里云Log Service、DataHub和消息队列RocketMQ版（简称ONS）等数据源。
EMR-4.9.0	Spark 2.4.7	• 修复Adaptive Execution部分场景无法生效的问题。 • 修复统计聚合函数行为和Hive不一致的问题。 • 修复读取Hive ORC表char类型数据正确性的问题。
EMR-4.8.0	Spark 2.4.7	• 优化了部分默认配置。 • 性能优化：支持Window TopK下推。 • 增强Hive读写CSV或JSON表的兼容性。 • ANALYZE语句支持省略全表列名。 • 支持一键开启或关闭LDAP功能。 开启或关闭LDAP功能详情，请参见管理LDAP认证。 • 改进Spark Beeline工具的易用性。
EMR-4.6.0	Spark 2.4.7	• 升级至2.4.7版本。 • 升级jQuery至3.5.1版本。 • 兼容Hive方式自动更新表和分区大小。 • 支持Spark元数据和作业运行信息输出至DataWorks。
EMR-4.5.0	Spark 2.4.5	支持数据湖构建（DLF）元数据。
EMR-4.4.0	Spark 2.4.5	• ThriftServer进程增加Java Agent。 • 优化Spark SQL的NOT IN子查询性能。 • 修复读取大量空文件时栈溢的出问题。 • 通过代码去除对HAS显式依赖或升级HAS依赖。
EMR-4.3.0	Spark 2.4.5	• 升级至2.4.5版本。 • 升级关联的Delta Lake至0.6.0版本。 • 修复开启Ranger Hive后，Pyspark无法正常运行的缺陷。
EMR-3.36.1	Spark 2.4.7	• 优化E-MapReduce控制台上，Spark服务**配置**页面的**spark-defaults**页签的配置项名称。 • 优化输出日志性能。 • 支持ZSTD（Zstandard）压缩格式。
EMR-3.35.0	Spark 2.4.7	• 修复Adaptive Execution部分场景无法生效的问题。 • 修复统计聚合函数行为和Hive不一致的问题。 • 修复读取Hive ORC表char类型数据正确性的问题。
EMR-3.34.0	Spark 2.4.7	• 优化了部分默认配置。 • 性能优化：支持Window TopK下推。 • 增强Hive读写CSV或JSON表的兼容性。 • ANALYZE语句支持省略全表列名。 • 支持一键开启或关闭LDAP功能。 • 改进Spark Beeline工具的易用性。
EMR-3.33.0	Spark 2.4.7	• 升级至2.4.7版本。 • 升级jQuery至3.5.1版本。 • 兼容Hive方式，自动更新表和分区大小。 • 支持Spark元数据和作业运行信息输出至DataWorks。

EMR版本	组件版本	功能增强
EMR-3.32.0	Spark 2.4.5	JindoTable支持打开或关闭数据采集功能。
EMR-3.30.0	Spark 2.4.5	• 支持阿里云DLF（Data Lake Formation）元数据。 • 升级HAS依赖至2.0.1。 • 修复Streaming SQL反引号问题。 • 移除Delta的JAR包，修改为Delta单独部署。 • 修改日志路径统一写至HDFS下。
EMR-3.29.0	Spark 2.4.5	• Spark升级至2.4.5.2.0。 • 支持第三方Metastore的功能。 • 增加datalake metastore-client。
EMR-3.28.0	Spark 2.4.5	• 升级至2.4.5版本。 • 兼容DataFactory的streaming-sql脚本。 • 支持Delta 0.6.0版本。
EMR-3.27.2	Spark 2.4.3	• CUBE中支持日期类型分区字段。 • 调大Spark-Submit的stack深度。
EMR-3.26.3	Spark 2.4.3	配置默认的committer为JindoOssCommitter。
EMR-3.25.0	Spark 2.4.3	• 支持在控制台配置 `spark.sql.extensions` 等Delta相关参数。 • 支持Hive读取Delta table，避免set inputformat。 • 支持ALTER TABLE SET TBLPROPERTIES和UNSET TBLPROPERTIES语句。
EMR-3.24.0	Spark 2.4.3	• 增加Delta相关参数支持。 • 增加对Ranger spark plugin配置的支持。 • JindoCube升级到0.3.0版本。
EMR-3.23.0	Spark 2.4.3	• 更新spark thriftserver，解决class loader问题。 • 重构spark事务相关代码，提升稳定性。 • 解决升builtin hive至2.3版本后orc格式读写问题。 • 支持merge into语法。 • 支持scan和stream语法。 • Structured Streaming Kafka sink支持EOS。 • delta更新至0.4.0。

EMR版本	组件版本	功能增强
EMR-3.22.1版本	Spark 2.4.3	• Relational Cache 支持Relational Cache，Relational Cache通过预计算加速用户查询。用户可以创建Relational Cache对数据进行预计算，在执行用户查询时，Spark Optimizer自动发现合适的Cache，并改写SQL执行计划，基于Cache的数据继续计算，从而提升查询速度，适用于报表、Dashboard、数据同步和多维分析等场景。 ○ 通过DDL，进行CACHE、UNCACHE、ALTER、SHOW等操作，Cache的数据支持Spark的所有数据源和数据格式。 ○ 支持自动的Cache数据更新以及通过REFRESH命令更新Cache数据，支持基于分区的增量更新。 ○ 支持基于Relational Cache的执行计划优化。 • Streaming SQL ○ 规范Stream Query Writer的参数配置。 ○ 优化Kafka数据表Schema兼容性检查。 ○ Kafka数据表Schema不存在时自动创建到SchemaRegistry。 ○ 优化Kafka Schema不兼容时的日志信息。 ○ 修复查询结果写Kafka表时必须显式指定列名的问题。 ○ 去掉流式SQL查询只支持Kafka和Loghub数据输入源的限制。 • Delta 新增Delta，用户可使用Spark创建Delta datasource，以支持流式数据写入、事务性读写、数据校验和数据回溯等应用场景。详情请参见Delta详细信息。 ○ 支持使用DataFrame API从Delta读取数据或者写入数据到Delta。 ○ 支持使用Stuctured Streaming API以Delta作为source或者sink进行数据的读或写。 ○ 支持使用Delta API对数据进行update、delete、merge、vaccum、optimize等操作。 ○ 支持使用SQL创建基于Delta的表、导入数据到Delta和读取Delta表等操作。 • Others ○ constraint feature，支持主键和外键。 ○ 解决servlet等jar冲突问题。

6.2.22.3. 基础使用

6.2.22.3.1. Spark Shell和RDD基础操作

本文为您介绍如何使用Spark Shell，以及RDD的基础操作。

前提条件

已创建集群，详情请参见创建集群。

启动Spark Shell

Spark的Shell作为一个强大的交互式数据分析工具，提供了一个简单的方式学习API。 Spark既可以使用Scala，也可以使用Python。

您可以按照以下操作步骤来启动Spark Shell。

1. 通过SSH方式连接集群，详情请参见登录集群。
2. 执行以下命令，进入Spark的安装目录。

   ```
   cd /usr/lib/spark-current
   ```

3. 执行以下命令，启动Spark Shell。

   ```
   ./bin/spark-shell
   ```

   在Spark Shell中，已经在名为sc的变量中为您创建了一个特殊的SparkContext，如果创建您自己的SparkContext将不生效。您可以使用 `--master` 参数设置SparkContext连接到哪个主节点，并且可以通过 `--jars` 参数来设置添加到CLASSPATH的JAR包，多个JAR包时使用逗号（,）分隔。更多参数信息，您可以通过命令 `./bin/spark-shell --help` 获取。

RDD基础操作

Spark围绕着弹性分布式数据集（RDD）的概念展开，RDD是可以并行操作的元素的容错集合。Spark支持通过集合来创建RDD和通过外部数据集构建RDD两种方式来创建RDD。例如，共享文件系统、HDFS、HBase或任何提供Hadoop InputFormat的数据集。

创建RDD示例：

- 通过集合来创建RDD

```
val data = Array(1, 2, 3, 4, 5)
val distData = sc.parallelize(data)
```

- 通过外部数据集构建RDD

```
val distFile = sc.textFile("data.txt")
```

RDD构建成功后，您可以对其进行一系列操作，例如Map和Reduce等操作。

例如，运行以下代码，首先从外部存储系统读一个文本文件构造了一个RDD，然后通过RDD的Map算子计算得到了文本文件中每一行的长度，最后通过Reduce算子计算得到了文本文件中各行长度之和。

```
val lines = sc.textFile("data.txt")
val lineLengths = lines.map(s => s.length)
val totalLength = lineLengths.reduce((a, b) => a + b)
```

通常，Spark RDD的常用操作有两种，分别为Transform操作和Action操作。Transform操作并不会立即执行，而是到了Action操作才会被执行。

- Transform操作

操作	描述
map()	参数是函数，函数应用于RDD每一个元素，返回值是新的RDD。
flatMap()	参数是函数，函数应用于RDD每一个元素，将元素数据进行拆分，变成迭代器，返回值是新的RDD。
filter()	参数是函数，函数会过滤掉不符合条件的元素，返回值是新的RDD。
distinct()	没有参数，将RDD里的元素进行去重操作。
union()	参数是RDD，生成包含两个RDD所有元素的新RDD。
intersection()	参数是RDD，求出两个RDD的共同元素。
subtract()	参数是RDD，将原RDD里和参数RDD里相同的元素去掉。
cartesian()	参数是RDD，求两个RDD的笛卡尔积。

- Action操作

操作	描述
collect()	返回RDD所有元素。
count()	返回RDD中的元素个数。
countByValue()	返回各元素在RDD中出现的次数。
reduce()	并行整合所有RDD数据，例如求和操作。
fold(0)(func)	和reduce()功能一样，但是fold带有初始值。
aggregate(0)(seqOp,combop)	和reduce()功能一样，但是返回的RDD数据类型和原RDD不一样。
foreach(func)	对RDD每个元素都是使用特定函数。

6.2.22.3.2. Spark SQL、Dataset和DataFrame基础操作

本文为您介绍Spark SQL、Dataset和DataFrame相关的概念，以及Spark SQL的基础操作。

Spark SQL、Dataset和DataFrame介绍

Spark SQL是一个用于结构化数据处理的Spark模块，与基本的Spark RDD的API不同，Spark SQL的接口还提供了更多关于数据和计算的结构化信息。Spark SQL可以用于执行SQL查询并从Hive表中读取数据。

Dataset是数据的分布式集合。Dataset是Spark 1.6中添加的一个新接口，它集成了RDD和Spark SQL的优点，可以从JVM对象构造数据集，然后使用函数转换（Map、FlatMap或Filter等）进行操作。Dataset API有Scala和Java两种版本。Python和R不支持Dataset API，但是由于Python和R的动态特性，Dataset API的许多优点已经可用。

DataFrame是组织成命名列的Dataset。他在概念上相当于关系数据库中的一个表，或R和Python中的一个DataFrame，但是进行了更丰富的优化。DataFrame可以从一系列广泛的源构建，例如：结构化数据文件、Hive中的表、外部数据库或现有RDD。DataFrame API有Scala、Java、Python和R版本。在Scala和Java中，DataFrame由行数据集表示。在Scala API中，DataFrame只是Dataset[Row]的类型别名，而在Java API中，您需要使用Dataset<Row>来表示数据帧。

Spark SQL基础操作

Spark SQL支持直接通过SQL语句操作数据，而Spark会将SQL进行解析、优化并执行。

以下示例展示了如何使用Spark SQL进行读取文件。示例如下：

- 示例1：Spark支持多种数据格式，本示例读取了JSON格式文件的数据，并输出为Parquet格式。

```
val peopleDF = spark.read.json("examples/src/main/resources/people.json")
peopleDF.write.parquet("people.parquet")
```

- 示例2：通过SQL从parquetFile表中读出年龄在13岁到19岁之间的年轻人的名字，并转化为DataFrame，随后通过Map操作将名字转化为一个可读的形式并输出。

```
val namesDF = spark.sql("SELECT name FROM parquetFile WHERE age BETWEEN 13 AND 19")
namesDF.map(attributes => "Name: " + attributes(0)).show()
```

6.2.22.3.3. PySpark基础操作

PySpark是Spark提供的Python API。您可以通过PySpark提供的DataFrame接口，完成各种计算逻辑。本文为您介绍PySpark的基础操作。

前提条件

已创建集群，详情请参见创建集群。

操作步骤

1. 初始化SparkSession。

   初始化SparkSession作为PySpark的执行入口。

   ```
   from pyspark.sql import SparkSession
   spark = SparkSession.builder.getOrCreate()
   ```

2. 创建DataFrame。

   ```
   from datetime import datetime, date
   import pandas as pd
   from pyspark.sql import Row
   df = spark.createDataFrame([
       (1, 2., 'string1', date(2000, 1, 1), datetime(2000, 1, 1, 12, 0)),
       (2, 3., 'string2', date(2000, 2, 1), datetime(2000, 1, 2, 12, 0)),
       (3, 4., 'string3', date(2000, 3, 1), datetime(2000, 1, 3, 12, 0))
   ], schema='a long, b double, c string, d date, e timestamp')
   ```

   DataFrame创建完成后，您就可以通过各种类型的transform算子完成数据计算了。

3. 打印DataFrame和Schema。

   ```
   df.show()
   df.printSchema()
   ```

6.2.22.4. 高阶使用

6.2.22.4.1. 管理LDAP认证

服务开启LDAP认证功能后，访问服务时需要提供LDAP身份认证（LDAP用户名和密码），以便于提升服务的安全性。开启LDAP功能对接的LDAP为E-MapReduce自带的OpenLDAP。开启LDAP认证的功能可以方便您使用LDAP认证，避免了复杂的配置过程。本文为您介绍如何一键开启和关闭LDAP认证。

前提条件

已创建Hadoop集群，详情请参见创建集群。

使用限制

EMR-3.34.0及后续版本或EMR-4.8.0及后续版本的Hadoop集群，支持一键开启LDAP认证。

开启LDAP认证

注意 Spark开启LDAP认证后，Hue访问Spark需要进行额外的配置，详情请参见Hue连接开启LDAP认证的引擎。

1. 进入Spark页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务 > Spark**。
2. 开启LDAP认证。
   i. 在Spark服务页面，选择右上角的**操作 > 开启LDAP认证**。
   ii. 在**执行集群操作**对话中，单击**确认**。
3. 单击上方的**查看操作历史**。
   直至操作状态显示**成功**。
4. 重启Spark ThriftServer 。
   i. 在Spark服务页面，选择右上角的**操作 > 重启ThriftServer**。
   ii. 在**执行集群操作**对话中，输入**执行原因**，单击**确定**。
   iii. 在**确认**对话中，单击**确定**。

通过命令方式访问ThriftServer

开启LDAP认证后，访问ThriftServer需要提供LDAP认证凭据。

1. 通过SSH方式连接集群，详情请参见登录集群。
2. 通过命令方式访问ThriftServer。
   - Beeline客户端：
     ```
     /usr/lib/spark-current/bin/beeline -u jdbc:hive2://emr-header-1:10001 -n <user> -p <password>
     ```
   - JDBC：
     ```
     jdbc:hive2://emr-header-1:10001/default;user=<user>;password=<password>
     ```

     说明 user为LDAP的用户名，password为LDAP的密码。开启LDAP认证后，访问Spark ThriftServer需要提供LDAP的用户名和密码，获取方式请参见管理用户。

关闭LDAP认证

1. 进入Spark页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务 > Spark**。
2. 关闭LDAP认证。
   i. 在Spark服务页面，选择右上角的**操作 > 关闭LDAP认证**。
   ii. 在**执行集群操作**对话中，单击**确认**。
3. 单击上方的**查看操作历史**。
   直至操作状态显示**成功**。
4. 重启Spark ThriftServer 。
   i. 在Spark服务页面，选择右上角的**操作 > 重启ThriftServer**。
   ii. 在**执行集群操作**对话中，输入**执行原因**，单击**确定**。
   iii. 在**确认**对话中，单击**确定**。

6.2.22.5. 最佳实践

6.2.22.5.1. Spark处理Delta Lake和Hudi数据

Delta Lake和Hudi是当前主流的数据湖产品，并且都支持了Spark的读写操作。本文为您介绍Spark如何处理Delta Lake和Hudi数据。

背景信息

通过以下方面介绍Spark处理Delta Lake和Hudi数据的操作：

- 准备工作
- 写操作
- 读操作
- 更新操作

准备工作

需要在项目中引入Delta Lake或Hudi相关的pom依赖。

例如，Delta Lake为1.0.0版本，Hudi为0.9.0版本，Scala使用2.12版本。使用Maven构建项目，Delta Lake或Hudi需要添加以下依赖：

- Delta Lake依赖

```
<dependency>
  <groupId>io.delta</groupId>
  <artifactId>delta-core_2.12</artifactId>
  <version>1.0.0</version>
</dependency>
```

- Hudi依赖

```
<dependency>
    <groupId>org.apache.hudi</groupId>
    <artifactId>hudi-spark_2.12</artifactId>
    <version>0.9.0</version>
</dependency>
```

写操作

Delta Lake或Hudi写操作示例如下：

- Delta Lake

```
val data = spark.range(0, 5)
data.write.format("delta").save("/tmp/delta-table")
```

- Hudi

```
val tableName = "hudi_trips_cow"
val basePath = "file:///tmp/hudi_trips_cow"
val dataGen = new DataGenerator
val inserts = convertToStringList(dataGen.generateInserts(10))
val df = spark.read.json(spark.sparkContext.parallelize(inserts, 2))
df.write.format("hudi").
  options(getQuickstartWriteConfigs).
  option(PRECOMBINE_FIELD_OPT_KEY, "ts").
  option(RECORDKEY_FIELD_OPT_KEY, "uuid").
  option(PARTITIONPATH_FIELD_OPT_KEY, "partitionpath").
  option(TABLE_NAME, tableName).
  mode(Overwrite).
  save(basePath)
```

读操作

Delta Lake或Hudi读操作示例如下：

- Delta Lake

```
val df = spark.read.format("delta").load("/tmp/delta-table")
df.show()
```

- Hudi

```
val tripsSnapshotDF = spark.
  read.
  format("hudi").
  load(basePath)
tripsSnapshotDF.show()
```

更新操作

Delta Lake或Hudi更新操作示例如下：

- Delta Lake

```
val data = spark.range(5, 10)
data.write.format("delta").mode("overwrite").save("/tmp/delta-table")
df.show()
```

- Hudi

```
val tableName = "hudi_trips_cow"
val basePath = "file:///tmp/hudi_trips_cow"
val dataGen = new DataGenerator
val updates = convertToStringList(dataGen.generateUpdates(10))
val df = spark.read.json(spark.sparkContext.parallelize(updates, 2))
df.write.format("hudi").
  options(getQuickstartWriteConfigs).
  option(PRECOMBINE_FIELD_OPT_KEY, "ts").
  option(RECORDKEY_FIELD_OPT_KEY, "uuid").
  option(PARTITIONPATH_FIELD_OPT_KEY, "partitionpath").
  option(TABLE_NAME, tableName).
  mode(Append).
  save(basePath)
```

6.2.22.6. 常见问题

本文汇总了Spark使用时的常见问题。

如何指定PySpark使用Python 3版本？

E-MapReduce中的PySpark默认为Python 2版本，本文以EMR-3.35版本为例，为您介绍如何指定PySpark使用Python 3版本。

您可以通过以下两种方式修改Python的版本：

- 临时生效方式
  i. 通过SSH方式登录集群，详情请参见登录集群。
  ii. 执行以下命令，修改Python的版本。

```
export PYSPARK_PYTHON=/usr/bin/python3
```

  iii. 执行以下命令，查看Python的版本。

```
pyspark
```

  当返回信息中包含如下信息时，表示已修改Python版本为Python 3。

```
Using Python version 3.6.8 (default, Apr 20 2020 14:49:33)
```

- 永久生效方式

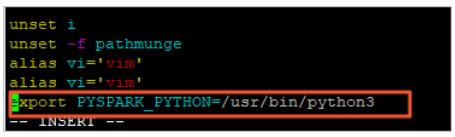
> 注意 此方式的修改是全局性的，可能会导致集群出现异常，因此请谨慎操作。

  i. 通过SSH方式登录集群，详情请参见登录集群。
  ii. 修改配置文件。
    a. 执行以下命令，打开文件*profile*。

```
vi /etc/profile
```

    b. 按下 `i` 键进入编辑模式。
    c. 在*profile*文件末尾添加以下信息，以修改Python的版本。

```
export PYSPARK_PYTHON=/usr/bin/python3
```



    d. 按下 `Esc` 键退出编辑模式，输入 `:wq` 保存并关闭文件。
  iii. 执行以下命令，重新执行刚修改的配置文件，使之立即生效。

```
source /etc/profile
```

iv. 执行以下命令，查看Python的版本。

```
pyspark
```

当返回信息中包含如下信息时，表示已修改Python版本为Python 3。

```
Using Python version 3.6.8 (default, Apr 20 2020 14:49:33)
```

6.2.23. Hive

6.2.23.1. Hive概述

Hive是一个基于Hadoop的数据仓库框架，在大数据业务场景中，主要用来进行数据提取、转化和加载（ETL）以及元数据管理。

背景信息

E-MapReduce（简称EMR）版本中，Hadoop、Hive版本和EMR集群的配套情况，请参见版本概述。

Hive结构

名称	说明
HiveServer2	HiveQL查询服务器，可以配置为Thrift或者HTTP协议，接收来自JDBC客户端提交的SQL请求，支持多客户端并发以及身份验证。
Hive MetaStore	元数据管理模块，此模块被其他引擎所依赖，用于存储Database和Table等元信息。例如，Spark和Presto均依赖此模块作为其元数据管理。
Hive Client	Hive客户端，直接利用该客户端提交SQL作业，根据其设置运行引擎配置，可以将SQL转换成MR作业、Tez作业和Spark作业，该模块在所有EMR节点上均有安装。

Hive语法

EMR产品最大程度的保持了开源社区的语法以及体验，在Hive语法上保持与开源社区Hive语法100%的兼容性。

关于Apache Hive的更多介绍，请参见Apache Hive官网。

入门指导

- Hive的使用请参见Hive作业配置。
- 在Zeppelin中使用Hive的详情请参见Zeppelin概述。

6.2.23.2. EMR Hive功能增强

本文为您介绍E-MapReduce（简称EMR）各版本对应的Hive组件版本，以及各版本中Hive相对开源增强的功能。

Hive针对开源功能增强的功能如下表。

EMR版本	组件版本	功能增强
EMR-5.2.1	Hive 3.1.2	• 修复使用DLF元数据执行 `show create table` 命令，结果显示不正确的问题。 • 优化Hive默认参数，以提升作业性能。 • 修改E-MapReduce控制台上，Hive服务**配置**页面的**hive-env**页签的配置项名称为大写，便于用户使用。 • 修复UDF（User Define Function）导致HiveServer2内存泄露的问题。 • 优化文件系统与MetaStore不一致时写Hive表的报错信息。
EMR-4.8.0	Hive 3.1.2	• 优化了部分默认配置。 • 性能优化：增强CBO。 • 支持一键开启或关闭LDAP功能。 开启或关闭LDAP功能详情，请参见管理LDAP认证。
EMR-4.6.0	Hive 3.1.2	• HCatalog支持Data Lake Formation。 • 支持Hive元数据和作业运行信息输出至DataWorks。

EMR版本	组件版本	功能增强
EMR-4.5.0	Hive 3.1.2	• 支持数据湖构建（DLF）元数据。 • 支持Ranger Ownership权限。
EMR-4.4.1	Hive 3.1.2	优化默认的参数配置。
EMR-4.4.0	Hive 3.1.2	• 升级至3.1.2版本。 • 优化JindoFS。 • 优化MSCK。 • HCatalog支持JindoCommitter。 • 升级HAS依赖。
EMR-4.3.0	Hive 3.1.1	支持自定义部署。
EMR-3.36.1	Hive 2.3.8	• 升级Hive至2.3.8版本。 • 修复使用DLF（DataLakeFormation）元数据执行 `show create table` 命令时，结果显示不正确的问题。 • 优化Hive默认参数，以提升作业性能。 • 修改E-MapReduce控制台上，Hive服务**配置**页面的**hive-env**页签的配置项名称为大写，便于用户使用。 • 优化文件系统与MetaStore不一致时写Hive表的报错信息。
EMR-3.35.0	Hive 2.3.7	修复Fetch Task相关的社区问题。
EMR-3.34.0	Hive 2.3.7	• 优化了部分默认配置。 • 性能优化：增强CBO。 • 支持一键开启或关闭LDAP功能。 开启或关闭LDAP功能详情，请参见管理LDAP认证。 • 升级Calcite版本至1.12.0。 • 增加参数**hive.security.authorization.sqlstd.confwhitelist.append**。
EMR-3.33.0	Hive 2.3.7	• 升级至2.3.7版本。 • HCatalog支持Data Lake Formation。 • 支持Hive元数据和作业运行信息输出至DataWorks。
EMR-3.32.0	Hive 2.3.5	• 修复了HiveServer连接池泄漏的问题。 • JindoTable支持打开或关闭数据采集功能。 • 优化 `ADD COLUMN` 的性能。 • 修复了读取HUDI表时数据不正确的问题。 • 默认的参数配置，可以根据集群节点大小调整。
EMR-3.30.0	Hive 2.3.5	• 支持阿里云DLF（Data Lake Formation）元数据。 • 解决了读Delta表空目录时写DUMMY文件问题。 • 升级HAS依赖至2.0.1。
EMR-3.29.0	Hive 2.3.5	• Hive升级至2.3.5.6.0。 • 支持第三方Metastore的功能。 • 增加datalake metastore-client。
EMR-3.28.0	Hive 2.3.5	支持Delta 0.6.0版本。
EMR-3.27.2	Hive 2.3.5	• hcatalog表支持magic committer。 • 移除一些过时的默认配置。
EMR-3.26.3	Hive 2.3.5	hcatalog表支持direct committer。
EMR-3.25.0	Hive 2.3.5	修复自动LOCAL模式下MR任务执行失败的问题

EMR版本	组件版本	功能增强
EMR-3.24.0	Hive 2.3.5	• 增加SQL兼容性检查功能逻辑。 • Hive2.3.5+Hadoop2.8.5组合发布。 • 重启组件时不同步*hiveserver2-site.xml*中的内容至spark-conf下的*hive-site.xml*。 • 支持使用MSCK命令添加增量目录。 • 修复Hive复用tez container时出现的bug。 • 支持使用MSCK命令优化列目录。
EMR-3.23.0	Hive 2.3.5	• 删除老版本的hive hook。 • 添加支持多个count distinct字段的数据倾斜处理优化。 • 解决join不同bucketversion的表时丢数据的问题。
EMR-3.23.0之前版本	Hive 2.x	外部统一数据库保存至Hive Meta，所有使用外部Hive Meta的集群共享同一份Meta信息。

6.2.23.3. 基础使用

6.2.23.3.1. Hive基础操作

本文介绍如何通过Hive在E-MapReduce集群上创建库和表等操作。

前提条件

已创建集群，详情请参见创建集群。

进入Hive命令行

1. 使用SSH方式登录到集群主节点，详情请参见登录集群。
2. 执行以下命令，切换为hadoop用户。

   ```
   su hadoop
   ```

3. 执行以下命令，进入Hive命令行。

   ```
   hive
   ```

   返回信息如下所示时，表示进入Hive命令行。

   ```
   Logging initialized using configuration in file:/etc/ecm/hive-conf-2.3.5-2.0.3/hive-log4j2.properties Async: true
   Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different executio
   n engine (i.e. spark, tez) or using Hive 1.X releases.
   ```

库操作

本文示例中的数据库以testdb为例介绍。

- 创建库

  ```
  create database if not exists testdb;
  ```

  当返回信息包含OK时，表示创建库testdb成功。

- 查看库

  ```
  desc database testdb;
  ```

  查看信息如下所示。

  ```
  hive> desc database testdb;
  OK
  testdb          hdfs://emr-header-1.cluster-2[redacted]:9000/user/hive/warehouse/test
  db.db   hadoop  USER
  Time taken: 0.201 seconds, Fetched: 1 row(s)
  ```

- 使用数据库

  ```
  use testdb;
  ```

  返回信息如下所示。

  ```
  hive> use testdb;
  OK
  Time taken: 0.092 seconds
  ```

- 删除库

```
drop database if exists testdb;
```

当返回信息包含OK时，表示删除库成功。

表操作

本文示例中的表以t为例介绍。

- 创建表

```
create table if not exists t (id bigint, value string);
```

当返回信息包含OK时，表示创建表t成功。

- 查看表信息

```
desc formatted t;
```

查看信息如下所示。

```
hive> desc formatted t;
OK
# col_name              data_type               comment

id                      bigint
value                   string

# Detailed Table Information
Database:               test
Owner:                  hadoop
CreateTime:             Thu Jan 14 17:27:13 CST 2021
LastAccessTime:         UNKNOWN
Retention:              0
Location:               hdfs://emr-header-1.cluster-[redacted]:9000/user/hive/warehouse/test.db/t
Table Type:             MANAGED_TABLE
Table Parameters:
        COLUMN_STATS_ACCURATE   {\"BASIC_STATS\":\"true\"}
        numFiles                0
        numRows                 0
        rawDataSize             0
        totalSize               0
        transient_lastDdlTime   1610616433

# Storage Information
SerDe Library:          org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
InputFormat:            org.apache.hadoop.mapred.TextInputFormat
OutputFormat:           org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
Compressed:             No
Num Buckets:            -1
Bucket Columns:         []
Sort Columns:           []
Storage Desc Params:
        serialization.format    1
Time taken: 0.173 seconds, Fetched: 31 row(s)
```

- 查看所有表

```
show tables;
```

返回信息如下所示。

```
OK
t
```

- 删除表

```
drop table if exists t;
```

当返回信息包含OK时，表示删除表成功。

SQL操作

- 插入记录

```
insert into table t select 1, 'value-1';
```

当返回信息包含OK时，表示插入信息成功。

```
OK
Time taken: 14.73 seconds
```

- 查询表中的前10条信息

```
select * from t limit 10;
```

返回信息如下所示。

```
OK
1       value-1
Time taken: 11.48 seconds, Fetched: 1 row(s)
```

- 聚合操作

```
select value, count(id) from t group by value;
```

返回信息如下所示。

```
OK
value-1 1
Time taken: 20.11 seconds, Fetched: 1 row(s)
```

6.2.23.3.2. Hive连接方式

本文为您介绍在E-MapReduce集群提交Hive SQL的两种方式。

前提条件

已登录集群，详情请参见登录集群。

方式一：通过Hive客户端

- 普通集群，提交方式如下所示。

```
hive
```

返回信息如下所示。

```
Logging initialized using configuration in file:/etc/ecm/hive-conf-2.3.5-2.0.3/hive-log4j2.properties Async: true
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution
engine (i.e. spark, tez) or using Hive 1.X releases.
```

- 当集群有高安全选项时，提交方式如下所示。
  i. 执行如下命令进行认证。

  ```
  kinit -kt /etc/ecm/hive-conf/hive.keytab hive/emr-header-1.cluster-xxx@EMR.xxx.COM
  ```

  本文示例中的 `xxx` ，您可以在emr-header-1上通过命令 `hostname` 获取。

  您也可以通过用户管理功能创建用户，在连接Hive前使用 `kinit 用户名` 并输入密码，即可通过新建的用户使用Hive客户端。创建用户详情请参见管理用户。

  a. 执行如下命令进行认证。

  ```
  kinit 用户名
  ```

  b. 根据提示输入用户的密码。

  ```
  Password for 用户名@EMR.xxx.COM:
  ```

  ii. 连接Hive。

  ```
  hive
  ```

  返回信息如下所示。

  ```
  Logging initialized using configuration in file:/etc/ecm/hive-conf-2.3.5-2.0.3/hive-log4j2.properties Async: true
  Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execut
  ion engine (i.e. spark, tez) or using Hive 1.X releases.
  ```

方式二：通过Beeline

- 普通集群，提交方式如下所示。

```
beeline -u jdbc:hive2://emr-header-1:10000
```

返回信息如下所示。

```
Connecting to jdbc:hive2://emr-header-1:10000
Connected to: Apache Hive (version 2.3.5)
Driver: Hive JDBC (version 2.3.5)
Transaction isolation: TRANSACTION_REPEATABLE_READ
Beeline version 2.3.5 by Apache Hive
0: jdbc:hive2://emr-header-1:10000>
```

- 当集群有高可用选项时，提交方式如下所示

```
beeline -u 'jdbc:hive2://emr-header-1:2181,emr-header-2:2181,emr-header-3:2181/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=hiveserver2'
```

返回信息如下。

```
Connecting to jdbc:hive2://emr-header-1:2181,emr-header-2:2181,emr-header-3:2181/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=hiveserver2
21/08/27 14:37:03 [main]: INFO jdbc.HiveConnection: Connected to emr-header-1.cluster-***:10000
Connected to: Apache Hive (version 3.1.2)
Driver: Hive JDBC (version 3.1.2)Transaction isolation: TRANSACTION_REPEATABLE_READ
Beeline version 3.1.2 by Apache Hive
0: jdbc:hive2://emr-header-1:2181,emr-header->
```

- 当集群有高安全选项时，通过以下步骤提交。
  i. 执行如下命令进行认证。

  ```
  kinit -kt /etc/ecm/hive-conf/hive.keytab hive/emr-header-1.cluster-xxx@EMR.xxx.COM
  ```

  本文示例中的 `xxx` ，您可以在emr-header-1上通过命令 `hostname` 获取。

  您也可以通过用户管理功能创建用户，在连接Beeline前使用 `kinit 用户名` 并输入密码，即可通过新建用户使用Beeline客户端。创建用户详情请参见管理用户。

  a. 执行如下命令进行认证。

  ```
  kinit 用户名
  ```

  b. 根据提示输入用户的密码。

  ```
  Password for 用户名@EMR.xxx.COM:
  ```

  ii. 连接Beeline。

  ```
  beeline -u "jdbc:hive2://emr-header-1:10000/;principal=hive/emr-header-1.cluster-xxx@EMR.xxx.COM"
  ```

  > **说明** `jdbc` 链接串需要用双引号括起来。

  返回信息如下。

  ```
  Connecting to jdbc:hive2://emr-header-1:10000/;principal=hive/emr-header-1.cluster-202618@EMR.202618.COM
  Connected to: Apache Hive (version 2.3.5)
  Driver: Hive JDBC (version 2.3.5)
  Transaction isolation: TRANSACTION_REPEATABLE_READ
  Beeline version 2.3.5 by Apache Hive
  0: jdbc:hive2://emr-header-1:10000/>
  ```

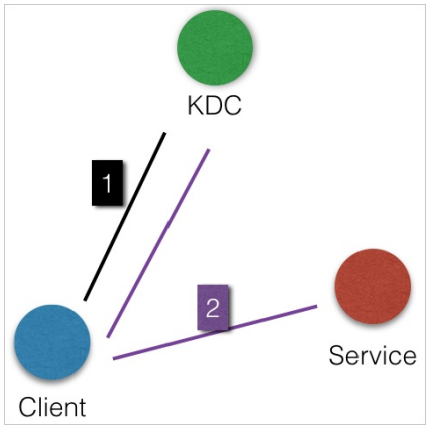
6.2.23.4. 高阶使用

6.2.23.4.1. Hive使用Kerberos

Kerberos是一种基于对称密钥技术的身份认证协议，可以为其他服务提供身份认证功能，且支持SSO（即客户端身份认证后，可以访问多个服务，例如HBase和HDFS）。本文为您介绍Hive如何使用Kerberos。

背景信息

Kerberos协议过程主要有两个阶段，第一个阶段是KDC对Client身份认证，第二个阶段是Service对Client身份认证。



- KDC：Kerberos的服务端程序。
- Client：需要访问服务的用户（Principal），KDC和Service会对用户的身份进行认证。
- Service：集成了Kerberos的服务。例如，HDFS、YARN和HBase。
- 第一阶段：KDC对Client身份认证

  当客户端用户（Principal）访问一个集成了Kerberos的服务之前，需要先通过KDC的身份认证。

  如果身份认证通过，则客户端会获取到一个TGT（Ticket Granting Ticket），后续就可以使用该TGT去访问集成了Kerberos的服务。
- 第二阶段：Service对Client身份认证

  当用户获取TGT后，就可以继续访问Service服务。

  使用TGT以及需要访问的服务名称（例如HDFS）去KDC获取SGT（Service Granting Ticket），然后使用SGT去访问Service。Service会利用相关信息对Client进行身份认证，认证通过后就可以正常访问Service服务。

前提条件

已创建Hadoop集群，详情请参见创建集群。

创建集群时，在**软件配置**页面的**高级设置**区域中，打开**Kerberos集群模式**开关。

操作步骤

1. 创建Principal。
   i. 通过SSH方式连接集群的emr-header-1节点，详情请参见登录集群。
   ii. 执行如下命令，进入Kerberos的admin工具。
   - EMR-3.30.0及后续版本和EMR-4.5.1及后续版本：

     ```
     sh /usr/lib/has-current/bin/admin-local.sh /etc/ecm/has-conf -k /etc/ecm/has-conf/admin.keytab
     ```
   - EMR-3.30.0之前版本和EMR-4.5.1之前版本：

     ```
     sh /usr/lib/has-current/bin/hadmin-local.sh /etc/ecm/has-conf -k /etc/ecm/has-conf/admin.keytab
     ```

   iii. 执行如下命令，创建用户名为test的Principal。

   本示例密码设置为123456。

   ```
   addprinc -pw 123456 test
   ```

   说明 需要记录用户名和密码，在创建TGT时会用到。如果您不想记录用户名和密码，则可以执行下一步，把Principal的用户名和密码导入到keytab文件中。

iv. （可选）执行如下命令，生成keytab文件。

```
ktadd -k /root/test.keytab test
```

执行 `quit` 命令，可以退出Kerberos的admin工具。

2. 创建TGT。

创建TGT的机器，可以是任意一台需要运行Hive Client的机器。

i. 使用root用户执行以下命令，创建test用户。

```
useradd test
```

ii. 执行以下命令，切换为test用户。

```
su test
```

iii. 生成TGT。

- 方式一：使用用户名和密码方式，创建TGT。

  执行 `kinit` 命令，回车后输入test的密码123456。

```
[root@emr-header-1 ~]# su test
[test@emr-header-1 root]$ kinit
Password for test@EMR.238075.COM:
[test@emr-header-1 root]$
```

- 方式二：使用keytab文件，创建TGT。

  在步骤1中的*test.keytab*文件，已经保存在emr-header-1机器的*/root/*目录下，需要使用 `scp` 命令拷贝到当前机器的*/home/test/*目录下。

```
kinit -kt /home/test/test.keytab test
```

iv. 查看TGT。

使用 `klist` 命令，如果出现如下信息，则说明TGT创建成功，即可以访问Hive了。

```
Ticket cache: FILE:/tmp/krb5cc_1012
Default principal: test@EMR.23****.COM
Valid starting       Expires              Service principal
07/24/2021 13:20:44  07/25/2021 13:20:44  krbtgt/EMR.23****.COM@EMR.23****.COM
        renew until 07/25/2021 13:20:44
```

> 注意 需要记录下回显信息 `EMR.23****.COM` 中的数字 `23****` ，即为 `cluster_id` 的值，后面访问Hive时需要。

3. 访问应用Hive。

i. 执行以下命令，进入Hive命令行。

```
hive
```

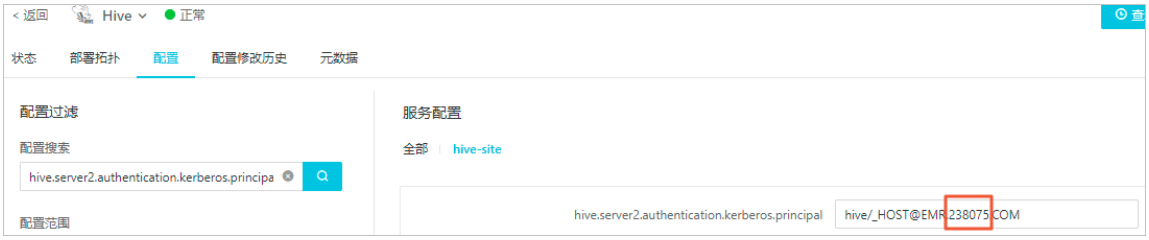
返回信息如下所示时，表示进入Hive命令行。

```
Logging initialized using configuration in file:/etc/ecm/hive-conf-2.3.5-2.0.3/hive-log4j2.properties Async: true
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different exec
ution engine (i.e. spark, tez) or using Hive 1.X releases.
```

ii. 执行以下命令，访问Hive。

```
/usr/lib/hive-current/bin/beeline -u "jdbc:hive2://emr-header-1:10000/default;principal=hive/emr-header-1@EMR.<clus
ter_id>.COM"
```

代码中的 `<cluster_id>` 为查看TGT中获取到的，或者您也可以在EMR控制台的Hive服务的配置页面，在搜索区域，搜索参数**hive.server2.authentication.kerberos.principal**，参数值中的数字即为 `<cluster_id>` 。



相关文档

- 创建Principal的官方文档，请参见Database administration。
- 创建TGT的官方文档，请参见kinit。

6.2.23.4.2. Hive授权

Hive内置有基于底层HDFS的权限（Storage Based Authorization）和基于标准SQL的grant等命令（SQL Standards Based Authorization）。本文为您介绍Hive的两种授权方式。

前提条件

- 已创建集群，详情请参见创建集群。
- 已登录集群，详情请参见登录集群。

背景信息

- 如果您可以直接通过HDFS或Hive Client访问Hive的数据，需要对Hive在HDFS中的数据进行相关的权限控制，通过HDFS权限控制，进而可以控制Hive SQL相关的操作权限。您可以使用Storage Based Authorization的授权方式，详情请参见方式一：Storage Based Authorization。
- 如果您不能直接通过HDFS或Hive Client访问，只能通过 `HiveServer2（Beeline或JDBC）` 等来执行Hive相关的命令，可以使用SQL Standards Based Authorization的授权方式，详情请参见方式二：SQL Standards Based Authorization。

> **说明** Storage Based Authorization（针对HiveMetaStore）和SQL Standards Based Authorization两种授权机制可以同时配置，不冲突。

方式一：Storage Based Authorization

- 授权配置
  i. 进入Hive页面。
     a. 登录阿里云E-MapReduce控制台。
     b. 在顶部菜单栏处，根据实际情况选择地域和资源组。
     c. 单击上方的**集群管理**页签。
     d. 在**集群管理**页面，单击相应集群所在行的**详情**。
     e. 在左侧导航栏中，选择**集群服务 > Hive**。
  ii. 在Hive页面，修改配置。
     a. 单击**配置**页签。
     b. 在**服务配置**区域，单击**hive-site**。
     c. 单击右侧的**自定义配置**，在**新增配置项**对话框中，增加配置项。

Key	Value
hive.metastore.pre.event.listeners	org.apache.hadoop.hive.ql.security.authorization.AuthorizationPreEventListener
hive.security.metastore.authorization.manager	org.apache.hadoop.hive.ql.security.authorization.StorageBasedAuthorizationProvider
hive.security.metastore.authenticator.manager	org.apache.hadoop.hive.ql.security.HadoopDefaultMetastoreAuthenticator

  iii. 保存配置。
     a. 在Hive服务页面，单击右上角的**保存**。
     b. 在**确认修改**对话框中，输入**执行原因**，单击**确定**。
  iv. 重启服务。
     a. 在**Hive**页面，选择**操作 > 重启Hive MetaStore**。
     b. 在**执行集群操作**对话框中，输入**执行原因**，单击**确定**。
     c. 在**确认**对话框中，单击**确定**。
- 权限控制

  EMR的Kerberos安全集群中已经设置了Hive的warehouse的HDFS相关权限。

  对于非Kerberos的安全集群，您需要执行步骤设置Hive基本的HDFS权限：

  i. 执行以下命令，配置Hive中warehouse文件夹的权限。

```
hadoop fs -chmod 1771 /user/hive/warehouse
```

也可以通过如下命令配置，其中1表示stick bit（不能删除别人创建的文件或文件夹）。

```
hadoop fs -chmod 1777 /user/hive/warehouse
```

ii. 执行以下命令，切换为has用户。

```
sudo su has
```

iii. 修改用户或用户组的权限。

- 授予test对warehouse文件夹的rwx权限。

```
hadoop fs -setfacl -m user:test:rwx /user/hive/warehouse
```

- 授予hivegrp对warehouse文件夹的rwx权限。

```
hadoop fs -setfacl -m group:hivegrp:rwx /user/hive/warehouse
```

> **说明** 您只能访问HDFS中自己账号创建的hive表的数据。

- 验证

i. 使用test用户建表testtbl。

```
create table testtbl(a string);
```

返回如下错误信息，提示test用户没有创建表的权限，需要给test用户添加权限。

```
FAILED: Execution Error, return code 1 from org.apache.hadoop.hive.ql.exec.DDLTask. MetaException(message:Got excepti
on: org.apache.hadoop.security.AccessControlException Permission denied: user=test, access=WRITE, inode="/user/hive/w
arehouse/testtbl":hadoop:hadoop:drwxrwx--t
at org.apache.hadoop.hdfs.server.namenode.FSPermissionChecker.check(FSPermissionChecker.java:320)
at org.apache.hadoop.hdfs.server.namenode.FSPermissionChecker.check(FSPermissionChecker.java:292)
```

ii. 使用has用户，给test账号添加ACL，配置warehouse文件夹的rwx权限。

```
hadoop fs -setfacl -m user:test:rwx /user/hive/warehouse;
```

iii. 使用test账号进行以下操作。

a. 执行以下命令，创建表。

```
create table testtbl(a string);
```

b. 执行以下命令，查看hdfs中testtbl的目录的权限。

```
hadoop fs -ls /user/hive/warehouse;
```

返回信息如下所示。

```
drwxr-x---   - test hadoop          0 2020-11-25 14:51 /user/hive/warehouse/testtbl
```

从返回信息可以看出，test用户创建的表数据只有test和hadoop组可以读取，其他用户没有任何权限。

c. 执行以下命令，向表中插入数据。

```
insert into table testtbl select "hz";
```

iv. 使用foo用户访问testtbl表。

```
select * from testtbl;
```

返回如下信息。

```
FAILED: SemanticException Unable to fetch table testtbl. java.security.AccessControlException: Permission denied: use
r=foo, access=READ, inode="/user/hive/warehouse/testtbl":test:hadoop:drwxr-x---
    at org.apache.hadoop.hdfs.server.namenode.FSPermissionChecker.check(FSPermissionChecker.java:320)
    at org.apache.hadoop.hdfs.server.namenode.FSPermissionChecker.checkPermission(FSPermissionChecker.java:219)
```

返回信息提示foo用户无法对test用户进行读操作。测试修改和删除操作，也同样没有权限。需要授权给foo用户，需要通过HDFS的授权来实现。

v. 授权foo用户。

a. 执行以下命令，切换为has用户。

```
su has
```

b. 执行以下命令，授予foo用户对testtbl表的读权限。

```
hadoop fs -setfacl -R -m user:foo:r-x /user/hive/warehouse/testtbl
```

c. 使用foo用户访问testtbl表。

```
select * from testtbl;
```

返回如下信息，表示查询成功。

```
OK
```

> **说明** 通常可以根据需求新建一个hive用户的group，然后通过给group授权，将新用户添加到group中，同一个group的数据权限都可以访问。

方式二：SQL Standards Based Authorization

- 授权配置
  i. 进入Hive页面。
    a. 登录阿里云E-MapReduce控制台。
    b. 在顶部菜单栏处，根据实际情况选择地域和资源组。
    c. 单击上方的**集群管理**页签。
    d. 在**集群管理**页面，单击相应集群所在行的**详情**。
    e. 在左侧导航栏中，选择**集群服务 > Hive**。
  ii. 在Hive页面，修改配置。
    a. 单击**配置**页签。
    b. 在**服务配置**区域，单击**hive-site**。
    c. 单击右侧的**自定义配置**，在**新增配置项**对话框中，增加配置项。

Key	Value
hive.security.authorization.enabled	true
hive.users.in.admin.role	hive
hive.security.authorization.createtable.owner.grants	ALL

  iii. 保存配置。
    a. 在Hive服务页面，单击右上角的**保存**。
    b. 在**确认修改**对话框中，输入**执行原因**，单击**确定**。
  iv. 重启服务。
    a. 在**Hive**页面，选择**操作 > 重启HiveServer2**。
    b. 在**执行集群操作**对话框中，输入**执行原因**，单击**确定**。
    c. 在**确认**对话框中，单击**确定**。
- 权限控制

  权限操作的具体命令，请参见Apache Hive。
- 验证
  i. 使用foo用户通过Beeline客户端，访问test用户的testtbl表。

```
select * from testtbl;
```

  返回如下错误信息。

```
Error: Error while compiling statement: FAILED: HiveAccessControlException Permission denied: Principal [name=foo, type=USER] does not have following privileges for operation QUERY [[SELECT] on Object [type=TABLE_OR_VIEW, name=default.testtbl]] (state=42000,code=40000)
```

  ii. 使用test账号执行grant命令，授权foo的select操作权限。

```
grant select on table testtbl to user foo;
```

  iii. 再次使用foo用户访问testtbl表。

```
select * from testtbl;
```

foo用户可以正常访问testtbl表，返回信息如下所示。

```
INFO  : OK
+-------------+--+
| testtbl.a   |
+-------------+--+
| hz          |
+-------------+--+
1 row selected (0.787 seconds)
```

iv. 使用test账号回收foo的select权限。

```
revoke select from user foo;
```

返回信息如下所示。

```
 OK
 Time taken: 1.094 seconds
```

v. 使用foo用户访问testtbl表。

```
select * from testtbl;
```

foo用户已经无法正常访问testtbl表，返回信息如下所示。

```
Error: Error while compiling statement: FAILED: HiveAccessControlException Permission denied: Principal [name=foo, ty
pe=USER] does not have following privileges for operation QUERY [[SELECT] on Object [type=TABLE_OR_VIEW, name=default
.testtbl]] (state=42000,code=40000)
```

6.2.23.4.3. 管理LDAP认证

服务开启LDAP认证功能后，访问服务时需要提供LDAP身份认证（LDAP用户名和密码），以便于提升服务的安全性。开启LDAP功能对接的LDAP为E-MapReduce自带的OpenLDAP。开启LDAP认证的功能可以方便您使用LDAP认证，避免了复杂的配置过程。本文为您介绍如何一键开启和关闭LDAP认证。

前提条件

已创建Hadoop集群，详情请参见创建集群。

使用限制

EMR-3.34.0及后续版本或EMR-4.8.0及后续版本的Hadoop集群，支持一键开启LDAP认证。

开启LDAP认证

> 注意 Hive开启LDAP认证后，Hue访问Hive需要进行额外的配置，请参见Hue连接开启LDAP认证的引擎。

1. 进入Hive页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务 > Hive**。
2. 开启LDAP认证。
   i. 在Hive服务页面，选择右上角的**操作 > 开启LDAP认证**。
   ii. 在**执行集群操作**对话中，单击**确认**。
3. 单击上方的**查看操作历史**。
   直至操作状态显示**成功**。
4. 重启HiveServer2。
   i. 在Hive服务页面，选择右上角的**操作 > 重启HiveServer2**。
   ii. 在**执行集群操作**对话中，输入**执行原因**，单击**确定**。
   iii. 在**确认**对话中，单击**确定**。

访问HiveServer2

开启LDAP认证后，当您访问HiveServer2时需要提供LDAP认证凭据。

1. 通过SSH方式连接集群，请参见登录集群。
2. 您可以通过以下两种方式访问HiveServer2。
   - Beeline客户端：

     ```
     beeline -u jdbc:hive2://emr-header-1:10000 -n <user> -p <password>
     ```

     说明 user为LDAP的用户名，password为LDAP的密码。开启LDAP认证后，访问HiveServer2需要提供LDAP的用户名和密码，获取方式请参见管理用户。
   - JDBC：

     ```
     jdbc:hive2://emr-header-1:10000/default;user=<user>;password=<password>
     ```

关闭LDAP认证

1. 进入Hive页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务 > Hive**。
2. 关闭LDAP认证。
   i. 在Hive服务页面，选择右上角的**操作 > 关闭LDAP认证**。
   ii. 在**执行集群操作**对话中，单击**确认**。
3. 单击上方的**查看操作历史**。
   直至操作状态显示**成功**。
4. 重启HiveServer2。
   i. 在Hive服务页面，选择右上角的**操作 > 重启HiveServer2**。
   ii. 在**执行集群操作**对话中，输入**执行原因**，单击**确定**。
   iii. 在**确认**对话中，单击**确定**。

6.2.23.4.4. Hive作业调优

您可以通过调整内存、CPU和Task个数等，实现对Hive作业的调优。本文为您介绍如何调优Hive作业。

作业调优方案

作业调优方向	调优方案
参数调优	• 内存参数 • CPU参数 • Task数量优化 • 并行运行 • Fetch task • 开启向量化 • 合并小文件
代码优化	代码优化

代码优化

- 数据清洗
  i. 读取表时分区过滤，避免全表扫描。
  ii. 数据过滤之后再JOIN。
  iii. 重复使用数据时，避免重复计算，构建中间表，重复使用中间表。
- 多Distinct优化

- 优化前代码

  多个Distinct时，数据会出现膨胀。

```
select k,count(distinct case when a > 1 then user_id) user1,
       count(distinct case when a > 2 then user_id) user2,
       count(distinct case when a > 3 then user_id) user3,
       count(distinct case when a > 4 then user_id) user4
from t
group by k
```

- 优化后代码

  通过两次Group By的方式代替Distinct操作，通过内层的Group By去重并降低数据量，通过外层的Group By取sum，即可实现Distinct的效果。

```
select k,sum(case when user1 > 0 then 1 end) as user1,
       sum(case when user2 > 0 then 1 end) as user2,
       sum(case when user3 > 0 then 1 end) as user3,
       sum(case when user4 > 0 then 1 end) as user4
from
        (select k,user_id,count(case when a > 1 then user_id) user1,
                count(case when a > 2 then user_id) user2,
                count(case when a > 3 then user_id) user3,
                count(case when a > 4 then user_id) user4
        from t
        group by k,user_id
        ) tmp
group by k
```

- 数据倾斜

  热点Key处理方式如下：

  - 如果是Group By出现热点，请按照以下方法操作：

    a. 先开启Map端聚合。

```
set hive.map.aggr=true
hive.groupby.mapaggr.checkinterval=100000（用于设定Map端进行聚合操作的条目数）
```

    b. 可以对Key随机化打散，多次聚合，或者直接设置。

```
set hive.groupby.skewindata=true;
```

    当选项设定为true时，生成的查询计划有两个MapReduce任务。在第一个MapReduce中，Map的输出结果集合会随机分布到Reduce中， 每个部分进行聚合操作，并输出结果。这样处理的结果是，相同的Group By Key有可能分发到不同的Reduce中，从而达到负载均衡的目的；第二个MapReduce任务再根据预处理的数据结果按照Group By Key分布到Reduce中（这个过程可以保证相同的Group By Key分布到同一个Reduce中），最后完成最终的聚合操作。

  - 如果两个大表进行JOIN操作时，出现热点，则使用热点Key随机化。

    例如，log表存在大量user_id为null的记录，但是表bmw_users中不会存在user_id为空，则可以把null随机化再关联，这样就避免null值都分发到一个Reduce Task上。代码示例如下。

```
SELECT * FROM log a LEFT OUTER
JOIN bmw_users b ON
CASE WHEN a.user_id IS NULL THEN CONCAT('dp_hive',RAND()) ELSE a.user_id=b.user_id END;
```

  - 如果大表和小表进行JOIN操作时，出现热点，则使用MAP JOIN。

内存参数

您可以通过设置以下参数，对Map和Reduce阶段的内存进行调优：

- Map阶段

参数	描述	示例
mapreduce.map.java.opts	默认参数，表示JVM堆内存。	-Xmx2048m
mapreduce.map.memory.mb	默认参数，表示整个JVM进程占用的内存，计算方法为 `堆内存+堆外内存=2048+256` 。	2304

- Reduce阶段

参数	描述	示例
mapreduce.reduce.java.opts	默认参数，表示JVM堆内存。	-Xmx2048m
mapreduce.reduce.memory.mb	默认参数，表示整个JVM进程占用的内存，计算方法为 `堆内存+堆外内存=2048+256` 。	2304

CPU参数

您可以通过设置以下参数，对Map和Reduce任务的CPU进行调优。

参数	描述
mapreduce.map.cpu.vcores	每个Map任务可用的最多的CPU Core数目。
mapreduce.reduce.cpu.vcores	每个Reduce任务可用的最多的CPU Core数目。 **说明** 此设置在公平队列是不生效的，通常vCores用于较大的集群，以限制不同用户或应用程序的CPU。

Task数量优化

- Map Task数量优化

  在分布式计算系统中，决定Map数量的一个因素就是原始数据，在不加干预的情况下，原始数据有多少个块，就可能有多少个起始的Task，因为每个Task对应读取一个块的数据；当然这个也不是绝对的，当文件数量特别多，并且每个文件的大小特别小时，您就可以限制减少初始Map对相应的Task的数量，以减少计算资源的浪费，如果文件数量较少，但是单个文件较大，您可以增加Map的Task的数量，以减小单个Task的压力。

  通常，Map Task数量是由mapred.map.tasks、mapred.min.split.size和dfs.block.size决定的。

  i. Hive的文件基本上都是存储在HDFS上，而HDFS上的文件，都是分块的，所以具体的Hive数据文件在HDFS上分多少块，可能对应的是默认Hive起始的Task的数量，使用default_mapper_num参数表示。使用数据总大小除以dfs默认的最大块大小来决定初始默认数据分区数。

     初始默认的Map Task数量，具体公式如下。

     ```
     default_mapper_num = total_size/dfs.block.size
     ```

  ii. 计算Split的size，具体公式如下。

     ```
     default_split_size = max(mapred.min.split.size, min(mapred.max.split.size, dfs.block.size))
     ```

     上述公式中的mapred.min.split.size和mapred.max.split.size，分别为Hive计算的时Split的最小值和最大值。

  iii. 将数据按照计算出来的size划分为数据块，具体公式如下。

     ```
     split_num = total_size/default_split_size;
     ```

  iv. 计算的Map Task数量，具体公式如下。

     ```
     map_task_num = min(split_num, max(mapred.map.tasks, default_mapper_num))
     ```

     从上面的过程来看，Task的数量受各个方面的限制，不至于Task的数量太多，也不至于Task的数量太少。如果需要提高Task数量，就要降低mappred.min.split.size的数值，在一定的范围内可以减小default_split_size的数值，从而增加split_num的数量，也可以增大mapred.map.tasks的数量。

     注意 Hive on TEZ和Hive on MR使用是有差异的。例如，在Hive中执行一个Query时，可以发现Hive的执行引擎在使用Tez与MR时，两者生成的mapper数量差异较大。主要原因在于Tez中对inputSplit做了grouping操作，可以将多个inputSplit组合成更少的groups，然后为每个group生成一个mapper任务，而不是为每个inputSplit生成一个mapper任务。

- Reduce Task数量优化
  - 通过hive.exec.reducers.bytes.per.reducer参数控制单个Reduce处理的字节数。

    Reduce的计算方法如下。

    ```
    reducer_num = min(total_size/hive.exec.reducers.bytes.per.reducers, hive.exec.reducers.max)。
    ```

- 通过mapred.reduce.tasks参数来设置Reduce Task的数量。

  > **说明** 在TEZ引擎模式下，通过命令 `set hive.tez.auto.reducer.parallelism = true;` ，TEZ将会根据vertice的输出大小动态预估调整Reduce Task的数量。

  同Map一样，启动和初始化Reduce也会消耗时间和资源。另外，有多少个Reduce，就会有多少个输出文件，如果生成了很多个小文件，并且这些小文件作为下一个任务的输入，则会出现小文件过多的问题。

并行运行

并行运行表示同步执行Hive的多个阶段。Hive在执行过程中，将一个查询转化成一个或者多个阶段。某个特定的Job可能包含多个阶段，而这些阶段可能并非完全相互依赖的，也就是可以并行运行的，这样可以使得整个Job的运行时间缩短。

您可以通过设置以下参数，控制不同的作业是否可以同时运行。

参数	描述
hive.exec.parallel	默认值为false。设置true时，表示允许任务并行运行。
hive.exec.parallel.thread.number	默认值为8。表示允许同时运行线程的最大值。

Fetch task

您可以通过设置以下参数，在执行查询等语句时，不执行MapReduce程序，以减少等待时间。

参数	描述
hive.fetch.task.conversion	默认值为none。参数取值如下： • none：关闭Fetch task优化。 在执行语句时，执行MapReduce程序。 • minimal：只在SELECT、FILTER和LIMIT的语句上进行优化。 • more：在minimal的基础上更强大，SELECT不仅仅是查看，还可以单独选择列，FILTER也不再局限于分区字段，同时支持虚拟列（别名）。

开启向量化

您可以通过设置以下参数，在执行查询等语句时，不执行MapReduce程序，以减少等待时间。

参数	描述
hive.vectorized.execution.enabled	默认值为true。开启向量化查询的开关。
hive.vectorized.execution.reduce.enabled	默认值为true。表示是否启用Reduce任务的向量化执行模式。

合并小文件

大量小文件容易在文件存储端造成瓶颈，影响处理效率。对此，您可以通过合并Map和Reduce的结果文件来处理。

您可以通过设置以下参数，合并小文件。

参数	描述
hive.merge.mapfiles	默认值为true。表示是否合并Map输出文件。
hive.merge.mapredfiles	默认值为false。表示是否合并Reduce输出文件。
hive.merge.size.per.task	默认值为256000000，单位字节。表示合并文件的大小。

6.2.23.4.5. HiveServer2负载均衡

当E-MapReduce（简称EMR）集群有多个HiveServer2服务时，可以借助Zookeeper服务或负载均衡SLB（Server Load Balancer）实现访问HiveServer2的负载均衡，将HiveServer2的压力分担到多个节点上去。本文详细介绍HiveServer2负载均衡的配置及使用方法，请根据EMR集群（普通集群和Kerberos集群）的实际情况进行选择。

前提条件

已创建高可用集群，详情请参见创建集群。

使用限制

本文内容仅适用于打开**高可用**开关的集群。

EMR普通集群

以下方式适用于未打开**Kerberos集群模式**开关的集群。

高可用集群默认安装了Zookeeper服务，您可以使用以下连接方式选择一个HiveServer2进行连接，达到负载均衡的效果。您可以在EMR控制台查看Zookeeper服务**zookeeper_hosts**的参数值，获取Zookeeper的连接地址。详细操作如下：

1. 获取Zookeeper的连接地址。

   在EMR控制台的Zookeeper服务的配置页签，在搜索区域，搜索参数**zookeeper_hosts**，可以获取Zookeeper的连接地址。本示例中参数**zookeeper_hosts**的参数值为emr-header-1,emr-header-2,emr-worker-1，所以Zookeeper的连接地址为emr-header-1:2181,emr-header-2:2181,emr-worker-1:2181。

2. 访问HiveServer2。

   i. 通过SSH方式连接集群，详情请参见登录集群。

   ii. 执行以下命令，Zookeeper服务会选择一个HiveServer2进行连接。

   ```
   beeline -u 'jdbc:hive2://emr-header-1:2181,emr-header-2:2181,emr-worker-1:2181/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=hiveserver2'
   ```

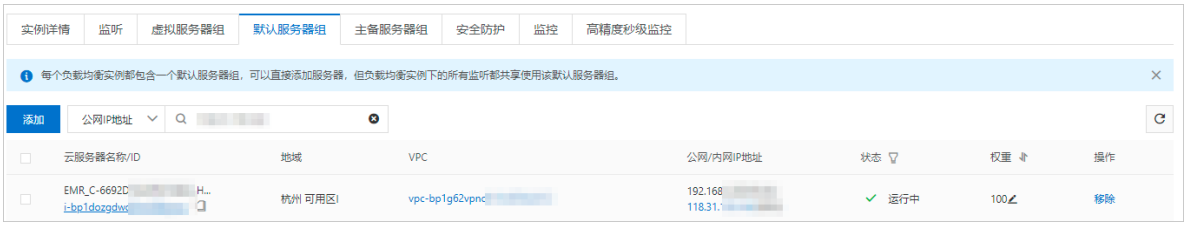
   > **说明** 代码中的emr-header-1:2181,emr-header-2:2181,emr-worker-1:2181为您获取到的Zookeeper的连接地址。

   Hue连接该负载均衡HiveServer2，需要在Hue配置的**hue**页签中添加如下三个自定义参数。

参数	描述
zookeeper.clusters.default.hostports	Zookeeper的连接地址，请根据实际情况填写，本示例为emr-header-1:2181,emr-header-2:2181,emr-worker-1:2181。
beeswax.hive_discovery_hs2	固定值为true。
beeswax.hive_discovery_hiveserver2_znode	固定值为/hiveserver2。

1. 创建SLB实例，详情请参见创建和管理CLB实例。
2. 在默认服务器组中添加EMR集群HiveServer2服务所在ECS实例，根据需要配置ECS的权重。

   详情请参见添加默认服务器和编辑后端服务器的权重。

3. 配置TCP端口，端口自定义，**后端协议/端口**配置为TCP:10000。SLB的调度算法根据需要进行选择。

   详情请参见添加TCP监听。

4. 访问HiveServer2。
   i. 通过SSH方式链接集群，详情请参见登录集群。
   ii. 执行以下命令，直接访问SLB对应的IP地址（或您使用已有的hostname绑定SLB的IP地址）和端口，即可以负载均衡的方式连接集群多个HiveServer2服务。

   ```
   beeline -u 'jdbc:hive2://<slb_ip_or_host>:<slb_port>'
   ```

   其中*<slb_ip_or_host>*为SLB的IP地址或绑定的hostname，*<slb_port>*为SLB前端监听的端口。

   Hue连接该负载均衡HiveServer2，需要在Hue配置的**hue**页签中修改以下参数。

参数	描述
hive_server2_host	填写为SLB的IP地址或绑定的hostname。
hive_server2_port	填写为SLB前端监听的端口。

Zookeeper实现负载均衡

SLB实现负载均衡

EMR Kerberos集群

以下方式适用于打开**Kerberos集群模式**开关的集群。

1. 首先执行 `kinit` 命令，通过Kerberos认证，获取Kerberos Ticket，详情请参见Hive使用Kerberos。
2. 获取Zookeeper的连接地址。

   在EMR控制台的Zookeeper服务的配置页签，在搜索区域，搜索参数**zookeeper_hosts**，可以获取Zookeeper的连接地址。本示例中参数**zookeeper_hosts**的参数值为emr-header-1,emr-header-2,emr-worker-1，所以Zookeeper的连接地址为emr-header-1:2181,emr-header-2:2181,emr-worker-1:2181。

3. 访问HiveServer2。

i. 通过SSH方式链接集群，详情请参见登录集群。

ii. 执行以下命令，Zookeeper服务会选择一个HiveServer2进行连接。

```
beeline -u 'jdbc:hive2://emr-header-1:2181,emr-header-2:2181,emr-worker-1:2181/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=hiveserver2'
```

> **说明** 代码中的emr-header-1:2181,emr-header-2:2181,emr-worker-1:2181为您获取到的Zookeeper的连接地址。

Hue暂不支持连接EMR Kerberos集群的Zookeeper负载均衡的HiveServer2，因此请使用SLB方式访问HiveServer2。

可以参见EMR普通集群中的内容创建及配置SLB负载均衡。在Kerberos集群，还需要额外配置HiveServer2所使用的Kerberos Principal才能正常连接。因此分下面两种方式介绍需要的操作。

- 方式一：使用SLB的IP地址以负载均衡方式访问HiveServer2

  本示例假设SLB的IP地址为121.41.**.**，前端监听端口为10000，具体请根据实际情况进行修改。

  i. 创建121.41.**.**地址对应的Hive Principal并导出Keytab。

  如果EMR Kerberos集群使用的是EMR自带的HAS Kerberos，则使用如下方式创建并导出SLB所需的Principal。

  a. 登录集群的emr-header-1节点，详情请参见登录集群。

  b. 执行以下命令，进入Kerberos的admin工具。

  - EMR-3.30.0及后续版本和EMR-4.5.1及后续版本。

    ```
    sh /usr/lib/has-current/bin/admin-local.sh /etc/ecm/has-conf -k /etc/ecm/has-conf/admin.keytab
    ```

  - EMR-3.30.0之前版本和EMR-4.5.1之前版本。

    ```
    sh /usr/lib/has-current/bin/hadmin-local.sh /etc/ecm/has-conf -k /etc/ecm/has-conf/admin.keytab
    ```

  c. 连接上HAS客户端之后，执行以下命令创建Principal，并将其导出到*/tmp/slb.keytab*中。

  ```
  addprinc -randkey hive/121.41.**.**
  xst -k /tmp/slb.keytab hive/121.41.**.**
  exit
  ```

  如果EMR Kerberos集群对接使用的是外部MIT Kerberos，则按照MIT Kerberos开源用法使用kadmin.local或kadmin连接KDC，执行 `addprinc` 和 `xst` 命令即可。

  > **注意** MIT Kerberos多次导出keytab会导致之前导出的keytab失效，所以需确保slb.keytab只导出一次。

  ii. 将*slb.keytab*写入到*hive.keytab*中。

  将*slb.keytab*传输到集群所有HiveServer2服务所在节点，使用ktutil工具将*slb.keytab*中存储的Principal导入到各个节点*/etc/ecm/hive-conf/hive.keytab*中，执行以下命令。

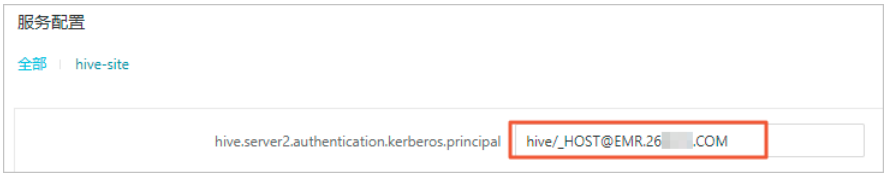
  ```
  ktutil
  rkt /tmp/slb.keytab
  wkt /etc/ecm/hive-conf/hive.keytab
  ```

  *hive.keytab*中应该有如下内容。

  ```
  [root@emr-header-1 ~]# klist -kt /etc/ecm/hive-conf/hive.keytab
  Keytab name: FILE:/etc/ecm/hive-conf/hive.keytab
  KVNO Timestamp           Principal
  ---- ------------------- ------------------------------------------------------
     1 2021-12-28T14:24:52 hive/emr-header-1.cluster-272[redacted]@EMR.272[redacted].COM
     1 2021-12-28T14:24:52 hive/emr-header-1.cluster-272[redacted]@EMR.272[redacted].COM
     1 2021-12-28T14:24:52 hive/121.40.[redacted]@EMR.272[redacted].COM
     1 2021-12-28T14:24:52 hive/121.40.[redacted]@EMR.272[redacted].COM
  ```

  iii. 修改集群的Hive配置。

  在EMR控制台的Hive服务的配置页签，在搜索区域，搜索参数hive.server2.authentication.kerberos.principal，修改参数值为hive/121.41.**.**@EMR.**.COM，即修改为上步骤中创建的Principal。然后保存并选中自动更新配置。

iv. 重启集群HiveServer2服务。

a. 在EMR控制台的Hive服务的配置页签，在右上角选择**操作 > 重启HiveServer2**。

b. 选择失败处理策略，输入执行原因，单击**确定**。

c. 在**确认**对话框中，单击**确定**。

v. 使用以下命令访问HiveServer2。

```
beeline -u 'jdbc:hive2://121.41.**.**/default;principal=hive/121.41.**.**@EMR.**.COM'
```

- 方式二：使用SLB的hostname以负载均衡方式访问HiveServer2

> 注意 Beeline在进行Kerberos认证时，会将Principal都转换为小写字母进行服务端Principal验证，为了与之保持一致，hostname中的英文字母必须全部小写不能有大写字母，否则容易因为大小写不统一造成Kerberos认证失败。与方式一相同，只是创建Principal和修改hive-site配置时，将IP地址修改为hostname。连接时也使用hostname进行连接。

使用以下命令访问HiveServer2。

```
beeline -u 'jdbc:hive2://<slb_hostname>/default;principal=hive/<slb_hostname>@EMR.**.COM'
```

Hue连接该负载均衡HiveServer2，需要在Hue配置的**hue**页签中修改以下参数。

参数	描述
hive_server2_host	填写为SLB的IP地址或绑定的hostname。
hive_server2_port	填写为SLB前端监听的端口。

Zookeeper实现负载均衡

SLB实现负载均衡

6.2.23.5. 开发指南

6.2.23.5.1. 自定义函数（UDF）

Hive提供了很多内建函数来满足您的计算需求，您也可以通过创建自定义函数（UDF）来满足不同的计算需求。UDF在使用上与普通的内建函数类似。本文为您介绍自定义函数的开发和使用流程。

背景信息

UDF分类如下表。

UDF分类	描述
UDF（User Defined Scalar Function）	自定义标量函数，通常称为UDF。其输入与输出是一对一的关系，即读入一行数据，写出一条输出值。
UDTF（User Defined Table-valued Function）	自定义表值函数，用来解决一次函数调用输出多行数据场景的，也是唯一一个可以返回多个字段的自定义函数。
UDAF（User Defined Aggregation Function）	自定义聚合函数，其输入与输出是多对一的关系，即将多条输入记录聚合成一条输出值，可以与SQL中的Group By语句联合使用。

前提条件

- 已创建集群，详情请参见创建集群。
- 本地安装了文件传输工具（SSH Secure File Transfer Client）。

开发UDF

1. 使用IDE，创建Maven工程。

   工程基本信息如下，您可以自定义groupId和artifactId。

   ```
   <groupId>org.example</groupId>
   <artifactId>hiveudf</artifactId>
   <version>1.0-SNAPSHOT</version>
   ```

2. 添加pom依赖。

```
<dependency>
      <groupId>org.apache.hive</groupId>
      <artifactId>hive-exec</artifactId>
      <version>2.3.7</version>
      <exclusions>
         <exclusion>
           <groupId>org.pentaho</groupId>
           <artifactId>*</artifactId>
         </exclusion>
      </exclusions>
</dependency>
```

3. 创建一个类，继承Hive UDF类。

   类名您可以自定义，本文示例中类名为MyUDF。

```
package org.example;
import org.apache.hadoop.hive.ql.exec.UDF;
/**
 * Hello world!
 *
 */
public class MyUDF extends UDF
{
    public String evaluate(final String s) {
        if (s == null) { return null; }
        return s + ":HelloWorld";
    }
}
```

4. 将自定义的代码打成JAR包。

   在*pom.xml*所在目录，执行如下命令制作JAR包。

```
mvn clean package -DskipTests
```

   *target*目录下会出现*hiveudf-1.0-SNAPSHOT.jar*的JAR包，即代表完成了UDF开发工作。

使用UDF

1. 使用文件传输工具，上传生成的JAR包至集群root目录。
2. 上传JAR包至HDFS。
   i. 通过SSH方式登录集群，详情请参见登录集群。
   ii. 执行以下命令，上传JAR包到HDFS。

```
hadoop fs -put hiveudf-1.0-SNAPSHOT.jar /user/hive/warehouse/
```

   您可以通过 `hadoop fs -ls /user/hive/warehouse/` 命令，查看是否上传成功。待返回信息如下所示表示上传成功。

```
Found 1 items
-rw-r--r--   1 xx xx 2668 2021-06-09 14:13 /user/hive/warehouse/hiveudf-1.0-SNAPSHOT.jar
```

3. 创建UDF函数。
   i. 执行以下命令，进入Hive命令行。

```
hive
```

   ii. 执行以下命令，应用生成的JAR包创建函数。

```
create function myfunc as "org.example.MyUDF" using jar "hdfs:///user/hive/warehouse/hiveudf-1.0-SNAPSHOT.jar";
```

   代码中的 `myfunc` 是UDF函数的名称， `org.example.MyUDF` 是开发UDF中创建的类， `dfs:///user/hive/warehouse/hiveudf-1.0-SNAPSHOT.jar` 为上传JAR包到HDFS的路径。

   当出现以下信息时，表示创建成功。

```
Added [/private/var/folders/2s/wzzsgpn13rn8rl_0fc4xxkc00000gp/T/40608d4a-a0e1-4bf5-92e8-b875fa6a1e53_resources/hiveudf-1.0-SNAPSHOT.jar] to class path
Added resources: [hdfs:///user/hive/warehouse/myfunc/hiveudf-1.0-SNAPSHOT.jar]
```

   > **说明** 您也可以通过命令 `SHOW FUNCTIONS LIKE '*myfunc* '`，验证函数是否创建成功。

4. 执行以下命令，使用UDF函数。

该函数与内置函数使用方式一样，直接使用函数名称即可访问。

```
select myfunc("abc");
```

返回如下信息。

```
OK
abc:HelloWorld
```

相关文档

- UDF
- UDAF
- UDTF

6.2.23.6. 最佳实践

6.2.23.6.1. Hive访问EMR HBase数据

Hive支持通过内表或外表两种方式访问E-MapReduce（简称EMR）HBase数据。本文通过示例为您介绍，如何使用EMR上的Hive处理EMR HBase数据。

前提条件

已创建Hadoop集群，并且选择了HBase和Zookeeper服务，详情请参见创建集群。

Hive通过内表访问HBase

如果HBase中没有已经创建好的表，则可以在Hive中创建表，Hive会自动把表结构和数据写入到HBase中。本示例是在Hive中新建表访问HBase。

1. 进入Hive命令行。
   i. 使用SSH方式登录到集群主节点，详情请参见登录集群。
   ii. 执行以下命令，进入Hive命令行。

   ```
   hive
   ```

   返回信息如下所示时，表示进入Hive命令行。

   ```
   Logging initialized using configuration in file:/etc/ecm/hive-conf-2.3.5-2.0.3/hive-log4j2.properties Async: true
   Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different exec
   ution engine (i.e. spark, tez) or using Hive 1.X releases.
   ```

2. 在Hive中创建并查询表数据。
   i. 执行以下命令，在Hive中创建HBase表。

   ```
   create table hive_hbase_table(key int, value string)
   stored by 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
   with serdeproperties("hbase.columns.mapping" = ":key,cf1:val")
   tblproperties("hbase.table.name" = "hive_hbase_table", "hbase.mapred.output.outputtable" = "hive_hbase_table");
   ```

   > **说明** 表的存储方式是HBaseStorageHandler，可以存储和读取HBase数据。

   ii. 执行以下命令，向表中插入数据。

   ```
   insert into hive_hbase_table values(212,'bab');
   ```

   iii. 执行以下命令，查看表数据。

   ```
   select * from hive_hbase_table;
   ```

   返回信息如下。

   ```
   OK
   212 bab
   Time taken: 0.337 seconds, Fetched: 1 row(s)
   ```

3. 进入HBase命令行。
   i. 使用SSH方式登录到集群主节点，详情请参见登录集群。

ii. 执行以下命令，进入HBase命令行。

```
hbase shell
```

返回信息如下所示时，表示进入HBase命令行。

```
[root@emr-header-1 ~]# hbase shell
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/apps/ecm/service/hbase/1.4.9-1.0.0/packag
e/hbase-1.4.9-1.0.0/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBin
der.class]
SLF4J: Found binding in [jar:file:/opt/apps/ecm/service/hadoop/2.8.5-1.6.4/packa
ge/hadoop-2.8.5-1.6.4/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4
j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
Version 1.4.9, r8214a16c5d80f077abf1aa01bb31285151la2b15, Thu Jan 31 20:35:22 CS
T 2019

hbase(main):001:0>
```

4. 执行以下命令，查看是否已经通过Hive在HBase中创建了表。

```
describe 'hive_hbase_table'
```

返回信息如下。

```
hbase(main):001:0> describe 'hive_hbase_table'
Table hive_hbase_table is ENABLED
hive_hbase_table
COLUMN FAMILIES DESCRIPTION
{NAME => 'cf1', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEE
65536', REPLICATION_SCOPE => '0'}
1 row(s) in 0.2320 seconds
```

> **说明** 查看表已存在，说明Hive已经在HBase中创建了表。

5. 执行以下命令，在HBase中查看Hive写的数据是否已存在。

```
scan 'hive_hbase_table'
```

返回信息如下。

```
ROW                                     COLUMN+CELL

 212                                    column=cf1:val, timestamp=1624513121062, value=bab

1 row(s) in 0.2320 seconds
```

> **说明** 查看数据已存在，并且与在Hive中插入的数据一致，说明Hive已经成功访问了HBase的数据。

Hive通过外表访问HBase

如果已经在HBase中创建了表，想通过Hive访问，则可以使用Hive外表的方式与HBase中的表建立映射关系，进而通过Hive访问HBase中已经存在的表。

1. 进入HBase命令行。
   i. 使用SSH方式登录到集群主节点，详情请参见登录集群。

ii. 执行以下命令，进入HBase命令行。

```
hbase shell
```

返回信息如下所示时，表示进入HBase命令行。

```
[root@emr-header-1 ~]# hbase shell
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/apps/ecm/service/hbase/1.4.9-1.0.0/packag
e/hbase-1.4.9-1.0.0/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBin
der.class]
SLF4J: Found binding in [jar:file:/opt/apps/ecm/service/hadoop/2.8.5-1.6.4/packa
ge/hadoop-2.8.5-1.6.4/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4
j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
Version 1.4.9, r8214a16c5d80f077abf1aa01bb31285151la2b15, Thu Jan 31 20:35:22 CS
T 2019

hbase(main):001:0>
```

2. 在HBase中创建并查询表数据。

i. 执行以下命令，在HBase中创建表。

```
create 'hbase_table','f'
```

ii. 执行以下命令，向表中插入数据。

```
put 'hbase_table','1122','f:col1','hello'
```

iii. 执行以下命令，查看表数据。

```
scan 'hbase_table'
```

返回信息如下。

```
ROW                                                  COLUMN+CELL
 1122                                                column=f:col1, timestamp=1627027165760, value=hello
1 row(s) in 0.0170 seconds
```

3. 进入Hive命令行。

i. 使用SSH方式登录到集群主节点，详情请参见登录集群。

ii. 执行以下命令，进入Hive命令行。

```
hive
```

返回信息如下所示时，表示进入Hive命令行。

```
Logging initialized using configuration in file:/etc/ecm/hive-conf-2.3.5-2.0.3/hive-log4j2.properties Async: true
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different exec
ution engine (i.e. spark, tez) or using Hive 1.X releases.
```

4. 执行以下命令，在Hive中创建外表，并与HBase中的表建立映射关系。

```
create external table hbase_table(key int,col1 string,col2 string)
stored by 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
with serdeproperties("hbase.columns.mapping" = "f:col1,f:col2")
tblproperties("hbase.table.name" = "hbase_table", "hbase.mapred.output.outputtable" = "hbase_table");
```

5. 执行以下命令，在Hive中查看hbase_table数据。

```
select * from hbase_table;
```

返回信息如下。

```
OK
1122  hello NULL
Time taken: 2.201 seconds, Fetched: 1 row(s)
```

> **说明** 查看数据已存在，并且与在HBase中插入的数据一致，说明Hive已经成功访问了HBase的数据。

6.2.23.6.2. Hive访问EMR Phoenix数据

Hive支持通过内表或外表两种方式访问E-MapReduce（简称EMR）中的Phoenix数据。本文通过示例为您介绍如何使用EMR上的Hive处理EMR Phoenix数据。

前提条件

已创建Hadoop集群，并且选择了HBase、Zookeeper和Phoenix服务，详情请参见创建集群。

> **说明** 因为当前EMR-4.x和EMR-5.x系列版本未支持Phoenix服务，所以此文档仅适用于EMR-3.x系列版本。

Hive通过内表访问Phoenix

如果Phoenix中没有已经创建好的表，则可以在Hive中创建表，存储到Phoenix中。本示例是在Hive中新建表访问Phoenix。

1. 进入Hive命令行。
   i. 使用SSH方式登录到集群主节点，详情请参见登录集群。
   ii. 执行以下命令，进入Hive命令行。

   ```
   hive
   ```

   返回信息如下所示时，表示进入Hive命令行。

   ```
   Logging initialized using configuration in file:/etc/ecm/hive-conf-2.3.5-2.0.3/hive-log4j2.properties Async: true
   Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different exec
   ution engine (i.e. spark, tez) or using Hive 1.X releases.
   ```

2. 在Hive中创建并查询表数据。
   i. 执行以下命令，在Hive中创建Phoenix表。

   ```
   create table phoenix_hive_create_internal(s1 string,i1 int,f1 float,d1 double)
   stored by 'org.apache.phoenix.hive.PhoenixStorageHandler'
   tblproperties(
   "phoenix.table.name" = "phoenix_hive_create_internal",
   "phoenix.rowkeys" = "s1,i1",
   "phoenix.column.mapping" = "s1:s1,i1:i1,f1:f1,d1:d1",
   "phoenix.table.options" = "SALT_BUCKETS=10,DATA_BLOCK_ENCODING='DIFF'"
   );
   ```

   > **说明** 表的存储方式是PhoenixStorageHandler，可以存储和读取Phoenix数据。

   ii. 执行以下命令，向表中插入数据。

   ```
   insert into phoenix_hive_create_internal values('wyk',1,2.3412,3.14);
   ```

   iii. 执行以下命令，查看表数据。

   ```
   select * from phoenix_hive_create_internal;
   ```

   返回信息如下。

   ```
   OK
   wyk     1       2.3412  3.14
   Time taken: 0.569 seconds, Fetched: 1 row(s)
   ```

3. 进入Phoenix命令行。
   i. 使用SSH方式登录到集群主节点，详情请参见登录集群。
   ii. 执行以下命令，进入*phoenix-current*目录。

   ```
   cd /usr/lib/phoenix-current/
   ```

   iii. 执行以下命令，进入Phoenix命令行。

   ```
   python ./bin/sqlline.py
   ```

4. 执行以下命令，在Phoenix中查看Hive写的数据是否已存在。

   ```
   select * from phoenix_hive_create_internal;
   ```

   返回信息如下。

```
+------+-----+---------+-------+
|  s1  | i1  |   f1    |  d1   |
+------+-----+---------+-------+
| wyk  | 1   | 2.3412  | 3.14  |
+------+-----+---------+-------+
```

说明 查看数据已存在，并且与在Hive中插入的数据一致，说明Hive已经成功访问了Phoenix的数据。

Hive通过外表访问Phoenix

如果已经在Phoenix中创建了表*phoenix_hive_create_internal*，想通过Hive访问，则可以使用Hive外表的方式与Phoenix中的表建立映射关系，进而通过Hive访问Phoenix中已经存在的表。

1. 进入Hive命令行。
   i. 使用SSH方式登录到集群主节点，详情请参见登录集群。
   ii. 执行以下命令，进入Hive命令行。

   ```
   hive
   ```

   返回信息如下所示时，表示进入Hive命令行。

   ```
   Logging initialized using configuration in file:/etc/ecm/hive-conf-2.3.5-2.0.3/hive-log4j2.properties Async: true
   Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different exec
   ution engine (i.e. spark, tez) or using Hive 1.X releases.
   ```

2. 执行以下命令，在Hive中创建外表，建立与Phoenix表的映射关系。

   ```
   create external table ext_table(
     s1 string,
     i1 int,
     f1 float,
     d1 double
   )
   stored by 'org.apache.phoenix.hive.PhoenixStorageHandler'
   tblproperties(
     "phoenix.table.name" = "phoenix_hive_create_internal",
     "phoenix.rowkeys" = "s1, i1",
     "phoenix.column.mapping" = "s1:s1, i1:i1, f1:f1, d1:d1"
   );
   ```

3. 执行以下命令，在Hive中查看Phoenix表的数据。

   ```
   select * from ext_table;
   ```

   如果可以正常查询数据，说明Hive已经成功访问了Phoenix的数据。

相关文档

- Phoenix相关的介绍，请参见Phoenix。
- Phoenix接入Hive的内容，请参见Phoenix Storage Handler for Apache Hive。

6.2.23.6.3. Hive访问Delta Lake和Hudi数据

Hive不支持写入数据到Delta Lake和Hudi，但是可以通过外部表的方式查询Delta Lake和Hudi中的数据。本文通过示例为您介绍如何使用EMR上的Hive访问Delta Lake和Hudi数据。

前提条件

已创建Hadoop集群，详情请参见创建集群。

使用限制

EMR-3.36.0及后续版本和EMR-5.2.0及后续版本，支持Hive对Hudi进行读操作。

Hive访问Delta Lake数据

1. 进入Spark命令行。
   i. 使用SSH方式登录到集群主节点，详情请参见登录集群。
   ii. 执行以下命令，进入Spark命令行。

   ```
   spark-sql
   ```

2. 在Spark中创建并查询表数据。
   i. 执行以下命令，在Spark中创建Delta表。

   ```
   create table delta_table (id int) using delta location "/tmp/delta_table";
   ```

   ii. 执行以下命令，向表中插入数据。

   ```
   insert into delta_table values 0,1,2,3,4;
   ```

   iii. 执行以下命令，查看表数据。

   ```
   select * from delta_table;
   ```

   返回包含如下的信息。

   ```
   2
   3
   4
   0
   1
   Time taken: 1.847 seconds, Fetched 5 row(s)
   ```

3. 在Hive中查看Delta Lake数据。
   i. 执行以下命令，进入Hive命令行。

   ```
   hive
   ```

   ii. 执行以下命令，在Hive中查看Delta Lake表。

   ```
   desc formatted delta_table;
   ```

   返回如下信息。

   ```
   OK
   # col_name              data_type               comment
   id                      int
   # Detailed Table Information
   Database:               default
   Owner:                  root
   CreateTime:             Sat Jul 24 14:20:44 CST 2021
   LastAccessTime:         UNKNOWN
   Retention:              0
   Location:               hdfs://emr-header-1.cluster-238095:9000/tmp/delta_table
   Table Type:             EXTERNAL_TABLE
   Table Parameters:
           EXTERNAL                TRUE
           delta.database          default
           delta.syncMetadataToCatalog     true
           delta.table             delta_table
           delta.tableType         EXTERNAL_TABLE
           last_modified_time      1627107644
           numFiles                0
           spark.sql.sources.provider      delta
           spark.sql.sources.schema.numParts       1
           spark.sql.sources.schema.part.0 {\"type\":\"struct\",\"fields\":[{\"name\":\"id\",\"type\":\"integer\",\"nu
   llable\":true,\"metadata\":{}}]}
           totalSize               0
           transient_lastDdlTime   1627107644
   # Storage Information
   SerDe Library:          org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe
   InputFormat:            io.delta.hive.DeltaInputFormat
   OutputFormat:           io.delta.hive.DeltaOutputFormat
   Compressed:             No
   Num Buckets:            0
   Bucket Columns:         []
   Sort Columns:           []
   Storage Desc Params:
           path                    hdfs://emr-header-1.cluster-238095:9000/tmp/delta_table
           serialization.format    1
   Time taken: 1.09 seconds, Fetched: 37 row(s)
   ```

iii. 执行以下命令，在Hive中查看Delta Lake表的数据。

```
select * from delta_table;
```

返回如下信息。

```
OK
2
3
4
0
1
Time taken: 1.897 seconds, Fetched: 5 row(s)
```

> **说明** 查看数据与在Spark中插入的数据一致，说明Hive已经成功访问了Delta Lake的数据。

Hive访问Hudi数据

> **注意** EMR-3.36.0及后续版本和EMR-5.2.0及后续版本，支持Hive对Hudi进行读操作。

1. 进入Spark命令行。
   i. 使用SSH方式登录到集群主节点，详情请参见登录集群。
   ii. 执行以下命令，进入Spark命令行。

```
spark-sql --conf 'spark.serializer=org.apache.spark.serializer.KryoSerializer' \
--conf 'spark.sql.extensions=org.apache.spark.sql.hudi.HoodieSparkSessionExtension'
```

2. 在Spark中创建并查询表数据。
   i. 执行以下命令，在Spark中创建Hudi表。

```
create table h0 (
               id bigint,
               name string,
               price double
             ) using hudi;
```

您可以执行以下命令，查看表结构属性。

```
desc formatted h0;
```

返回信息中会多出以下字段，这些字段是Hudi的默认字段，会同步到元数据中。

```
_hoodie_commit_time string  NULL
_hoodie_commit_seqno  string  NULL
_hoodie_record_key  string  NULL
_hoodie_partition_path  string  NULL
_hoodie_file_name string  NULL
```

   ii. 执行以下命令，向表中插入数据。

```
insert into h0 select 1, 'a1', 10;
```

   iii. 执行以下命令，查询表数据。

```
select * from h0;
```

返回如下信息。

```
OK
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
20210728180336  20210728180336_0_1      74641e17-75d6-4243-be71-a5aa98b5c1af            49062904-52da-402c-82c1-84c
04d3c2a4c-0_0-6-6_20210728180336.parquet     1       a1      10.0
Time taken: 2.001 seconds, Fetched: 1 row(s)
```

3. 在Hive中查看Hudi数据。
   i. 执行以下命令，进入Hive命令行。

```
hive
```

ii. 执行以下命令，查询表数据。

```
select * from h0;
```

返回如下信息。

```
OK
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
20210728180336  20210728180336_0_1      74641e17-75d6-4243-be71-a5aa98b5c1af            49062904-52da-402c-82c1-84c
04d3c2a4c-0_0-6-6_20210728180336.parquet     1       a1      10.0
Time taken: 2.001 seconds, Fetched: 1 row(s)
```

说明 查看的数据与在Spark中插入的数据一致，说明Hive已经成功访问了Hudi的数据。如果在Hive中不想看到Hudi的默认系统字段，可以在Hive中创建一个外表指向Hudi的目录。

您也可以在Hive中查看Hudi表结构的信息。

```
desc formatted h0;
```

返回如下信息。

```
OK
# col_name              data_type             comment
_hoodie_commit_time   string
_hoodie_commit_seqno  string
_hoodie_record_key    string
_hoodie_partition_path  string
_hoodie_file_name     string
id                    bigint
name                  string
price                 double
# Detailed Table Information
Database:             default
Owner:                root
CreateTime:           Thu Jun 24 16:50:48 CST 2021
LastAccessTime:       UNKNOWN
Retention:            0
Location:             hdfs://emr-header-1.cluster-23****:9000/user/hive/warehouse/h0
Table Type:           MANAGED_TABLE
Table Parameters:
  last_commit_time_sync 20210624165059
  numFiles              1
  spark.sql.create.version  2.4.7
  spark.sql.sources.provider  hudi
  spark.sql.sources.schema.numParts 1
  spark.sql.sources.schema.part.0 {\"type\":\"struct\",\"fields\":[{\"name\":\"_hoodie_commit_time\",\"type\":\"str
ing\",\"nullable\":true,\"metadata\":{}},{\"name\":\"_hoodie_commit_seqno\",\"type\":\"string\",\"nullable\":true,\
"metadata\":{}},{\"name\":\"_hoodie_record_key\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}},{\"name\":\
"_hoodie_partition_path\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}},{\"name\":\"_hoodie_file_name\",\"
type\":\"string\",\"nullable\":true,\"metadata\":{}},{\"name\":\"id\",\"type\":\"long\",\"nullable\":true,\"metadat
a\":{}},{\"name\":\"name\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}},{\"name\":\"price\",\"type\":\"do
uble\",\"nullable\":true,\"metadata\":{}}]}
  totalSize             434752
  transient_lastDdlTime 1624524648
# Storage Information
SerDe Library:        org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe
InputFormat:          org.apache.hudi.hadoop.HoodieParquetInputFormat
OutputFormat:         org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat
Compressed:           No
Num Buckets:          -1
Bucket Columns:       []
Sort Columns:         []
Storage Desc Params:
  path                  hdfs://emr-header-1.cluster-234173:9000/user/hive/warehouse/h0
  serialization.format  1
Time taken: 0.204 seconds, Fetched: 40 row(s)
```

6.2.23.6.4. 通过Hive作业处理TableStore数据

本文介绍如何在E-MapReduce中通过Hive作业来处理TableStore中的数据。

前提条件

确保将实例部署在E-MapReduce集群相同的VPC环境下。

步骤一：创建Hadoop集群

1. 登录阿里云E-MapReduce控制台。
2. 创建Hadoop集群，详情请参见创建集群。

步骤二：获取JAR包并上传到Hadoop集群

1. 获取环境依赖的JAR包。

JAR包	获取方法
emr-tablestore-X.X.X.jar	Maven库中下载：emr-tablestore。
hadoop-lzo-X.X.X-SNAPSHOT.jar	登录Hadoop集群的emr-header-1主机，在*/opt/apps/ecm/service/hadoop/x.x.x-x.x.x/package/hadoop-x.x.x-x.x.x/lib/*下获取JAR包。
hive-exec-X.X.X.jar	登录Hadoop集群的emr-header-1主机，在*/opt/apps/ecm/service/hive/x.x.x-x.x.x/package/apache-hive-x.x.x-x.x.x-bin/lib/*下获取JAR包。
joda-time-X.X.X.jar	登录Hadoop集群的emr-header-1主机，在*/opt/apps/ecm/service/hive/x.x.x-x.x.x/package/apache-hive-x.x.x-x.x.x-bin/lib/*下获取JAR包。
tablestore-X.X.X-jar-with-dependencies.jar	下载EMR SDK相关的依赖包：进入tablestore链接，选择相应的版本，下载*tablestore-X.X.X-jar-with-dependencies.jar*。

> **说明**
>
> - *X.X.X*表示JAR包的具体版本号；*x.x.x-x.x.x*表示文件目录，具体需要根据实际集群中的版本来修改这个JAR包。例如目录是*3.1.1-1.1.6*的，那么就是JAR包是*hive-exec-3.1.1.jar*。
>
> ```
> [root@emr-header-1 ~]# cd /opt/apps/ecm/service/hive/
> [root@emr-header-1 hive]# ll
> total 4
> drwxr-xr-x 3 root root 4096 Jul  1 11:57 3.1.1-1.1.6
> [root@emr-header-1 hive]# cd /opt/apps/ecm/service/hive/3.1.1-1.1.6/package/apache-hive-3.1.1-1.1.6-bin/lib
> [root@emr-header-1 lib]#
> ```
>
> - 登录Hadoop集群的emr-header-1主机的步骤，可参见步骤二的子步骤2 ~ 步骤二的子步骤4。

2. 在**集群管理**页面，单击已创建的Hadoop集群的集群ID。
3. 在左侧导航树中选择**主机列表**，然后在右侧查看Hadoop集群中emr-header-1主机的IP信息。
4. 在SSH客户端中新建一个命令窗口，登录Hadoop集群的emr-header-1主机。
5. 上传所有JAR包到emr-header-1节点的某个目录下。
6. 拷贝所有JAR包到emr-header-1节点的*/opt/apps/extra-jars/*目录。
7. 通过**scp**命令将所有JAR包拷贝到worker节点（所有worker节点均需要）的*/tmp*目录。

```
su hadoop
scp <file> emr-worker-1:/tmp
```

8. 登录worker节点，拷贝所有JAR包到worker节点的*/opt/apps/extra-jars/*目录。

> **说明** 所有worker节均需要进行此操作。

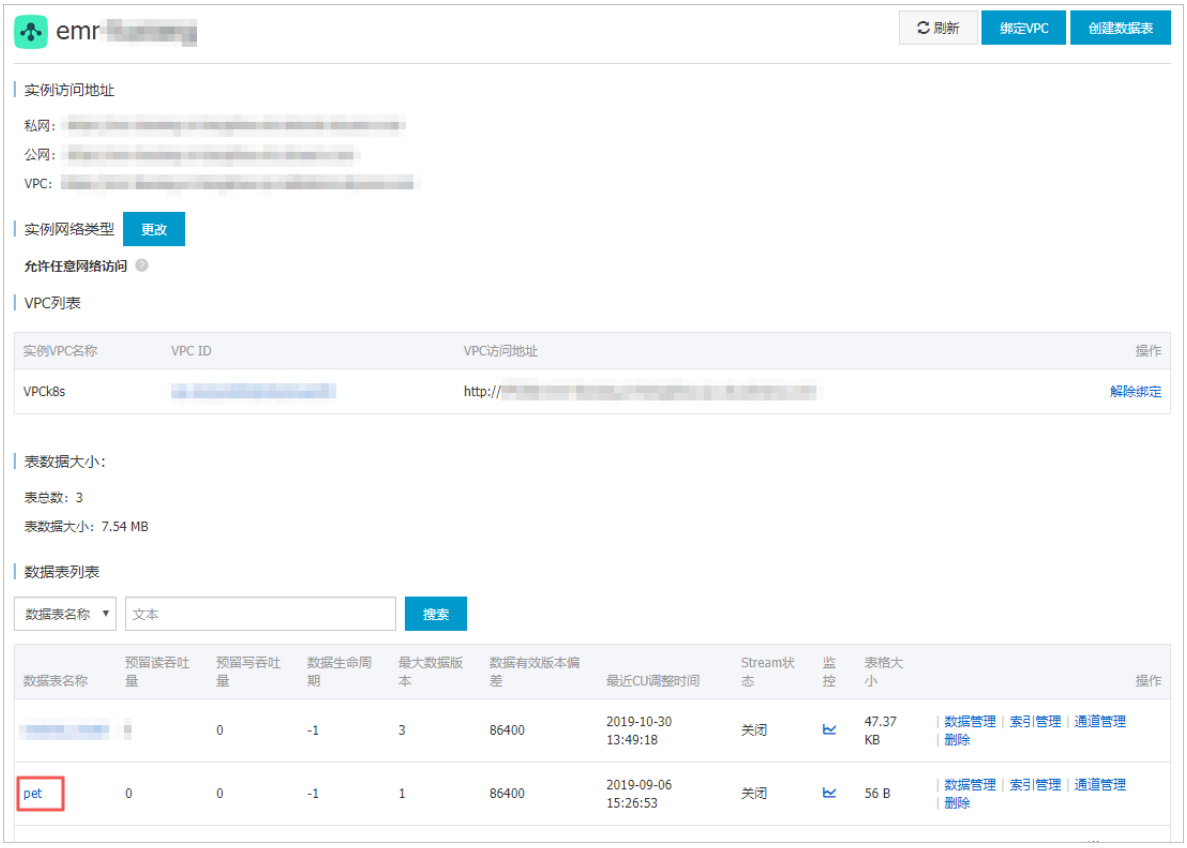
步骤三：重启Hive服务

1. 返回阿里云E-MapReduce控制台。
2. 在**集群管理**页面，单击已创建的Hadoop集群的集群ID。
3. 在**服务列表**中，单击Hive所在行的 ··· 图标，并在弹出框中单击**重启所有组件**。

步骤四：配置Table存储

1. 在TableStore上创建表格，具体请参见创建数据表。

   创建好后如下截图。



步骤五：处理TableStore数据

1. 创建表格。

```
CREATE EXTERNAL TABLE pet(name STRING, owner STRING, species STRING, sex STRING, birth STRING, death STRING)
STORED BY 'com.aliyun.openservices.tablestore.hive.TableStoreStorageHandler'
WITH SERDEPROPERTIES(
    "tablestore.columns.mapping"="name,owner,species,sex,birth,death")
TBLPROPERTIES (
    "tablestore.endpoint"="https://XXX.cn-beijing.ots-internal.aliyuncs.com",
    "tablestore.access_key_id"="LTAIbUa7OYIO****",
    "tablestore.access_key_secret"="u2yOdykvSNXiPChyoqbCoawZnt****",
    "tablestore.table.name"="pet");
```

- 当回显信息提示 `OK` 时，表示表格创建成功。
- 当提示创建失败时， 检查配置信息是否正确：
  - 是：提交工单。
  - 否：检查所有节点*/opt/apps/extra-jars/*目录下JAR包是否全部拷贝，如果已经全部拷贝请提交工单处理；没有全部拷贝，请依据步骤二的子步骤1中JAR包进行拷贝。

2. 向表中插入数据。

```
INSERT INTO pet VALUES("Fluffy", "Harold", "cat", "f", "1993-02-04", null);
```

当回显信息提示 `OK` 时，表示数据插入成功。

3. 执行查询操作。

```
hive> SELECT * FROM pet;
```

```
OK
Fluffy  Harold  cat  f  1993-02-04  NULL
Time taken: 0.23 seconds, Fetched: 3 row(s)
```

当回显信息包含 `OK` 时，表示查询成功。

6.2.23.6.5. 在EMR集群运行TPC-DS Benchmark

TPC-DS是大数据领域最为知名的Benchmark标准。阿里云E-MapReduce多次刷新TPC-DS官方最好成绩，并且是唯一一个通过认证的可运行TPC-DS 100 TB的大数据系统。本文介绍如何在EMR集群完整运行TPC-DS的99个SQL，并得到最佳的性能体验。

背景信息

TPC-DS是全球最知名的数据管理系统评测基准标准化组织TPC（事务性管理委员会）制定的标准规范，并由TPC管理测试结果的发布。TPC-DS官方工具只包含SQL生成器以及单机版数据生成工具，并不适合大数据场景，所以本文教程中使用的工具和集群信息如下：

- Hive TPC-DS Benchmark测试工具。

  该工具是业界最常用的测试工具，是由Hortonworks公司开发，支持使用Hive和Spark运行TPC-DS以及TPC-H等Benchmark。

- EMR集群版本为EMR-4.8.0。

  Hive TPC-DS Benchmark测试工具是基于Hortonworks HDP 3版本开发的，对应的Hive版本是3.1，所以最适合运行该工具的EMR集群版本为EMR-4.x。本文教程使用的是EMR-4.8.0版本，EMR-4.8.0及之后的版本均可运行该教程。

操作流程

- 步骤一：创建EMR集群和下载TPC-DS Benchmark工具
- 步骤二：编译并打包数据生成器
- 步骤三：生成并加载数据
- 步骤四：运行TPC-DS SQL

步骤一：创建EMR集群和下载TPC-DS Benchmark工具

1. 创建EMR-4.8.0集群，具体操作步骤，请参见创建集群。

   在创建集群时，请关注如下配置信息：

   - **集群类型**：选择**Hadoop**。
   - **实例规格**：如果想获得最佳性能，Core实例推荐使用大数据型或本地SSD。如果想能用小规模数据快速完成所有流程，Core实例也可以选择4 vCPU 16 GiB规格的通用型实例。

     > 注意 根据您选择运行的数据集确定集群规模，确保Core实例的数据盘总容量大于数据集规模的三倍。数据集相关信息，请参见步骤三：生成并加载数据。

   - **元数据选择**：推荐使用**数据湖元数据**。
   - **挂载公网**：开启。

2. 通过SSH方式连接EMR集群，具体操作步骤，请参见登录集群。
3. 安装Git和Maven。

```
sudo yum install -y git maven
```

4. 下载TPC-DS Benchmark工具。
   - 通过Git下载。

```
git clone https://github.com/hortonworks/hive-testbench.git
```

> **注意** 中国内地（大陆）地域的节点访问GitHub较慢，如果下载失败，可直接本地下载，将ZIP文件上传到EMR集群，并解压缩。

- 本地下载，将ZIP文件上传到EMR集群，并解压缩。

  具体操作步骤如下：

  a. 下载hive-testbench-hdp3.zip文件。

  b. 修改下载的*hive-testbench-hdp3.zip*文件名称为*hive-testbench.zip*。

  c. 上传ZIP文件到EMR集群。本步骤以本地操作系统为Linux为例介绍，操作命令如下：

  ```
  scp hive-testbench.zip root@xx.xx.xx.xx:/root/
  ```

  > **说明** xx.xx.xx.xx为EMR集群公网IP地址。获取方式，请参见获取主节点的公网IP地址。

  d. 在EMR集群解压缩上传的ZIP文件。

  ```
  unzip hive-testbench.zip
  ```

步骤二：编译并打包数据生成器

1. （可选）配置阿里云镜像。

   在中国内地（大陆）地区加速Maven编译，可以使用阿里云镜像。使用阿里云镜像，编译并打包数据生成器的耗时为2min~3min。

   i. 执行如下命令，新建文件目录。

   ```
   mkdir -p ~/.m2/
   ```

   ii. 执行如下命令，将Maven配置文件拷贝到新文件目录下。

   ```
   cp /usr/share/maven/conf/settings.xml ~/.m2/
   ```

   iii. 在*~/.m2/settings.xml*文件中添加镜像信息，具体内容如下：

   ```
   <mirror>
       <id>aliyun</id>
       <mirrorOf>central</mirrorOf>
       <name>Nexus aliyun</name>
       <url>http://maven.aliyun.com/nexus/content/groups/public</url>
   </mirror>
   ```

2. 切换到*hive-testbench*目录。

   ```
   cd hive-testbench
   ```

3. 使用TPC-DS工具集进行编译并打包数据生成器。

   ```
   ./tpcds-build.sh
   ```

步骤三：生成并加载数据

1. 设置数据规模SF（Scale Factor）。

   SF单位相当于GB，所以SF=1相当于1 GB，SF=100相当于100 GB，SF=1000相当于1 TB，以此类推。本步骤示例采用小规模数据集，推荐使用SF=3。具体命令如下：

   ```
   SF=3
   ```

   > **注意** 请确保数据盘总大小是数据集规模的3倍以上，否则后续流程中会出现报错情况。

2. 检查并清理Hive数据库。

   i. 检查Hive数据库是否存在。

   ```
   hive -e "desc database tpcds_bin_partitioned_orc_$SF"
   ```

ii. （可选）清理已经存在的Hive数据库。

> **注意** 如果Hive数据库tpcds_bin_partitioned_orc_$SF已经存在，需要执行下面的命令清理数据库，否则后续流程会报错。如果不存在，则跳过该步骤。

```
hive -e "drop database tpcds_bin_partitioned_orc_$SF cascade"
```

3. 配置Hive服务地址。

   *tpcds-setup.sh*脚本默认配置的Hive服务地址与EMR集群环境不一致，所以需要将脚本中HiveSever的地址替换为EMR集群中的Hive服务地址。具体命令如下：

   ```
   sed -i 's/localhost:2181\/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=hiveserver2?tez.queue.name=default/emr-header-1:10000\//' tpcds-setup.sh
   ```

   脚本默认配置的Hive服务地址为：`jdbc:hive2://localhost:2181/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=hiveserver2?tez.queue.name=default`。通过上述命令替换后的Hive服务地址为：`jdbc:hive2://emr-header-1:10000/`。

4. 生成并加载数据。

   在SF=3时，该步骤耗时为40min~50min。如果运行正常，TPC-DS数据表将会加载到tpcds_bin_partitioned_orc_$SF数据库中。生成的数据可以保存在HDFS和OSS，针对这两种场景具体的操作如下：

   - 使用HDFS存储数据的场景。

     EMR集群默认将TPC-DS Table数据保存在HDFS上。执行如下命令生成并加载数据：

     ```
     ./tpcds-setup.sh $SF
     ```

     ```
     TPC-DS text data generation complete.
     Loading text data into external tables.
     Optimizing table date_dim (1/24).
     Optimizing table time_dim (2/24).
     Optimizing table item (3/24).
     Optimizing table customer (4/24).
     Optimizing table customer_demographics (5/24).
     Optimizing table household_demographics (6/24).
     Optimizing table customer_address (7/24).
     Optimizing table store (8/24).
     Optimizing table promotion (9/24).
     Optimizing table warehouse (10/24).
     Optimizing table ship_mode (11/24).
     Optimizing table reason (12/24).
     Optimizing table income_band (13/24).
     Optimizing table call_center (14/24).
     Optimizing table web_page (15/24).
     Optimizing table catalog_page (16/24).
     Optimizing table web_site (17/24).
     Optimizing table store_sales (18/24).
     Optimizing table store_returns (19/24).
     Optimizing table web_sales (20/24).
     Optimizing table web_returns (21/24).
     Optimizing table catalog_sales (22/24).
     Optimizing table catalog_returns (23/24).
     Optimizing table inventory (24/24).
     Loading constraints
     Data loaded into database tpcds_bin_partitioned_orc_4.
     ```

   - 使用OSS存储数据的场景。

     通过EMR存储和计算分离的架构能力，可以很方便的做到将数据保存在OSS。具体操作步骤如下：

     a. 使用Hive命令修改数据库路径。

     ```
     hive --hivevar SF=$SF
     create database if not exists tpcds_bin_partitioned_orc_${SF};
     alter database tpcds_bin_partitioned_orc_${SF} set location 'oss://<bucket-name>/warehouse/tpcds_bin_partitioned_orc_${SF}.db';
     ```

     <bucket-name>需要改成和EMR集群在同一地域的OSS Bucket名称。

     b. 生成并加载据。

     ```
     cd ~/hive-testbench
     ./tpcds-setup.sh $SF
     ```

     > **说明** 执行以上命令后，生成的数据就直接保存在OSS上了。如果您同时使用了数据湖构建（DLF）来保存Hive表的元数据，数据生成后，您可以随时释放当前的EMR集群，并在同一地域的其他EMR集群上再次查询当前生成的TPC-DS数据集。

5. 获取Hive表统计信息。

推荐使用Hive SQL ANALYZE命令获取Hive表统计信息，可以加快后续SQL的查询速度。此步骤在SF=3时，耗时为20min~30min。

```
hive -f ./ddl-tpcds/bin_partitioned/analyze.sql \
    --hiveconf hive.execution.engine=tez \
    --database tpcds_bin_partitioned_orc_$SF
```

步骤四：运行TPC-DS SQL

本步骤分别介绍如何使用Hive和Spark运行TPC-DS SQL。

- 使用Hive运行TPC-DS SQL。
  i. 通过以下命令执行单SQL。

  TPC-DS SQL共有99个文件都放在*sample-queries-tpcds*工作目录下（包括*query10.sql*和*query11.sql*等文件）。在SF=3时，所有的SQL都可以在5min内返回结果。

  > **注意** 因为TPC-DS Query和数据都是随机生成，所以部分SQL查询返回结果数为0属于正常现象。

  ```
  cd sample-queries-tpcds
  hive --database tpcds_bin_partitioned_orc_$SF
  set hive.execution.engine=tez;
  source query10.sql;
  ```

  ii. 利用工具包中的脚本顺序执行99个完整SQL。具体命令如下：

  ```
  cd ~/hive-testbench
  # 生成一个Hive配置文件，并指定Hive执行引擎为Tez。
  echo 'set hive.execution.engine=tez;' > sample-queries-tpcds/testbench.settings
  ./runSuite.pl tpcds $SF
  ```

  ```
  [root@emr-header-1 hive-testbench]# ./runSuite.pl tpcds $SF
  filename,status,time,rows
  query10.sql,success,42,34
  query11.sql,success,75,100
  query12.sql,success,33,100
  query13.sql,success,45,1
  query14.sql,success,193,100
  query14.sql,success,193,100
  query15.sql,success,37,100
  query16.sql,success,63,1
  query17.sql,success,63,4
  query18.sql,success,56,100
  query19.sql,success,38,100
  query2.sql,success,59,2513
  query20.sql,success,35,100
  query21.sql,success,32,100
  query22.sql,success,88,100
  query23.sql,success,146,1
  query24.sql,success,118,2
  query26.sql,success,45,100
  query27.sql,success,47,1
  query28.sql,success,45,1
  query29.sql,success,69,3
  query3.sql,success,38,39
  query30.sql,success,41,100
  query31.sql,success,55,226
  query32.sql,success,32,1
  query33.sql,success,48,100
  query35.sql,success,59,100
  query36.sql,success,43,100
  query37.sql,success,40,1
  query38.sql,success,62,1
  ```

- 使用Spark运行TPC-DS SQL。

  TPC-DS工具集中包含Spark SQL用例，用例位于*spark-queries-tpcds*目录下，可以使用 `spark-sql` 或者 `spark-beeline` 等命令行工具执行这些SQL。本步骤以Spark Beeline工具连接Spark ThriftServer为例，介绍如何使用Spark运行TPC-DS SQL来查询步骤三生成的TPC-DS数据集。

  > **说明** EMR Spark支持HDFS和OSS等多种存储介质保存的数据表，也支持数据湖构建（DLF）元数据。

  i. 使用Spark Beeline ANALYZE命令获得Hive表统计信息，加快后续SQL查询速度。

  ```
  cd ~/hive-testbench
  spark-beeline -u jdbc:hive2://emr-header-1:10001/tpcds_bin_partitioned_orc_$SF \
    -f ./ddl-tpcds/bin_partitioned/analyze.sql
  ```

  ii. 切换到Spark SQL用例所在的文件目录。

  ```
  cd spark-queries-tpcds/
  ```

iii. 通过以下命令执行单个SQL。

```
spark-beeline -u jdbc:hive2://emr-header-1:10001/tpcds_bin_partitioned_orc_$SF -f q1.sql
```

iv. 通过脚本顺序执行99个SQL。

TPC-DS工具集中没有包含批量执行Spark SQL的脚本，所以本步骤提供一个简单脚本供参考。

```
for q in `ls *.sql`; do
  spark-beeline -u jdbc:hive2://emr-header-1:10001/tpcds_bin_partitioned_orc_$SF -f $q > $q.out
done
```

> 注意
>
> - SQL列表中*q30.sql*文件存在列名c_last_review_date_sk错写为c_last_review_date的情况，所以该SQL运行失败属于正常现象。
> - 通过脚本顺序执行99个Spark SQL的时候，如果出现报错情况，解决方案请参见常见问题。

常见问题

Q：通过脚本顺序执行99个Spark SQL的时候报错，怎么解决？

A：Spark ThriftServer服务的默认内存不适合较大规模数据集测试，如果在测试过程中出现Spark SQL作业提交失败，原因可能是Spark ThriftServer出现Out Of Memory异常。针对这种情况的解决方法为调整Spark服务配置spark_thrift_daemon_memory的值后重启ThriftServer服务。具体操作步骤如下：

1. 进入Spark服务页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务 > Spark**。
2. 调整服务配置spark_thrift_daemon_memory的值。
   i. 在Spark服务页面，单击**配置**页签。
   ii. 在**配置搜索**区域，输入spark_thrift_daemon_memory，单击图标。
   iii. 根据使用的数据集规模调整对应的数值。
   您可以将默认的1g调整为2g或者更大的数值。
   iv. 单击右上角的**保存**。
   v. 在**确认修改**对话框中，输入执行原因，单击**确定**。
3. 重启Spark ThriftServer。
   i. 在Spark服务页面，选择右上角**操作 > 重启ThriftServer**。
   ii. 在**执行集群操作**对话框中，输入**执行原因**，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

6.2.23.6.6. 通过JDBC连接HiveServer2来访问Hive数据

本文介绍如何通过JDBC连接HiveServer2访问Hive数据。适用于无法通过Hive Client和HDFS访问Hive数据的场景。

前提条件

- 已对Hive进行权限配置，详情请参见Hive配置。
- 因为HiveServer2默认不校验用户和密码，所以当您需要用户和密码认证时，请进行用户认证配置，详情请参见如何设置HiveServer2的认证方式为LDAP？。

背景信息

JDBC连接HiveServer2的方法如下：

- Beeline：通过HiveServer2的JDBC客户端进行连接。
- Java：编写Java代码进行连接。

> 说明 E-MapReduce集群中，Hue通过HiveServer2方式来访问Hive数据。

在E-MapReduce集群中，HiveServer2的JDBC连接地址如下：

- 标准集群：*jdbc:hive2://emr-header-1:10000*
- 高安全集群：*jdbc:hive2://${master1_fullhost}:10000/;principal=hive/${master1_fullhost}@EMR.$id.COM*

Beeline客户端连接HiveServer2

1. 登录集群主节点，详情请参见登录集群。
2. 执行如下命令，进入Beeline客户端。

```
[root@emr-header-1 ~]# beeline
```

返回如下信息。

```
Beeline version 1.2.1.spark2 by Apache Hive
```

3. 执行如下命令，连接HiveServer2。

```
beeline> !connect jdbc:hive2://emr-header-1:10000
```

返回如下信息。

```
Connecting to jdbc:hive2://emr-header-1:10000
```

4. 输入用户名和密码。

```
Enter username for jdbc:hive2://emr-header-1:10000: your_username
Enter password for jdbc:hive2://emr-header-1:10000: your_password
```

返回如下信息。

```
log4j:WARN No such property [datePattern] in org.apache.log4j.RollingFileAppender.
18/12/17 18:09:16 INFO Utils: Supplied authorities: emr-header-1:10000
18/12/17 18:09:16 INFO Utils: Resolved authority: emr-header-1:10000
18/12/17 18:09:16 INFO HiveConnection: Will try to open client transport with JDBC Uri: jdbc:hive2://emr-header-1:10000
Connected to: Apache Hive (version 2.3.3)
Driver: Hive JDBC (version 1.2.1.spark2)
Transaction isolation: TRANSACTION_REPEATABLE_READ
```

5. 查询Hive数据。

```
0: jdbc:hive2://emr-header-1:10000> select * from emrusers limit 10;
+------------------+-------------------+------------------+--------------------+--------------+--+
| emrusers.userid  | emrusers.movieid  | emrusers.rating  | emrusers.unixtime  | emrusers.dt  |
+------------------+-------------------+------------------+--------------------+--------------+--+
| 196              | 242               | 3                | 881250949          | 2018100102   |
| 186              | 302               | 3                | 891717742          | 2018100102   |
| 22               | 377               | 1                | 878887116          | 2018100102   |
| 244              | 51                | 2                | 880606923          | 2018100102   |
| 166              | 346               | 1                | 886397596          | 2018100102   |
| 298              | 474               | 4                | 884182806          | 2018100102   |
| 115              | 265               | 2                | 881171488          | 2018100102   |
| 253              | 465               | 5                | 891628467          | 2018100102   |
| 305              | 451               | 3                | 886324817          | 2018100102   |
| 6                | 86                | 3                | 883603013          | 2018100102   |
+------------------+-------------------+------------------+--------------------+--------------+--+
10 rows selected (1.455 seconds)
```

Java连接HiveServer2

在执行本操作前，确保您已安装Java环境和Java编程工具，并且已配置环境变量。

1. 在*pom.xml*文件中配置项目依赖（hadoop-common和hive-jdbc）。

本示例新增的项目依赖如下：

```
<dependencies>
        <dependency>
            <groupId>org.apache.hive</groupId>
            <artifactId>hive-jdbc</artifactId>
            <version>1.2.1</version>
        </dependency>
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-common</artifactId>
            <version>2.7.2</version>
        </dependency>
</dependencies>
```

2. 编写代码，连接HiveServer2并操作Hive数据。

   示例如下。

```
import java.sql.*;
public class App
{
    private static String driverName = "org.apache.hive.jdbc.HiveDriver";
    public static void main(String[] args) throws SQLException {
        try {
            Class.forName(driverName);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        Connection con = DriverManager.getConnection(
                "jdbc:hive2://emr-header-1:10000", "root", "");   //代码打包后，运行JAR包的环境需要在hosts文件中把emr-header-1映射到E-MapReduce集群的公网IP地址（或内网IP地址）。
        Statement stmt = con.createStatement();
        String sql = "select * from emrusers limit 10";
        ResultSet res = stmt.executeQuery(sql);
        while (res.next()) {
            System.out.println(res.getString(1) + "\t" + res.getString(2));
        }
    }
}
```

3. 打包项目工程（即生成JAR包），并上传JAR包至运行环境。

   > **注意** JAR包的运行需要依赖hadoop-common和hive-jdbc。如果运行环境的环境变量中未包含这两个依赖包，则您需要下载依赖包并配置环境变量，或者直接一起打包这两个依赖包。运行JAR包时，如果缺少这两个依赖包，则会提示以下错误：
   >
   > - 缺失*hadoop-common*：提示java.lang.NoClassDefFoundError: org/apache/hadoop/conf/Configuration
   > - 缺失*hive-jdbc*：提示java.lang.ClassNotFoundException: org.apache.hive.jdbc.HiveDriver

   本示例生成的JAR包为*emr-hiveserver2-1.0.jar*，上传到E-MapReduce集群的**emr-header-1**。

4. 运行JAR包，测试是否可正常运行。

   > **注意** 运行JAR包的服务器与E-MapReduce集群需要在同一个VPC和安全组下，并且网络可达。如果两者的VPC不同或网络环境不同，则需要通过公网地址访问，或先使用网络产品打通两者的网络，再通过内网访问。网络连通性测试方法：
   >
   > - 公网：**telnet** *emr-header-1的公网IP地址 10000*
   > - 内网：**telnet** *emr-header-1的内网IP地址 10000*

```
[root@emr-header-1 xxx-test]# java -jar emr-hiveserver2-1.0.jar
```

   返回如下信息。

```
log4j:WARN No appenders could be found for logger (org.apache.hive.jdbc.Utils).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
196     242
186     302
22      377
244     51
166     346
298     474
115     265
253     465
305     451
6       86
```

6.2.23.7. 常见问题

本文汇总了Hive使用时的常见问题。

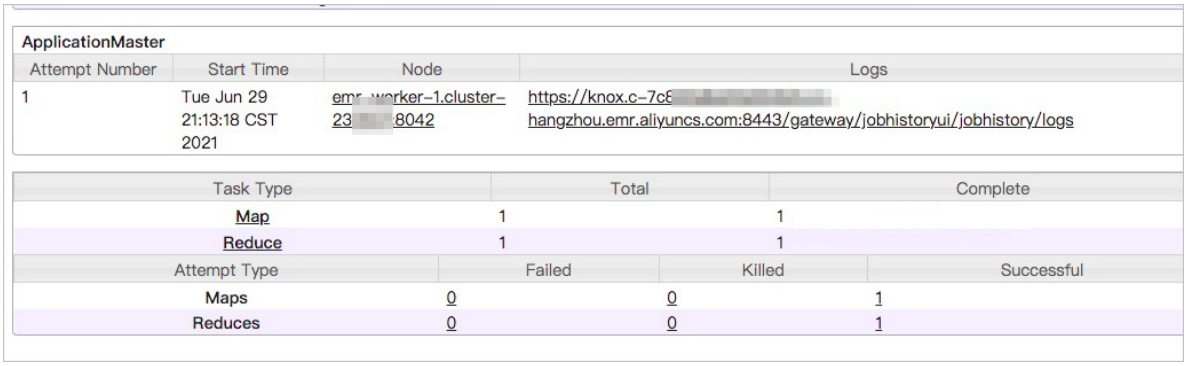
- 作业长时间处于等待状态，如何处理？
- Map端是否读取了小文件？
- Reduce Task任务耗时，是否出现了数据倾斜？

作业长时间处于等待状态，如何处理？

您可以通过以下步骤定位问题：

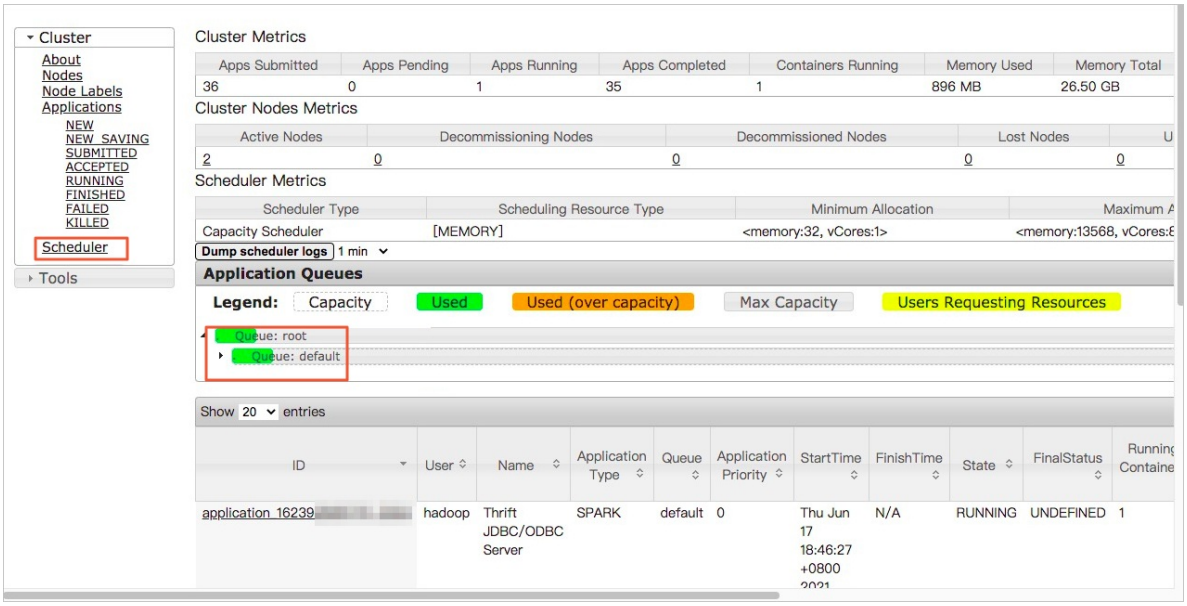
1. 在EMR控制台的**访问链接与端口**页面，单击YARN UI所在行的链接。
2. 单击Application ID。
3. 单击**Tracking URL**的链接。

   可以看到有多个作业处于等待状态。



4. 在左侧导航中，单击**Scheduler**。

即可进入队列，您可以看一下当前队列的繁忙程度，来分析是因为队列中没有空闲资源，还是当前任务确实比较耗时。如果是队列资源紧张，您可以考虑切换到空闲队列，否则需要优化代码。

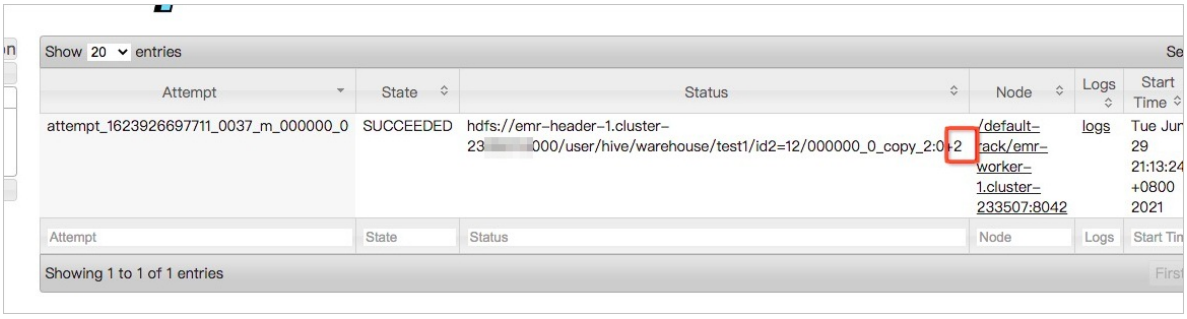


Map端是否读取了小文件？

您可以通过以下步骤定位问题：

1. 在EMR控制台的**访问链接与端口**页面，单击YARN UI所在行的链接。
2. 单击Application ID。

   进入Map Task的详情页面，可以看到每个Map Task读取的数据量，如下图所示，读取的数据量是2个字节记录。如果大部分的Map Task读取的文件量都比较小，就需要考虑小文件合并。

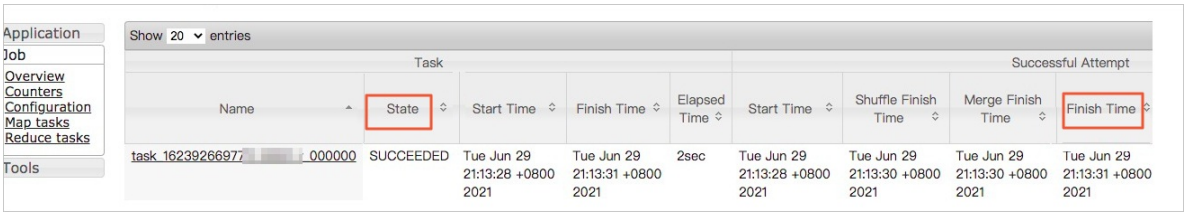


   您也可以通过查看Map Task的Log，获取更多的信息。

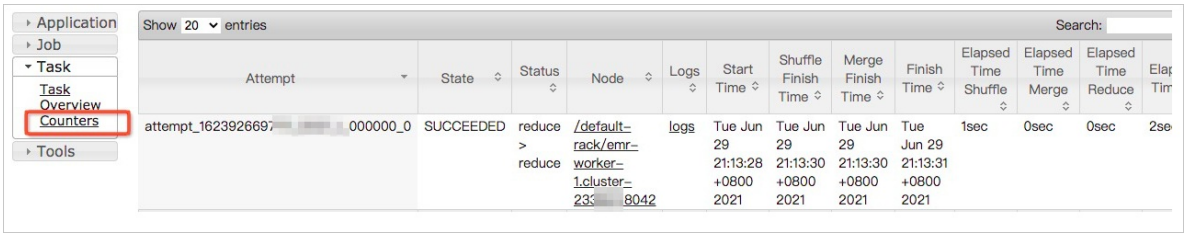
Reduce Task任务耗时，是否出现了数据倾斜？

您可以通过以下步骤定位问题：

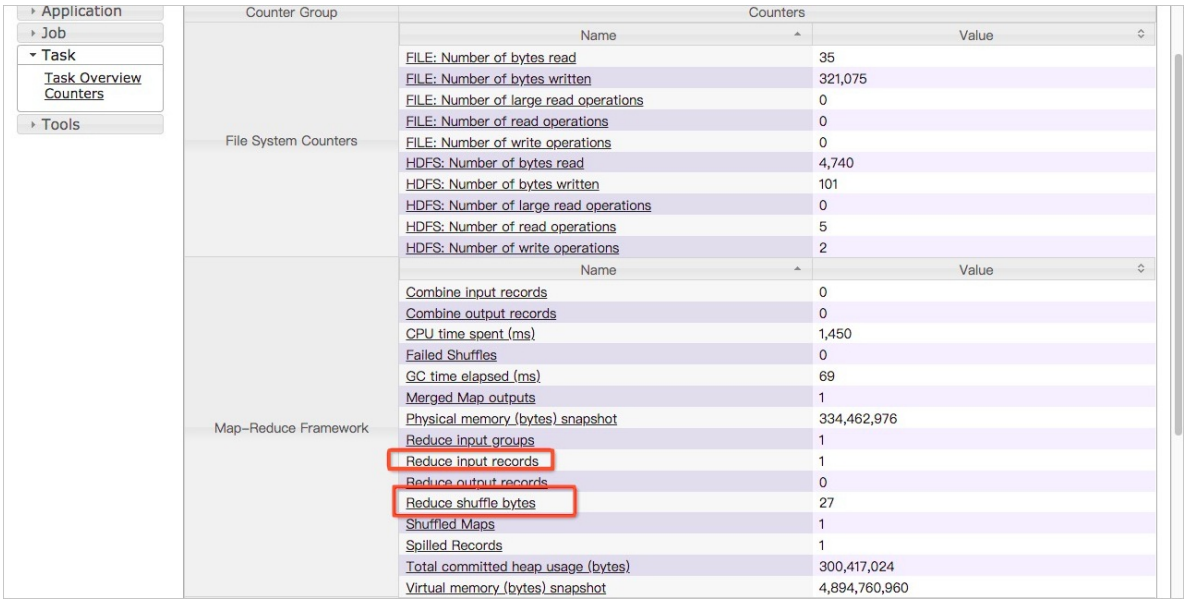
1. 在EMR控制台的**访问链接与端口**页面，单击YARN UI所在行的链接。
2. 单击Application ID。
3. 在Reduce Task列表页面，按照完成时间逆序排序，找出Top耗时的Reduce Task任务。



4. 单击Task的Name链接。
5. 在Task详情页面，单击左侧的Counters。



查看当前Reduce Task中Reduce Input bytes和Reduce shuffle bytes的信息，如果比其他的Task处理的数据量大很多，则说明出现了倾斜问题。



6.2.24. Impala

6.2.24.1. 概述

Impala为存储在Apache Hadoop中的数据，提供了高性能和低延迟的SQL查询。 使用Impala，您可以通过SELECT、JOIN和聚合函数实时查询存储在HDFS或HBase中的数据。

背景信息

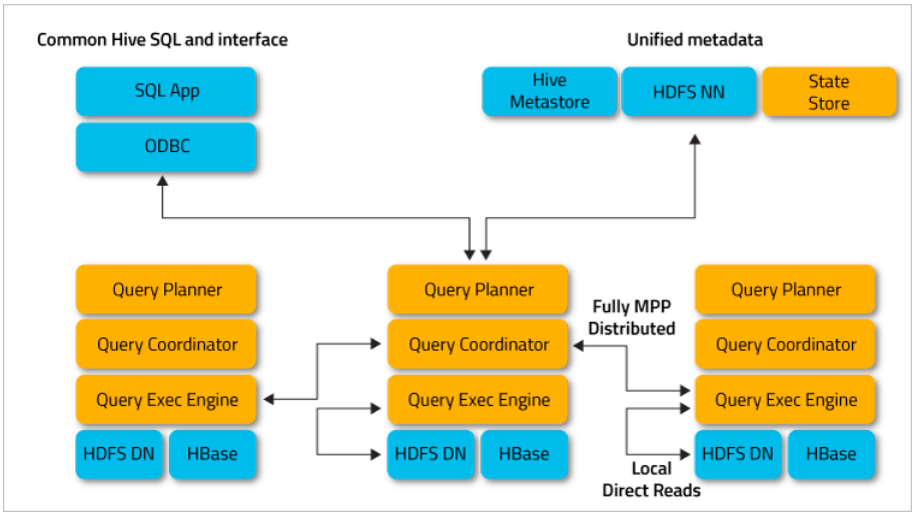
Impala使用与Apache Hive相同的元数据、SQL语法（Hive SQL）和ODBC驱动程序等，为面向批处理或实时查询提供了一个熟悉且统一的平台。

注意事项

如果使用Impala组件，请勿直接通过系统文件删除hive表分区目录，请使用Impala或者Hive命令删除，否则会导致该表不可用。

优点

为了避免延迟，Impala没有使用MapReduce，而是使用分布式查询引擎直接访问数据，该引擎与RDBMS中的查询引擎相似，其性能比Hive快了几个数量级，具体取决于查询和配置的类型。Impala部署结构图如下所示。



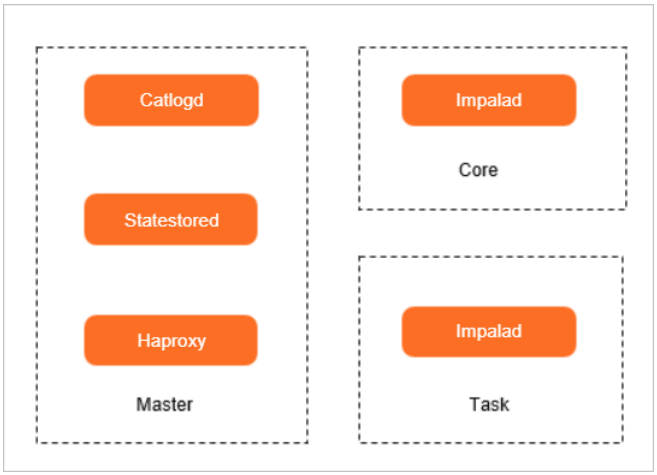
Impala相对于Hadoop上SQL查询，优点如下：

- 由于在数据节点上进行了本地处理，因此避免了网络的限制。
- 由于无需进行昂贵的数据格式转换，因此不会产生任何费用。
- 可以使用单个、开放和统一的元数据存储。
- 所有数据均可立即查询，无需等待ETL（Extract-transform-load）。
- 所有硬件均用于Impala查询以及MapReduce。
- 仅需单个计算机池即可扩展。

Impala的详细信息，请参见Apache Impala。

架构

E-MapReduce中Impala的架构如下图。



Impala组件如下：

- Impalad

  部署在Core节点和Task节点，允许扩容和缩容。

  Impala的核心组件是运行在各个节点上的Impala Daemon，进程名为Impalad，负责读取和写入数据文件，接收从 `impala-shell` 命令、Hue、JDBC或ODBC等接口发送的查询语句，并行查询语句和分发工作任务到集群的各个Impala节点上，同时负责将本地计算好的查询结果发送回协调器节点（Coordinator Node）。

- Statestored

  部署在Master节点的header-1机器。

  Statestore服务对应的进程名为Statestored，负责管理集群中所有Impalad进程的健康状态，并将状况结果转发到所有Impalad进程。当某一个Impalad进程由于节点异常、网络异常或软件问题等导致节点不可用时，StateStore确保将状况结果通知其他Impalad进程，当有新的查询请求时，Impalad进程将不会发送查询请求到该不可用的节点。

- Catalogd

  部署在Master节点的header-1机器。

  Catalogd负责将每个Impalad进程上的元数据变动同步到集群内其他Impalad进程。由于所有的请求都是通过StateStore进程传递的，所以建议StateStore和Catalog运行在同一个节点上。

- Haproxy

  部署在Master节点，负责代理连接集群中各Impalad节点，转发查询请求。

入门指导

- Impala SQL作业配置的详情请参见Impala SQL作业配置。
- 在Zeppelin中使用Impala的详情请参见Zeppelin概述。

6.2.24.2. 开发指南

6.2.24.2.1. 使用JDBC连接Impala

本文为您介绍如何使用JDBC连接Impala。

前提条件

已创建E-MapReduce的Hadoop集群，并且选择了Impala服务。详情请参见创建集群。

操作步骤

1. 通过SSH方式连接集群，详情请参见登录集群。
2. 下载Impala JDBC驱动。

   在Cloudera官网下载Impala JDBC驱动并将其添加到至*/usr/lib/hive-current/lib/*目录下。

   下载Impala JDBC驱动地址：Impala JDBC Connector。

   Impala支持通过Hive2 JDBC连接，也支持使用Cloudera Impala JDBC Connector连接。

3. 执行以下命令，进入Beeline客户端。

   ```
   beeline
   ```

   返回信息如下图所示，表示成功进入Beeline客户端。

   ```
   [root@emr-header-1 ~]# beeline
   SLF4J: Class path contains multiple SLF4J bindings.
   SLF4J: Found binding in [jar:file:/opt/apps/ecm/service/hive/3.1.2-hadoop3.1-1.1.2/package/apache-hive-3.1.2-hadoop3.1-1.1.2-bin/lib
   cLoggerBinder.class]
   SLF4J: Found binding in [jar:file:/opt/apps/ecm/service/hadoop/3.2.1-1.1.0/package/hadoop-3.2.1-1.1.0/share/hadoop/common/lib/slf4j-
   der.class]
   SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
   SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
   Beeline version 3.1.2 by Apache Hive
   beeline>
   ```

4. 使用JDBC连接Impala。

   > **说明** 如果Impala开启LDAP认证，则只能使用Impala JDBC连接Impala。

   - 执行如下命令，使用Hive JDBC连接Impala。

     ```
     !connect jdbc:hive2://emr-header-1:21050/;auth=noSasl;
     ```

     根据提示信息输入LDAP的用户名和密码。

   - 执行如下命令，使用Impala JDBC连接Impala。

     ```
     !connect jdbc:impala://emr-header-1:21050/default;AuthMech=3;UID=<user>;PWD=<password>;
     ```

     代码中的`<user>`需要替换为LDAP的用户名。`<password>`需要替换为LDAP用户的密码，获取方式请参见管理用户。

   连接成功后，您便可以使用SQL查询。

6.2.24.2.2. Impala命令行工具

本文为您介绍如何在E-MapReduce上使用命令行工具操作Impala控制台。

前提条件

已创建E-MapReduce的Hadoop集群，并且选择了Impala服务。详情请参见创建集群。

操作步骤

1. 通过SSH方式连接集群。

   详情请参见登录集群。

2. 执行如下命令，进入Impala控制台。

   - 普通集群：

     ```
     impala-shell
     ```

   - 高安全集群：

     ```
     impala-shell -k
     ```

     > **说明** 连接Impala的账户已通过安全认证，详情请参见兼容MIT Kerberos认证。

   返回信息包含如下信息时，表示成功进入Impala控制台。

   ```
   Welcome to the Impala shell.
   ```

   在连接Impala前，您可以执行 `impala-shell --help` 命令，获取控制台的帮助。

   ```
   -h, --help            show this help message and exit
   -i IMPALAD, --impalad=IMPALAD
                         <host:port> of impalad to connect to
                         [default: emr-header-1.cluster-20****:2****]
   -q QUERY, --query=QUERY
                         Execute a query without the shell [default: none]
   -f QUERY_FILE, --query_file=QUERY_FILE
                         Execute the queries in the query file, delimited by ;.
                         If the argument to -f is "-", then queries are read
                         from stdin and terminated with ctrl-d. [default: none]
   -k, --kerberos        Connect to a kerberized impalad [default: False]
   -o OUTPUT_FILE, --output_file=OUTPUT_FILE
                         If set, query results are written to the given file.
                         Results from multiple semicolon-terminated queries
                         will be appended to the same file [default: none]
   -B, --delimited       Output rows in delimited mode [default: False]
   --print_header        Print column names in delimited mode when pretty-
                         printed. [default: False]
   --output_delimiter=OUTPUT_DELIMITER
                         Field delimiter to use for output in delimited mode
                         [default: \t]
   -s KERBEROS_SERVICE_NAME, --kerberos_service_name=KERBEROS_SERVICE_NAME
                         Service name of a kerberized impalad [default: impala]
   -V, --verbose         Verbose output [default: True]
   -p, --show_profiles   Always display query profiles after execution
                         [default: False]
   --quiet               Disable verbose output [default: False]
   -v, --version         Print version information [default: False]
   -c, --ignore_query_failure
                         Continue on query failure [default: False]
   -r, --refresh_after_connect
                         Refresh Impala catalog after connecting
                         [default: False]
   -d DEFAULT_DB, --database=DEFAULT_DB
                         Issues a use database command on startup
                         [default: none]
   -l, --ldap            Use LDAP to authenticate with Impala. Impala must be
                         configured to allow LDAP authentication.
                         [default: False]
   -u USER, --user=USER  User to authenticate with. [default: root]
   --ssl                 Connect to Impala via SSL-secured connection
                         [default: False]
   --ca_cert=CA_CERT     Full path to certificate file used to authenticate
                         Impala's SSL certificate. May either be a copy of
                         Impala's certificate (for self-signed certs) or the
                         certificate of a trusted third-party CA. If not set,
                         but SSL is enabled, the shell will NOT verify Impala's
                         server certificate [default: none]
   --config_file=CONFIG_FILE
                         Specify the configuration file to load options. The
   ```

```
                        following sections are used: [impala],
                        [impala.query_options]. Section names are case
                        sensitive. Specifying this option within a config file
                        will have no effect. Only specify this as an option in
                        the commandline. [default: /root/.impalarc]
  --live_summary        Print a query summary every 1s while the query is
                        running. [default: False]
  --live_progress       Print a query progress every 1s while the query is
                        running. [default: False]
  --auth_creds_ok_in_clear
                        If set, LDAP authentication may be used with an
                        insecure connection to Impala. WARNING: Authentication
                        credentials will therefore be sent unencrypted, and
                        may be vulnerable to attack. [default: none]
  --ldap_password_cmd=LDAP_PASSWORD_CMD
                        Shell command to run to retrieve the LDAP password
                        [default: none]
  --var=KEYVAL          Defines a variable to be used within the Impala
                        session. Can be used multiple times to set different
                        variables. It must follow the pattern "KEY=VALUE", KEY
                        starts with an alphabetic character and contains
                        alphanumeric characters or underscores. [default:
                        none]
  -Q QUERY_OPTIONS, --query_option=QUERY_OPTIONS
                        Sets the default for a query option. Can be used
                        multiple times to set different query options. It must
                        follow the pattern "KEY=VALUE", KEY must be a valid
                        query option. Valid query options  can be listed by
                        command 'set'. [default: none]
```

3. （可选）执行 `quit;` ，您可以退出Impala控制台。

6.2.24.3. 高阶使用

6.2.24.3.1. 管理LDAP认证

服务开启LDAP认证功能后，访问服务时需要提供LDAP身份认证（LDAP用户名和密码），以便于提升服务的安全性。开启LDAP功能对接的LDAP为E-MapReduce自带的OpenLDAP。开启LDAP认证的功能可以方便您使用LDAP认证，避免了复杂的配置过程。本文为您介绍如何一键开启和关闭LDAP认证。

前提条件

已创建Hadoop集群，详情请参见创建集群。

使用限制

EMR-3.34.0及后续版本或EMR-4.8.0及后续版本的Hadoop集群，支持一键开启LDAP认证。

开启LDAP认证

> 注意 Impala开启LDAP认证后，Hue访问Impala需要进行额外的配置，请参见Hue连接开启LDAP认证的引擎。

1. 进入Impala页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务 > Impala**。
2. 开启LDAP认证。
   i. 在Impala服务页面，选择右上角的**操作 > 开启LDAP认证**。
   ii. 在**执行集群操作**对话中，单击**确认**。
3. 单击上方的**查看操作历史**。
   直至操作状态显示**成功**。
4. 重启Impala。
   i. 在Impala服务页面，选择右上角的**操作 > 重启All Components**。
   ii. 在**执行集群操作**对话中，输入**执行原因**，单击**确定**。

iii. 在**确认**对话中，单击**确定**。

访问Impala

开启LDAP认证后，当您访问Impala时需要提供LDAP认证凭据。

1. 通过SSH方式连接集群，请参见登录集群。
2. 您可以通过以下两种方式访问Impala。
   - impala-shell：

     ```
     impala-shell -l -u <user> --auth_creds_ok_in_clear
     ```

     说明 user为LDAP的用户名。开启LDAP认证后，访问Impala需要提供LDAP的密码，获取方式请参见管理用户。
   - JDBC：

     开启LDAP认证后，JDBC访问Impala需要提供LDAP认证凭据，同时需要前往Cloudera官网下载Impala JDBC驱动并将其添加到至*/usr/lib/hive-current/lib/*目录下。

     下载Impala JDBC驱动地址：Impala JDBC Connector。

     ```
     !connect jdbc:impala://emr-header-1:21050/default;AuthMech=3;UID=<user>;PWD=<password>;
     ```

关闭LDAP认证

1. 进入Impala页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务 > Impala**。
2. 关闭LDAP认证。
   i. 在Impala服务页面，选择右上角的**操作 > 关闭LDAP认证**。
   ii. 在**执行集群操作**对话中，单击**确认**。
3. 单击上方的**查看操作历史**。

   直至操作状态显示**成功**。
4. 重启Impala。
   i. 在Impala服务页面，选择右上角的**操作 > 重启All Components**。
   ii. 在**执行集群操作**对话中，输入**执行原因**，单击**确定**。
   iii. 在**确认**对话中，单击**确定**。

6.2.24.4. 常见问题

本文汇总了使用Impala时的常见问题。

- Impala 3.4使用JDBC查询Impala 10秒提示会话超时，该怎么办？
- Impala无法找到Hive新增的表，该怎么办？
- Impala写入Hive表数据的时候，是否可以修改写入文件的owner？
- Impalad节点内存配置表示什么意思？
- 如何限制单条查询语句消耗内存的大小？
- 如何提升非JOIN语句的查询效率？

Impala 3.4使用JDBC查询Impala 10秒提示会话超时，该怎么办？

您可以在Impala控制台通过命令设置FETCH_ROWS_TIMEOUT_MS参数为0，指定会话永不过期。命令示例如下所示。

```
jdbc:impala://impala-hive.ymt.io:21050/ymtcube;FETCH_ROWS_TIMEOUT_MS=0
```

Impala无法找到Hive新增的表，该怎么办？

在Impala之外操作元数据后，您可以在Impala控制台通过使用命令INVALIDATE METADATA，以刷新全库或者某个表的元数据。

Impala写入Hive表数据的时候，是否可以修改写入文件的owner？

使用Impala写入的表文件的owner默认为Impala，暂不支持修改。

Impalad节点内存配置表示什么意思？

您可以在EMR控制台Impala服务的**配置**页签，在搜索区域搜索**mem_limit**参数，默认值为80%，表示允许使用本机内存的80%来计算。

如何限制单条查询语句消耗内存的大小？

您可以在Impala控制台通过命令设置mem_limit参数来限制单条查询语句消耗内存的大小。例如，设置mem_limit参数为10 GB。

如何提升非JOIN语句的查询效率？

对于单条没有JOIN的查询语句，您可以在Impala控制台通过命令设置mt_dop参数为n以提升并发度，其中n为每台机器的并发度。

6.2.25. Zookeeper

6.2.25.1. 概述

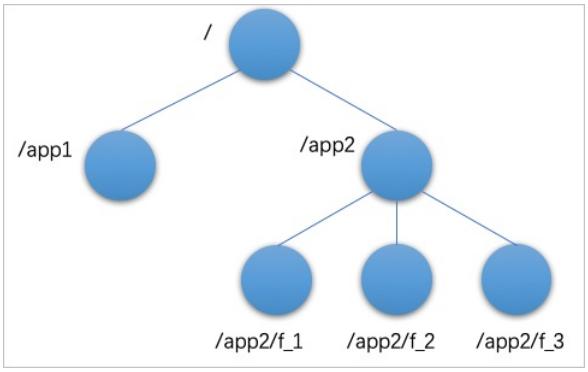
ZooKeeper是一个分布式、高可用性的协调服务。ZooKeeper提供分布式配置服务、同步服务和命名注册等功能。

基本原理

一个ZooKeeper集群需要由奇数个（2N+1）节点构成，通过内部选举协议选出一个Leader节点，其余为Follower节点。写入数据时，由Leader节点负责统一协调写请求，至少（N+1）个节点投票成功才能确定本次数据写入成功，因此至少有（N+1）个存活的节点才能保证ZooKeeper整体服务可用。当Leader节点异常退出时，ZooKeeper集群会重新发起选举，选出新的Leader节点，保证整体服务的高可用。

数据组织

ZooKeeper的数据组织方式与标准文件系统类似，组织成类似文件树的结构，在ZooKeeper中使用znode（ZooKeeper node）来描述文件，与标准文件系统不同的是，znode并不区分目录或者文件的概念，每个znode都可以存储数据。



ZooKeeper作为一个协调服务，znode主要用来存储协调性数据，例如，服务状态信息和配置信息等，不应把ZooKeeper作为文件系统来存储大量数据。

6.2.25.2. 基础使用

本文为您介绍如何在E-MapReduce上查看ZooKeeper的节点信息，以及连接Zookeeper。

前提条件

已创建集群，并且选择了ZooKeeper服务。详情请参见创建集群。

> **说明** 高可用集群默认选择了Zookeeper服务。

关于ZooKeeper的更多介绍，请参见ZooKeeper Getting Started Guide。

连接Zookeeper

您可以使用命令行工具*zkCli.sh*连接ZooKeeper，*zkCli.sh*在 `/usr/lib/zookeeper-current/bin/` 下。

连接方式如下。

```
/usr/lib/zookeeper-current/bin/zkCli.sh -server IP地址/主机名:port
```

`IP地址/主机名` 的获取方式请参见查看节点信息， `port` 为端口。

连接Zookeeper示例如下。

```
/usr/lib/zookeeper-current/bin/zkCli.sh -server emr-header-1:2181
```

连接成功后，即可使用 `/ls` 命令查看Zookeeper上znode的信息。

查看节点信息

1. 进入集群详情页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
2. 在左侧导航栏中，选择**集群服务 > ZooKeeper**。
3. 单击**部署拓扑**。
   您可以查看ZooKeeper的节点信息。

6.2.25.3. 常见问题

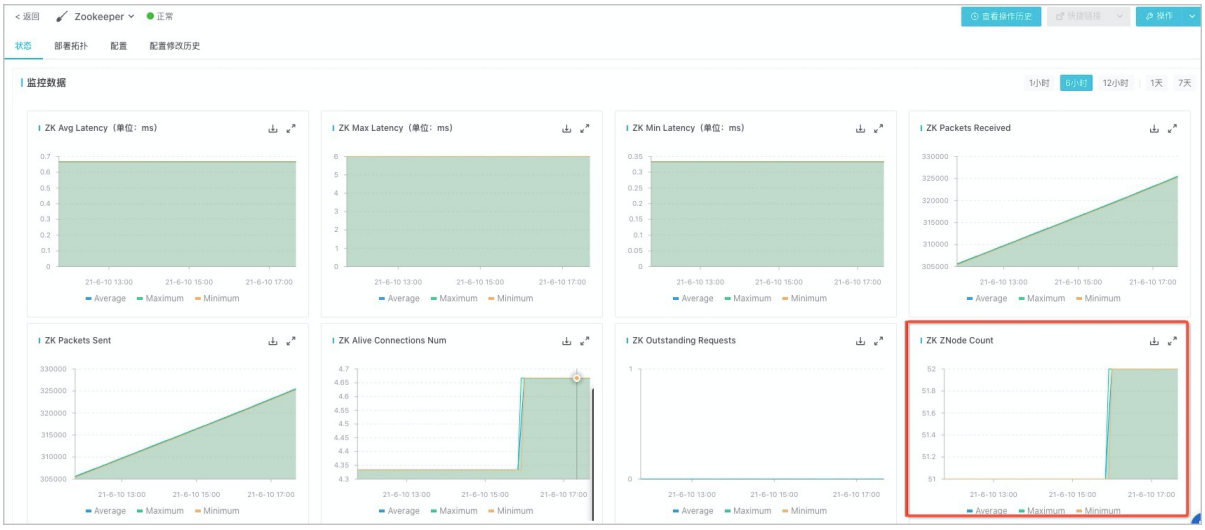
本文汇总了使用Zookeeper时的常见问题。

- ZooKeeper服务不稳定，发生异常重启等情况，怎么办？
- 报错信息包含Too many connections，怎么办？
- 如何平滑迁移ZooKeeper的数据目录？

ZooKeeper服务不稳定，发生异常重启等情况，怎么办？

造成服务不稳定的情况可能有很多，最常见的情况是znode数量过大或者snapshot过大，由于ZooKeeper将所有的znode维护在内存中，并且需要在节点间进行数据同步，因此过大的znode数量或者容量会对服务稳定性造成影响。ZooKeeper的定位是一个分布式协调服务，不能把ZooKeeper当作一个文件系统来使用，通常，znode数量应尽量保持在10万以下，snapshot大小应在800 MB以下。

- 您可以在ZooKeeper服务的**状态**页签，查看znode数量监控。



- 查看snapshot大小。
  i. 您可以先在ZooKeeper服务的**配置**页签，搜索并查看**zk_data_dirs**的参数值，获取Zookeeper的数据目录。

  ii. 在ZooKeeper数据目录下查看snapshot的大小。

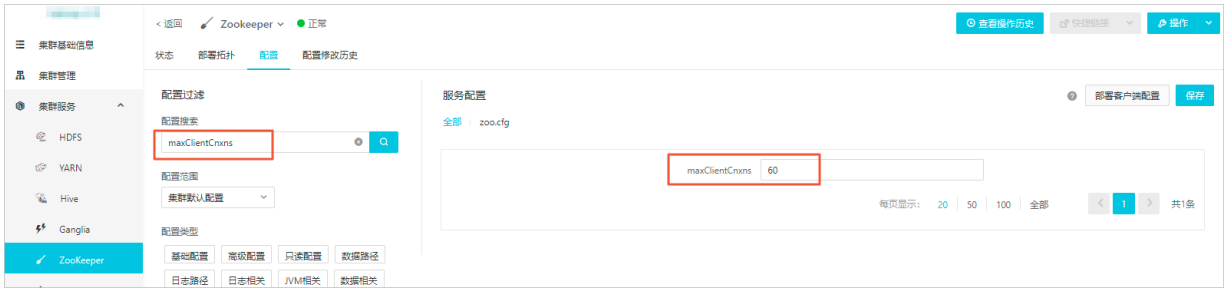
```
[root@emr-header-3 ~]# ls -lrt /mnt/disk1/zookeeper/version-2/snapshot*
-rw-rw-r-- 1 hadoop hadoop 610 Jun  6 15:48 /mnt/disk1/zookeeper/version-2/snapshot.0
```

如果出现znode数量或snapshot过大的情况，需要排查znode分布情况，避免相关上层应用对ZooKeeper的过度使用。

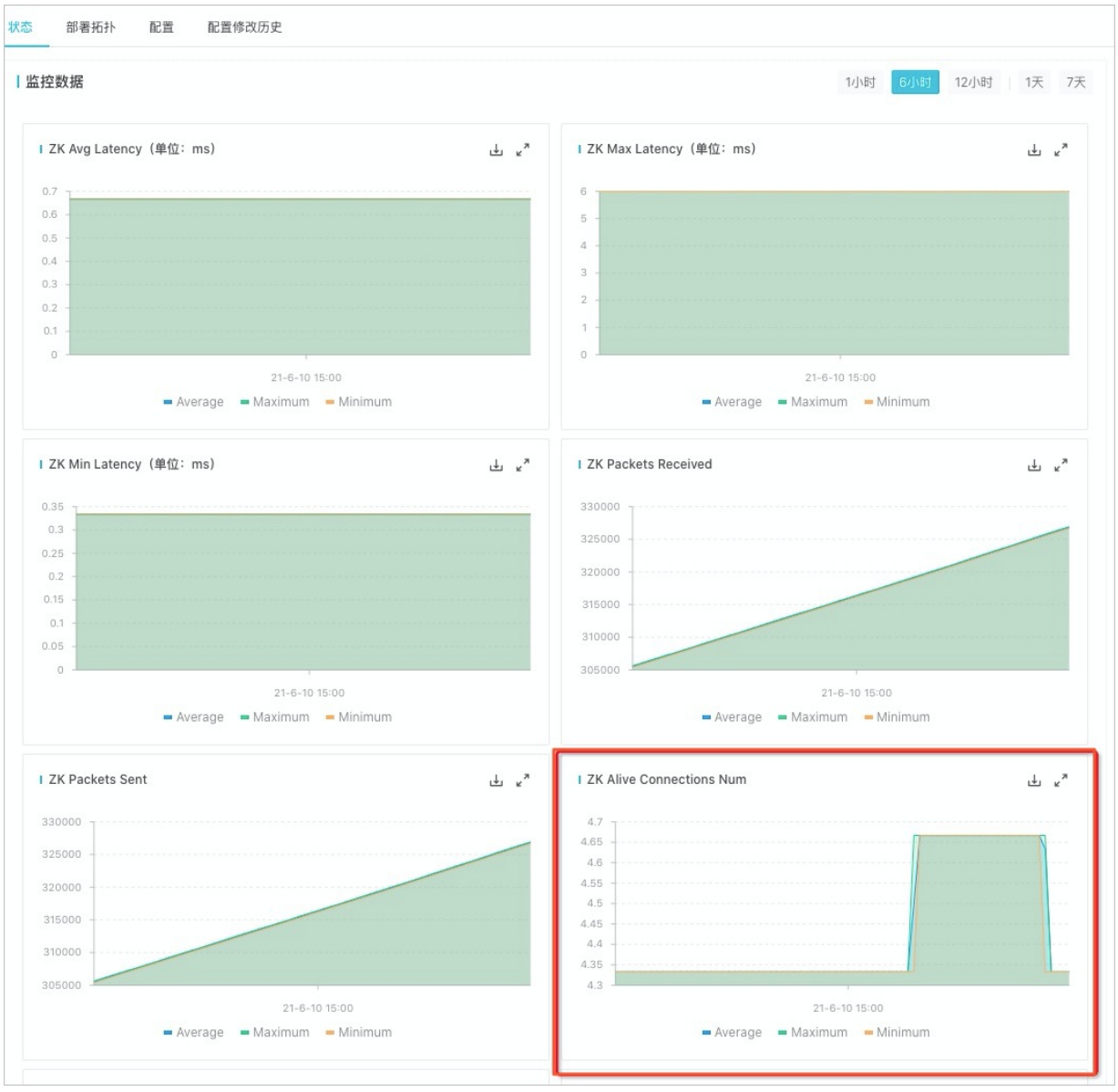
报错信息包含Too many connections，怎么办？

当使用ZooKeeper客户端的进程出现类似Too many connections的报错时，可能是因为ZooKeeper客户端的连接数超过了限制。

您可以在ZooKeeper服务的**配置**页签，搜索并修改**maxClientCnxns**参数值，该配置限制了每个ZooKeeper节点和单个客户端IP之间的连接数，您可以适当调大该参数值，然后重启ZooKeeper服务使其生效。



当通过调整**maxClientCnxns**参数值后，仍然无法满足需求时，您可以在ZooKeeper状态页面观察一下连接数监控。如果发生连接数不断上升的情况，您需要排查一下使用ZooKeeper客户端的进程，是否存在ZooKeeper使用问题，例如使用后没有正确释放连接。根据排查的问题进行相应的处理，使得ZooKeeper客户端的进程可以正常运行。



如何平滑迁移ZooKeeper的数据目录？

如果因为磁盘空间或者磁盘性能等问题，希望更改ZooKeeper的数据目录，您可以按照以下步骤逐个节点修改并迁移数据目录，实现不停止ZooKeeper服务的平滑迁移。

> **说明** 例如，将数据目录从*/mnt/disk1/zookeeper*更改为*/mnt/disk2/zookeeper*。集群的emr-worker-2节点为leader，emr-header-1和emr-worker-1节点为follower，迁移时建议先操作follower再操作leader。

1. 修改数据目录并保存配置。
   i. 在ZooKeeper服务的**配置**页签，搜索**zk_data_dirs**参数，修改参数值为新目录/mnt/disk2/zookeeper。

   ii. 单击右上角的**保存**。
   iii. 在**确认修改**对话框中，输入执行原因，单击**确定**。
2. 部署配置。
   i. 在ZooKeeper服务的**配置**页签，单击右上角的**部署客户配置**。
   ii. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。
3. （可选）验证数据目录。
   i. 使用SSH方式登录集群，详情请参见登录集群。
   ii. 执行以下命令，查看*zoo.cfg*配置文件的dataDir参数值。

```
cat /etc/ecm/zookeeper-conf/zoo.cfg
```

   返回信息如下，显示数据目录已更新为新目录。

```
[root@emr-header-1 ~]# cat /etc/ecm/zookeeper-conf/zoo.cfg
clientPort=2181
clientPortAddress=0.0.0.0
maxSessionTimeout=360000
tickTime=2000
initLimit=10
syncLimit=30

dataDir=/mnt/disk2/zookeeper

server.1=emr-header-1.cluster-260[redacted]88
server.2=emr-worker-1.cluster-260[redacted]88
server.3=emr-worker-2.cluster-260[redacted]88
```

4. 停止emr-header-1节点。
   i. 在ZooKeeper服务的**部署拓扑**页签，单击emr-header-1节点操作列的**停止**。
   ii. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。
   iv. 单击上方的**查看操作历史**，直至操作的**状态**显示为**成功**。
5. 目录迁移。
   i. 使用SSH方式登录集群，详情请参见登录集群。
   ii. 执行以下命令，在emr-header-1节点进行目录迁移及相关权限设置。

```
sudo rm -rf /mnt/disk2/zookeeper && sudo cp -rf /mnt/disk1/zookeeper /mnt/disk2/zookeeper && sudo chown hadoop:hadoop -R /mnt/disk2/zookeeper
```

6. 启动emr-header-1节点。
   i. 在ZooKeeper服务的**部署拓扑**页签，单击emr-header-1节点操作列的**启动**。
   ii. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。
   刷新页面，直至emr-header-1节点的健康状态显示为**良好**。
7. 登录emr-worker-1节点，重复执行步骤4~步骤6。

8. 登录emr-worker-2节点，重复执行步骤4~步骤6。

   待所有节点恢复正常时，节点迁移完成。

   > **说明** emr-worker-2节点原本为leader，执行停止操作后会变为follower，leader会切换到另两个节点的任一节点上。

6.2.26. Flink

6.2.26.1. 概述

Flink（VVR）是基于Apache Flink（以下简称Flink）开发的商业版，VVR引擎接口完全兼容Flink开源版本，且提供GeminiStateBackend等高增值功能，以提升作业性能及稳定性。

背景信息

Flink核心是一个流式的数据流执行引擎，其针对数据流的分布式计算提供了数据分布、数据通信以及容错机制等功能。基于流执行引擎，Flink提供了更高抽象层的API以便您编写分布式任务。

Flink（VVR）完全兼容开源Flink，相关内容请参见如下文档：

- DataStream API
- Table API&SQL
- Python API

使用场景

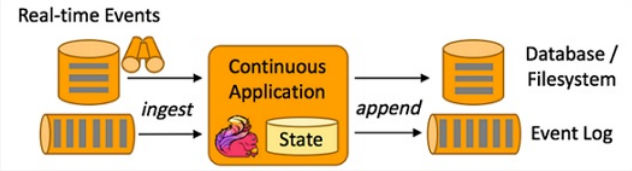
Flink广泛应用于大数据实时化的场景，本文从技术领域和企业应用场景进行介绍。

- 技术领域

  从技术领域的角度，Flink主要用于以下场景：

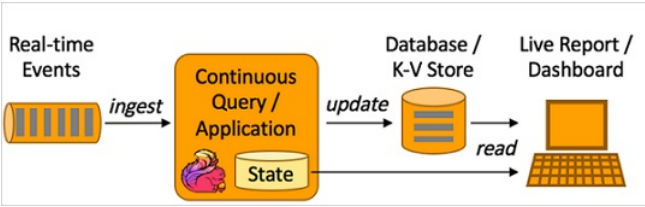
  - 实时ETL（Extract-transform-load）和数据流

    实时ETL和数据流的目的是实时地把数据从A点投递到B点。在投递的过程中可能添加数据清洗和集成的工作，例如实时构建搜索系统的索引和实时数仓中的ETL过程等。

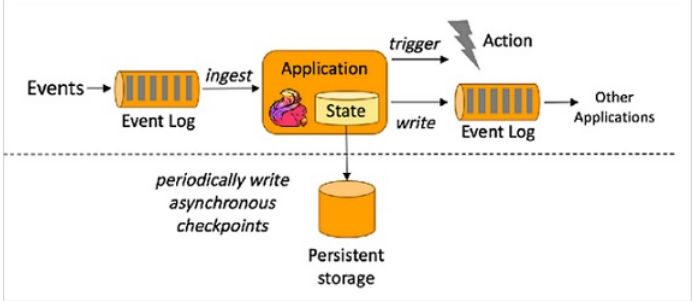
  - 实时数据分析

    实时数据分析指的是根据业务目标，从原始数据中抽取对应信息并整合的过程。例如，查看每天销量前10的商品、仓库平均周转时间、文档平均单击率和推送打开率等。实时数据分析则是上述过程的实时化，通常在终端体现为实时报表或实时大屏。

  - 事件驱动应用

    事件驱动应用是对一系列订阅事件进行处理或作出响应的系统。事件驱动应用通常需要依赖内部状态，例如欺诈检测、风控系统、运维异常检测系统等。当您的行为触发某些风险控制点时，系统会捕获这个事件，并根据您当前和之前的行为进行分析，决定是否对您进行风险控制。

关于Apache Flink的更多介绍，请参见Apache Flink官网。

- 企业应用

  从企业应用的角度，Flink主要用于以下场景：

  - 业务部门：实时风控、实时推荐和搜索引擎的实时索引构建等。
  - 数据部门：实时数仓、实时报表和实时大屏等。
  - 运维部门：实时监控、实时异常检测和预警以及全链路Debug等。

入门指导

- Flink（VVR）的使用请参见Flink（VVR）作业配置。
- 在Zeppelin中使用Flink的详情请参见Zeppelin概述。

6.2.26.2. 基础使用

本文为您介绍如何在E-MapReduce上提交Flink作业以及查看作业。

背景信息

Dataflow集群中的Flink是以YARN模式部署的，在Dataflow集群中，您可以通过两种方式提交Flink作业：

- 通过控制台的**数据开发**进行提交。

  提交Flink作业详情，请参见Flink（VVR）作业配置。

- 通过SSH方式登录Flink模式的Dataflow集群，在命令行中进行提交。

  在基于YARN模式部署的Dataflow集群支持Session模式、Per-Job Cluster模式和Application模式。

模式	描述	特点
Session模式	Seesion模式会根据您设置的资源参数创建一个Flink集群，所有作业都将被提交到这个集群上运行。该集群在作业运行结束之后不会自动释放。 例如，某个作业发生异常，导致一个Task Manager关闭，则其他所有运行在该Task Manager上的作业都会失败。另外由于同一个集群中只有一个Job Manager，随着作业数量的增多，Job Manager的压力会相应增加。	○ 优点：提交作业时，资源分配导致的时间开销相比其他模式较小。 ○ 缺点：由于所有作业都运行在该集群中，会存在对资源的竞争以及作业间的相互影响。 根据以上特点，该模式适合部署需要较短启动时间且运行时间相对较短的作业。
Per-Job Cluster模式	当使用Per-Job Cluster模式时，每次提交一个Flink作业，YARN都会为这个作业新启动一个Flink集群，然后运行该作业。当作业运行结束或者被取消时，该作业所属的Flink集群也会被释放。	○ 优点：作业之间资源隔离，一个作业的异常行为不会影响到其他作业。 因为每个作业都和一个Job Manager一一对应，因此不会出现一个Job Manager因为运行多个Job而导致负载过高的问题。 ○ 缺点：每次运行一个作业都要启动一个专属Flink集群，启动作业的开销更大。 根据以上特点，该模式通常适合运行时间较长的作业。
Application模式	当使用Application模式时，每次提交一个Flink Application（一个Application包含一个或多个作业），YARN都会为这个Application新启动一个Flink集群。当Application运行结束或者被取消时，该Application所属的Flink集群也会被释放。 该模式与Per-Job模式不同的是，Application对应的JAR包中的main()方法会在集群中的Job Manager中被执行。 如果提交的JAR包中包含多个作业，则这些作业都会在该Application所属的集群中执行。	○ 优点：可以减轻客户端提交作业时的负担。 ○ 缺点：每次运行一个Flink Application都要启动一个专属Flink集群，启动Application的时间开销会更大。

  您可以根据需求，选择以下三种模式提交并查看作业：

  - Session模式提交并查看作业
  - Per-Job Cluster模式提交并查看作业
  - Application模式提交并查看作业

前提条件

已创建Flink模式的Dataflow集群，详情请参见创建集群。

Session模式提交并查看作业

1. 通过SSH方式连接集群，详情请参见登录集群。
2. 执行以下命令，启动YARN Session。

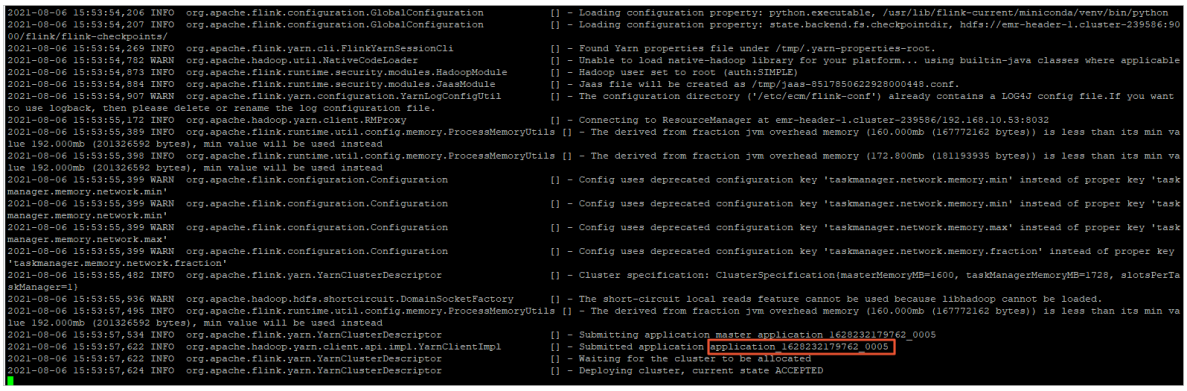
```
yarn-session.sh --detached
```

3. 执行以下命令，提交作业。

```
flink run /usr/lib/flink-current/examples/streaming/TopSpeedWindowing.jar
```

> **说明** 本文使用Flink自身提供的TopSpeedWindowing示例进行介绍，该示例是一个会长时间运行的流作业。

提交成功后，会返回已提交的Flink作业的YARN Application ID。返回如下类似信息。



4. 执行以下命令，查看作业状态。

```
flink list -t yarn-session -Dyarn.application.id=<application_XXXX_YY>
```

> **说明** 本文示例中的 `<application_XXXX_YY>` 为作业运行后返回的Application ID。

您也可以通过Web UI的方式查看作业状态，详情请参见通过Web UI查看作业状态。

Per-Job Cluster模式提交并查看作业

1. 通过SSH方式连接集群，详情请参见登录集群。
2. 执行以下命令，提交作业。

```
flink run -t yarn-per-job --detached /usr/lib/flink-current/examples/streaming/TopSpeedWindowing.jar
```

提交成功后，会返回已提交的Flink作业的YARN Application ID。返回如下类似信息。

您可以执行以下命令，查看作业状态。

```
flink list -t yarn-per-job -Dyarn.application.id=<application_XXXX_YY>
```

> **说明** 本文示例中的 `<application_XXXX_YY>` 为作业运行后返回的Application ID。

您也可以通过Web UI的方式查看作业状态，详情请参见通过Web UI查看作业状态。

Application模式提交并查看作业

1. 通过SSH方式连接集群，详情请参见登录集群。
2. 执行以下命令，提交作业。

```
flink run-application -t yarn-application /usr/lib/flink-current/examples/streaming/TopSpeedWindowing.jar
```

提交成功后，会返回已提交的Flink作业的YARN Application ID。返回如下类似信息。

```
Waiting for response...
------------------ Running/Restarting Jobs -------------------
06.08.2021 15:29:08 : c03f5afe3bacda7d9d7e82e4e03bc53a : CarTopSpeedWindowingExample (RUNNING)
--------------------------------------------------------------
No scheduled jobs.
[root@emr-header-1 ~]# flink run-application -t yarn-application /usr/lib/flink-current/examples/streaming/TopSpeedWindowing.jar
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/apps/ecm/service/flink/1.12-vvr-3.0.2/package/flink-1.12-vvr-3.0.2/lib/log4j-slf4j-impl-2.12.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/apps/ecm/service/hadoop/2.8.5-1.6.4/package/hadoop-2.8.5-1.6.4/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
2021-08-06 15:50:32,597 INFO  org.apache.flink.yarn.cli.FlinkYarnSessionCli                 [] - Found Yarn properties file under /tmp/.yarn-properties-root.
2021-08-06 15:50:32,597 INFO  org.apache.flink.yarn.cli.FlinkYarnSessionCli                 [] - Found Yarn properties file under /tmp/.yarn-properties-root.
2021-08-06 15:50:33,215 WARN  org.apache.flink.yarn.configuration.YarnLogConfigUtil        [] - The configuration directory ('/etc/ecm/flink-conf') already contains a LOG4J config file.If you want
to use logback, then please delete or rename the log configuration file.
2021-08-06 15:50:33,413 INFO  org.apache.hadoop.yarn.client.RMProxy                        [] - Connecting to ResourceManager at emr-header-1.cluster-239586/192.168.10.53:8032
2021-08-06 15:50:33,550 INFO  org.apache.flink.yarn.YarnClusterDescriptor                  [] - No path for the flink jar passed. Using the location of class org.apache.flink.yarn.YarnClusterDescri
ptor to locate the jar
2021-08-06 15:50:33,659 INFO  org.apache.flink.yarn.YarnClusterDescriptor                  [] - Cluster specification: ClusterSpecification(masterMemoryMB=1600, taskManagerMemoryMB=1728, slotsPerTa
skManager=1)
2021-08-06 15:50:34,026 WARN  org.apache.hadoop.hdfs.shortcircuit.DomainSocketFactory      [] - The short-circuit local reads feature cannot be used because libhadoop cannot be loaded.
2021-08-06 15:50:35,353 INFO  org.apache.flink.yarn.YarnClusterDescriptor                  [] - Submitting application master application_1628232179762_0003
2021-08-06 15:50:35,403 INFO  org.apache.hadoop.yarn.client.api.impl.YarnClientImpl        [] - Submitted application application_1628232179762_0003
2021-08-06 15:50:35,404 INFO  org.apache.flink.yarn.YarnClusterDescriptor                  [] - Waiting for the cluster to be allocated
2021-08-06 15:50:35,405 INFO  org.apache.flink.yarn.YarnClusterDescriptor                  [] - Deploying cluster, current state ACCEPTED
2021-08-06 15:50:40,936 INFO  org.apache.flink.yarn.YarnClusterDescriptor                  [] - YARN application has been deployed successfully.
2021-08-06 15:50:40,936 INFO  org.apache.flink.yarn.YarnClusterDescriptor                  [] - Found Web Interface emr-worker-1.cluster-239586:43135 of application 'application_1628232179762_0003'.
```

3. 执行以下命令，查看作业状态。

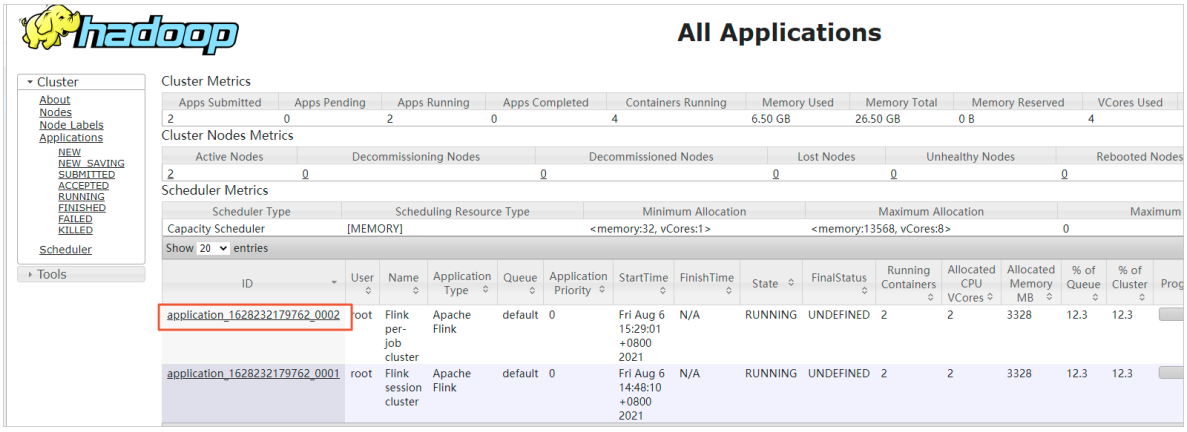
```
flink list -t yarn-application -Dyarn.application.id=<application_XXXX_YY>
```

> **说明** 本文示例中的 `<application_XXXX_YY>` 为作业运行后返回的Application ID。

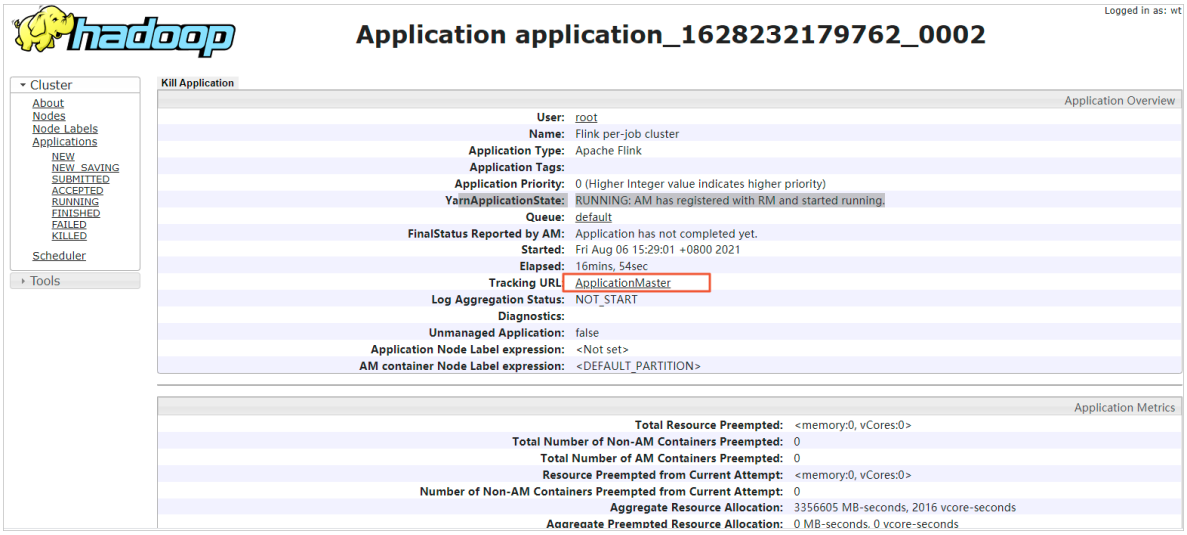
您也可以通过Web UI的方式查看作业状态，详情请参见通过Web UI查看作业状态。

通过Web UI查看作业状态

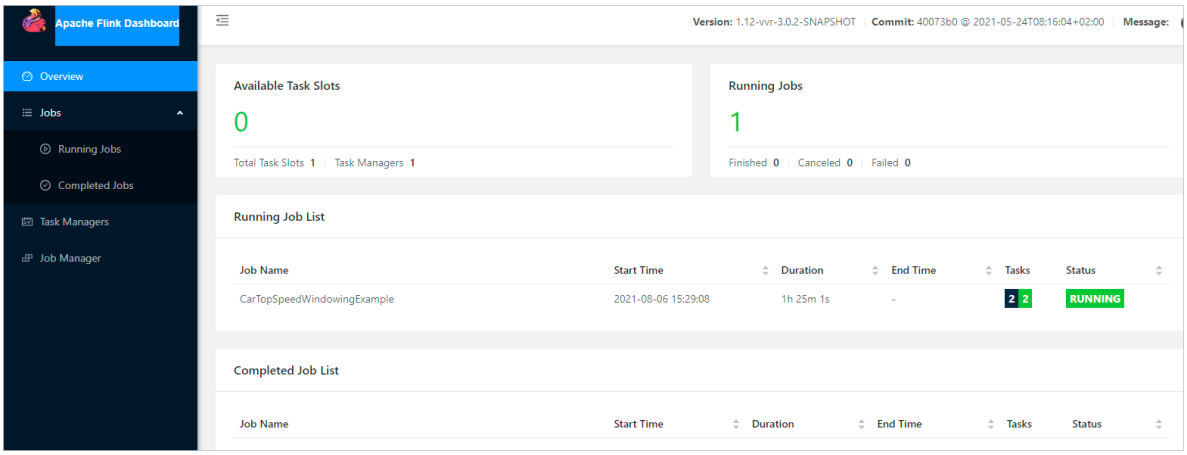
1. 访问Web UI。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏中，单击**访问链接与端口**。
   vi. 在**访问链接与端口**页面，单击YARN UI所在行的链接。
   访问Web UI的详细信息，请参见访问链接与端口。
2. 单击Application ID。



3. 单击Tracking URL的链接。



进入Apache Flink Dashboard页面，即可查看作业的状态。



相关文档

Flink on YARN的更多信息，请参见Apache Hadoop YARN。

6.2.26.3. 开发指南

6.2.26.3.1. Flink SQL参考

Flink SQL是为了简化计算模型、降低您使用Flink门槛而设计的一套符合标准SQL语义的开发语言。

本文通过以下方面，为您介绍Flink SQL的使用方法。

操作	文档
关键字	关键字
创建数据视图	创建数据视图

操作		文档
DDL数据定义语句	创建数据源表	• 日志服务SLS源表 • 消息队列Kafka源表 • 数据总线DataHub源表 • 全量MaxCompute源表 • 增量MaxCompute源表 • 消息队列RocketMQ版源表 • Hologres源表 • 全量Elasticsearch源表 • Postgres的CDC源表（公测中） • MySQL的CDC源表 • Upsert Kafka源表 • Datagen源表
	创建数据结果表	• 日志服务SLS结果表 • 消息队列Kafka结果表 • 数据总线DataHub结果表 • 表格存储Tablestore结果表 • MaxCompute结果表 • 消息队列RocketMQ版结果表 • 云数据库HBase版结果表 • 云数据库Redis版结果表 • Hologres结果表 • 云数据库MongoDB版结果表 • Elasticsearch结果表 • Phoenix5结果表 • 云数据库RDS MySQL结果表 • 云原生数据仓库AnalyticDB MySQL版3.0结果表 • ClickHouse结果表 • print结果表 • Upsert Kafka结果表 • Blackhole结果表 • InfluxDB结果表
	创建数据维表	• 云数据库HBase版维表 • 表格存储Tablestore维表 • MaxCompute维表 • 云数据库Redis版维表 • Elasticsearch维表 • Hologres维表 • 云数据库RDS MySQL维表 • 云原生数据仓库AnalyticDB MySQL版3.0维表 • FileSystem维表
DML数据操作语句（INSERT INTO）		INSERT INTO语句
DQL数据查询语句		DQL数据查询语句
窗口函数		• 概述 • 滚动窗口 • 滑动窗口 • 会话窗口 • OVER窗口

操作		文档
内置函数		• 概述 • 标量函数 • 表值函数 • 聚合函数 • 机器学习函数CLUSTER_SERVING
自定义函数	Java	• 概述 • 自定义标量函数（UDF） • 自定义聚合函数（UDAF） • 自定义表值函数（UDTF）
	Python	• 概述 • 自定义标量函数（UDF） • 自定义聚合函数（UDAF） • 自定义表值函数（UDTF）

6.2.26.3.2. Flink DataStream参考

本文通过以下方面，为您介绍Flink DataStream 的使用方法。

DataFlow集群的Flink DataStream API完全兼容开源的Flink版本，关于Flink DataStream API的详细信息，请参见Flink DataStream API Programming Guide。

上下游存储（Connector）

- 开源Flink的上下游存储，请参见DataStream Connectors。
- DataFlow集群中新增支持的上下游存储，请参见下表。

Connector版本	EMR版本	Connector类型	文档及Demo
1.13-vvr-4.0.10及以上	EMR-3.38.0及以上	ververica-connector-datahub	◦ 文档：DataHub DataStream Connector ◦ Demo：ververica-connector-datahub-demo
		ververica-connector-kafka	◦ 文档：Kafka DataStream Connector ◦ Demo：ververica-connector-kafka-demo
		◦ ververica-connector-odps ◦ ververica-connector-continuous-odps	◦ 文档：MaxCompute DataStream Connector ◦ Demo：ververica-connector-maxcompute-demo
		ververica-connector-mysql-cdc	文档：MySQL CDC DataStream Connector

6.2.26.3.3. Flink Python参考

本文通过以下方面，为您介绍Flink Python的使用方法。

背景信息

DataFlow集群的Flink Python API完全兼容开源的Flink版本，关于Flink Python API的详细信息，请参见Python API。

使用Python依赖

通过以下场景为您介绍如何使用Python依赖：

- 使用自定义的Python虚拟环境
- 使用第三方Python包
- 使用JAR包
- 使用数据文件

使用自定义的Python虚拟环境

- 方式一：在DataFlow集群中的某个节点创建Python虚拟环境

i. 在DataFlow集群的某个节点，准备*setup-pyflink-virtual-env.sh*脚本，其内容如下。

```
set -e
# 创建Python的虚拟环境。
python3.6 -m venv venv
# 激活Python虚拟环境。
source venv/bin/activate
# 准备Python虚拟环境。
pip install --upgrade pip
# 安装PyFlink依赖。
pip install "apache-flink==1.13.0"
# 退出Python虚拟环境。
deactivate
```

ii. 执行以下命令，运行该脚本。

```
./setup-pyflink-virtual-env.sh
```

该命令执行完成后，会生成一个名为*venv*的目录，即为Python 3.6的虚拟环境。您也可以修改上述脚本，安装其他版本的Python虚拟环境。

为了使用该Python虚拟环境，您可以选择将该Python虚拟环境分发到DataFlow集群的所有节点上，也可以在提交PyFlink作业的时候，指定使用该Python虚拟环境，详细信息请参见Command-Line Interface。

- 方式二：在本地开发机创建Python虚拟环境

i. 在本地准备*setup-pyflink-virtual-env.sh*脚本，其内容如下。

```
set -e
# 下载Python 3.7 miniconda.sh脚本。
wget "https://repo.continuum.io/miniconda/Miniconda3-py37_4.9.2-Linux-x86_64.sh" -O "miniconda.sh"
# 为Python 3.7 miniconda.sh脚本添加执行权限。
chmod +x miniconda.sh
# 创建Python的虚拟环境。
./miniconda.sh -b -p venv
# 激活Conda Python虚拟环境。
source venv/bin/activate ""
# 安装PyFlink依赖。
pip install "apache-flink==1.13.0"
# 退出Conda Python虚拟环境。
conda deactivate
# 删除缓存的包。
rm -rf venv/pkgs
# 将准备好的Conda Python虚拟环境打包。
zip -r venv.zip venv
```

ii. 在本地准备*build.sh*脚本，其内容如下。

```
#!/bin/bash
set -e -x
yum install -y zip wget
cd /root/
bash /build/setup-pyflink-virtual-env.sh
mv venv.zip /build/
```

iii. 在CMD命令行中，执行如下命令。

```
docker run -it --rm -v $PWD:/build  -w /build quay.io/pypa/manylinux2014_x86_64 ./build.sh
```

该命令执行完成后，会生成一个名为*venv.zip*的文件，即为Python 3.7的虚拟环境。您也可以修改上述脚本，安装其他版本的Python虚拟环境，或者在虚拟环境中安装所需的第三方Python包。

使用第三方Python包

如果您的第三方Python包是Zip Safe的，可以直接在Python作业中使用，详细信息请参见Python libraries。

如果您的第三方Python包是源码包，且源码包的根目录下存在*setup.py*文件，则这种类型的第三方Python包通常需要先编译才能被使用。您可以选择以下方式编译第三方Python包：

- 在DataFlow集群的某个节点编译第三方Python包。
- 使用*quay.io/pypa/manylinux2014_x86_64*镜像容器中的Python环境来编译第三方Python包，使用该容器编译生成的包兼容绝大多数Linux环境，关于该镜像容器的详细信息，请参见manylinux。

下面以第三方Python包*opencv-python-headless*为例，介绍如何编译和使用该第三方Python包。

1. 编译第三方Python包。

i. 在本地准备*requirements.txt*文件，其内容如下。

```
opencv-python-headless
```

ii. 在本地准备*build.sh*脚本，其内容如下。

```
#!/bin/bash
set -e -x
yum install -y zip
PYBIN=/opt/python/cp37-cp37m/bin
"${PYBIN}/pip" install --target __pypackages__ -r requirements.txt --no-deps
cd __pypackages__ && zip -r deps.zip . && mv deps.zip ../ && cd ..
rm -rf __pypackages__
```

iii. 在CMD命令行中，执行如下命令。

```
docker run -it --rm -v $PWD:/build  -w /build quay.io/pypa/manylinux2014_x86_64 /bin/bash build.sh
```

该命令执行完成后，会生成一个名为*deps.zip*的文件，该文件为编译之后的第三方Python包。您也可以修改*requirements.txt*，安装其他所需的第三方Python包。此外，*requirements.txt*文件中可以指定多个Python依赖。

2. 使用第三方Python包。

关于如何在PyFlink作业中，使用第三方Python包，详情请参见Python Libraries。

使用JAR包

如果您的Flink Python作业中使用了Java类，例如作业中使用了Connector或者Java自定义函数时，则需要指定Connector或者Java自定义函数所在的JAR包，详情请参见JAR Dependencies。

使用数据文件

如果您的Flink Python作业中需要访问数据文件，例如模型文件等，则可以通过Python Archives的方式来访问。

6.2.26.4. 作业迁移

6.2.26.4.1. 迁移方案

本文为您介绍从原有集群迁移Flink作业到DataFlow集群的整体流程，包括准备工作，如何迁移作业，以及常见问题等。

背景信息

EMR集群的基础信息，请参见集群规划。

准备工作

创建DataFlow集群时，集群Core实例的CPU和内存总量可以参考原有集群的规模，再根据作业在DataFlow集群的实际运行情况进行微调。在选择具体机型时，您可以根据是否有本地盘需求、是否希望使用规模较大的物理机等条件进行选择。

Master实例规格通常与整体集群规模大小有关，对应集群最大CU规格经验值如下表所示。

Master型号	集群最大CU规格
4核16 GB	80 CU
8核32 GB	160 CU
16核64 GB	800 CU
24核96 GB	800 CU以上

在EMR控制台创建完DataFlow集群之后，在迁移作业之前，您还需要确保提交Flink作业的客户端所在的机器与DataFlow集群之间的网络互通。

针对原集群的不同情况，您可以选择不同的解决方案：

- 线下IDC自建集群：可以通过阿里云高速通道建立线下IDC和线上E-MapReduce所在VPC网络的连通。
- ECS自建：由于VPC实现用户专有网络之间的逻辑隔离，所以建议使用VPC网络。
  - 经典网络与VPC网络互通：目前阿里云存在经典网络和VPC两种网络类型。由于E-MapReduce集群是在VPC网络中，而很多用户的业务系统还存在于经典网络中，为了解决此问题，阿里云推出了ClassicLink方案，您可以参见此方案进行网络互访，详情请参见建立ClassicLink连接。
  - VPC网络之间连通：选择新旧集群处在同一个区域的同一个可用区内。

由于DataFlow集群采用YARN部署模式，如果提交Flink作业的客户端不位于DataFlow集群内，您还需要在提交Flink作业的客户端的机器上配置好Hadoop相关的配置项，并设置好相关的环境变量。

您自建集群的环境可能与DataFlow集群的环境不完全一致，例如JDK版本不同等。为了保证作业的正常迁移，强烈建议您在迁移生产作业之前，先通过多种方式验证迁移方案是否有效，例如：

- 在迁移生产作业之前，先迁移若干测试作业，观察整个迁移流程是否完备。
- 分批迁移，在迁移重要作业之前，先迁移优先级比较低的作业。甚至可以在条件允许的情况下，您可以继续运行原来的作业，将迁移过来的作业试运行，如果没有问题，再下掉原来的作业。

集群规划

网络互通

Hadoop 环境配置

验证迁移环境

迁移和运行作业

类型	描述
迁移Checkpoint文件	将Checkpoint文件拷贝到DataFlow集群的HDFS上或者上传到OSS中，在提交Flink作业的时候，可以通过-s参数指定Checkpoint文件，即可在DataFlow集群中使用该Checkpoint文件恢复作业。 **注意** 对于DataStream作业来说，开源Flink和VVR的state是完全兼容的，但是对于SQL作业来说，VVR相比社区Flink，做了大量的优化工作，不能保证state完全兼容。对于state不能兼容的作业，无法从开源Flink生成的Checkpoint中恢复。对于这部分作业，客户可以选择使用VVR从零开始重跑，也可以选择使用社区Flink执行。
迁移Flink作业	与原集群中的提交方式类似，您只需要将Flink作业提交到DataFlow集群即可。
运行不同版本的Flink作业	您原集群中的作业使用的Flink版本可能不同，例如，既有基于Flink 1.9版本的，也有基于Flink 1.12版本的。如果您希望不同版本的Flink作业都可以在DataFlow集群中运行，详细方法如下： • 对于特定的Flink版本的作业，使用YARN per-job的方式提交。 在该方式下，作业执行过程中所使用的Flink版本为提交作业的客户端所使用的Flink版本，因此只需要在提交作业的客户端使用指定版本的Flink提交作业即可。 **说明** 如果希望使用本地Flink runtime，则无需指定yarn.provided.lib.dirs参数。 • 对于希望使用DataFlow集群自带的Flink版本（VVR）的作业，则需要通过yarn.provided.lib.dirs参数指定使用集群HDFS中的VVR Runtime（例如，`-D yarn.provided.lib.dirs=hdfs:///flink-current/`），并推荐使用YARN Application模式提交，充分利用集群资源。

对接自建平台

如果您自建了一套大数据平台，则DataFlow集群也可以轻松集成进您现有的平台中：

- 资源管理与运维

  DataFlow集群基于YARN进行资源调度与管理，因此只需要按照集成YARN集群到已有平台的通常操作进行即可。您可以根据需要配置YARN队列的资源等，之后可通过YARN的REST API来访问YARN作业状态，并进行运维。

- 日志查看
  - 对于运行中的作业，可以通过Flink Web UI进行查看，详细信息请参见基础使用。
  - 对于已经运行结束的作业，可以通过Flink History Server查看状态或者通过YARN提供的命令，例如`yarn logs -applicationId application_xxxx_yyyy`来访问作业的日志。

    说明 其中Flink History Server的日志默认存储在HDFS 群的*hdfs:///flink/flink-jobs/*目录下，YARN日志默认存储在HDFS集群的*hdfs:///tmp/logs/$USERNAME/logs/*目录下。

- 指标监控

  如果您的自建平台已有监控报警系统，则可以在Flink的作业配置中，配置相关的Metric Reporter即可。另外，DataFlow集群的Flink作业的指标也对接了EMR的监控报警。

- 报警

  除了可以使用您的报警体系之外，也可以使用阿里云提供的云监控（CloudMonitor）来配置报警规则，并对接邮箱、钉钉群等提醒机制。使用云监控配置详情，请参见创建阈值报警规则。

常见问题

- Q：JDK版本不一致，如何处理？

A：DataFlow集群中使用的是OpenJDK，如果您在原来集群的作业使用的是Oracle JDK，并且您的作业中使用了Oracle JDK中特有的功能时，JDK版本的不一致可能导致运行作业时部分类找不到（例如，javafx.util.Pair等）。因此，您的作业中需要避免显式使用Oracle JDK特有的依赖。

- Q：是否可以支持多个Flink版本？

  A：可以。DataFlow集群支持多种作业提交方式，例如YARN per-job方式、YARN Application方式等。基于Flink on YARN的部署模式，在未设置yarn.provided.lib.dirs参数的情况下，Flink作业在YARN集群中运行时所使用的Flink Runtime为提交作业的客户端所使用的Flink（例如，开源Flink 1.13等）。因此如果您想使用特定的Flink版本运行作业，有两种方式：

  - 直接使用Flink的特定版本进行提交，并且不设置yarn.provided.lib.dirs参数。
  - 通过指定yarn.provided.lib.dirs参数，来使用特定的Flink Runtime。另外，在该场景下，考虑到兼容性，推荐您使用YARN per-job模式进行提交。

6.2.26.5. 最佳实践

6.2.26.5.1. Dataflow集群连接数据湖元数据DLF

EMR-3.38.3后续版本的Dataflow集群，可以通过数据湖元数据DLF（Data Lake Formation）作为元数据读取Hadoop集群中的数据。 本文为您介绍Dataflow集群如何连接DLF，并读取Hive全量数据。

前提条件

- 已在E-MapReduce控制台上创建DataFlow集群和Hadoop集群，详情请参见创建集群。

  > **注意** 创建Hadoop集群时，元数据选择为**DLF统一元数据**。

- 已开通数据湖构建DLF，详情请参见快速入门。

使用限制

- DataFlow集群和Hadoop集群需要在同一VPC下。
- 创建的Dataflow集群需要为EMR-3.38.3后续版本。

操作流程

1. 步骤一：数据准备
2. 步骤二：Dataflow集群连接DLF
3. 步骤三：验证读取Hive全量数据

步骤一：数据准备

1. 下载Hive作业需要的测试数据至OSS对应的目录。

   例如，上传目录为*oss://<yourBucketName>/hive/userdata/*，其中<yourBucketName>为您在OSS控制台上创建的Bucket名称。上传文件详细信息，请参见上传文件。

2. 在DLF控制台创建元数据库，详情请参见创建元数据库。

   例如，创建的元数据库名称为myhudi，选择路径为oss://<yourBucketName>/hive/db。

3. 在Hadoop集群中，查看已经创建的元数据库。

   i. 通过SSH方式登录Hadoop集群，详情请参见登录集群。

   ii. 执行以下命令，切换为hadoop用户。

   ```
   su hadoop
   ```

   iii. 执行以下命令，进入Hive命令行。

   ```
   hive
   ```

   iv. 执行以下命令，查看数据库信息。

   ```
   desc database myhudi;
   ```

   > **说明** 命令中的myhudi为上一步骤中创建的数据库的名称。

   返回信息如下。

   ```
   OK
   myhudi          oss://aliyu****/hive/db    acs:ram::125046002175****:user/299153685100 86****       USER
   Time taken: 0.069 seconds, Fetched: 1 row(s)
   ```

4. 执行以下命令，创建Hive的外表。

```
USE myhudi;
set hive.input.format=org.apache.hadoop.hive.ql.io.HiveInputFormat;
set hive.stats.autogather=false;
DROP TABLE IF EXISTS emrusers;
CREATE EXTERNAL TABLE emrusers (
  userid INT,
  movieid INT,
  rating INT,
  unixtime STRING )
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY '\t'
 STORED AS TEXTFILE
 LOCATION 'oss://<yourBucketName>/hive/userdata/';
```

> **说明** 请替换命令中的<yourBucketName>为您实际在OSS控制台上创建的Bucket名称。

5. 在DLF控制台上，验证表信息。
   i. 登录数据湖构建控制台。
   ii. 在左侧导航栏，选择**元数据管理 > 元数据表**。
   iii. 在**元数据表**页面，通过库名过滤，可以查看已创建的表信息。

6. （可选）在Hive命令行中，查询数据。
   - 示例1

     ```
     SELECT userid,movieid,rating,unix_timestamp() from emrusers limit 10;
     ```

   - 示例2

     ```
     SELECT movieid,count(userid) as usercount from emrusers group by movieid order by usercount desc limit 50;
     ```

步骤二：Dataflow集群连接DLF

1. 通过SSH方式登录DataFlow集群，详情请参见登录集群。
2. 上传Hive配置文件到DataFlow集群的新建路径下。

   您可以执行以下命令，复制Hadoop集群中的*hive-site.xml*文件到DataFlow集群。

   ```
   scp root@<emr-header-1节点内网的IP地址>:/etc/ecm/hive-conf/hive-site.xml /root/test/
   ```

   > **说明** 命令中的<emr-header-1节点内网的IP地址>，您可以在EMR控制台的集群管理的**集群基础信息**页面或**主机列表**页面查看，*/root/test/*为DataFlow集群的路径，您可以根据实际情况修改。

3. 加载集群中内置的Hive Connector，启动SQL客户端。
   i. 在Maven仓库官网中下载依赖的JAR包，并上传至DataFlow集群中。

   本示例是上传到/root目录，下载的JAR包为Jackson Core、Jackson Databind和Jackson Annotations。

   具体版本请根据您实际情况下载，本示例下载的JAR包为*jackson-core-2.12.1.jar*、*jackson-databind-2.12.1.jar*和*jackson-annotations-2.12.1.jar*。

   ii. 执行以下命令，启动SQL客户端。

   ```
   sql-client.sh -j /opt/apps/ecm/service/flink/1.13-vvr-4.0.11-2/package/flink-1.13-vvr-4.0.11-2/opt/catalogs/hive-2.3.6/ververica-connector-hive-2.3.6-1.13-vvr-4.0.11-2-SNAPSHOT-jar-with-dependencies.jar -j /root/jackson-core-2.12.1.jar -j /root/jackson-databind-2.12.1.jar -j /root/jackson-annotations-2.12.1.jar
   ```

   > **说明** 本示例中vvr版本是4.0.11-2，您可以根据实际集群版本选择对应的Connector版本。

4. 在Flink SQL命令行中，创建Catalog。

```
CREATE CATALOG dlfcatalog WITH (
     'type' = 'hive',
     'default-database' = 'myhudi',
     'hive-version' = '2.3.6',
     'hive-conf-dir' = '/root/test',
     'hadoop-conf-dir' = '/etc/ecm/hadoop-conf'
 );
```

涉及参数如下表。

参数	描述
type	固定值为hive。
default-database	步骤一：数据准备中创建的数据库的名称。 本示例为myhudi。
hive-version	固定值为2.3.6。
hive-conf-dir	前一步骤中复制的*hive-site.xml*所在的目录。 本示例为/root/test。
hadoop-conf-dir	固定值为/etc/ecm/hadoop-conf。

返回信息如下，表示创建成功。

```
[INFO] Execute statement succeed.
```

5. 在Flink SQL命令行中，查看DLF的数据库。
   i. 执行以下命令，设置当前的Catalog。

   ```
   USE CATALOG dlfcatalog;
   ```

   ii. 执行以下命令，使用创建的Catalog。

   ```
   SHOW DATABASES;
   ```

   iii. 执行以下命令，设置当前的数据库。

   ```
   USE myhudi;
   ```

   iv. 执行以下命令，查看当前数据库中的表。

   ```
   SHOW TABLES;
   ```

   返回信息如下。

   ```
   +------------+
   | table name |
   +------------+
   |   emrusers |
   +------------+
   1 row in set
   ```

   v. 执行以下命令，查看表信息。

   ```
   desc emrusers;
   ```

   返回信息如下。

   ```
   +----------+--------+------+-----+--------+-----------+
   |     name |   type | null | key | extras | watermark |
   +----------+--------+------+-----+--------+-----------+
   |   userid |    INT | true |     |        |           |
   |  movieid |    INT | true |     |        |           |
   |   rating |    INT | true |     |        |           |
   | unixtime | STRING | true |     |        |           |
   +----------+--------+------+-----+--------+-----------+
   4 rows in set
   ```

步骤三：验证读取Hive全量数据

1. 通过SSH方式登录DataFlow集群，详情请参见登录集群。
2. 执行以下命令，设置为Per-Job Cluster模式。

```
echo "execution.target: yarn-per-job" >> /etc/ecm/flink-conf/flink-conf.yaml
```

3. 执行以下命令，启动SQL客户端。

```
sql-client.sh -j /opt/apps/ecm/service/flink/1.13-vvr-4.0.11-2/package/flink-1.13-vvr-4.0.11-2/opt/catalogs/hive-2.3.6/ververica-connector-hive-2.3.6-1.13-vvr-4.0.11-2-SNAPSHOT-jar-with-dependencies.jar -j /root/jackson-core-2.12.1.jar -j /root/jackson-databind-2.12.1.jar -j /root/jackson-annotations-2.12.1.jar
```

4. 在Flink SQL命令行中进行如下操作。
   i. 执行以下命令，创建Catalog。

```
CREATE CATALOG dlfcatalog WITH (
     'type' = 'hive',
     'default-database' = 'myhudi',
     'hive-version' = '2.3.6',
     'hive-conf-dir' = '/root/test',
     'hadoop-conf-dir' = '/etc/ecm/hadoop-conf'
 );
```

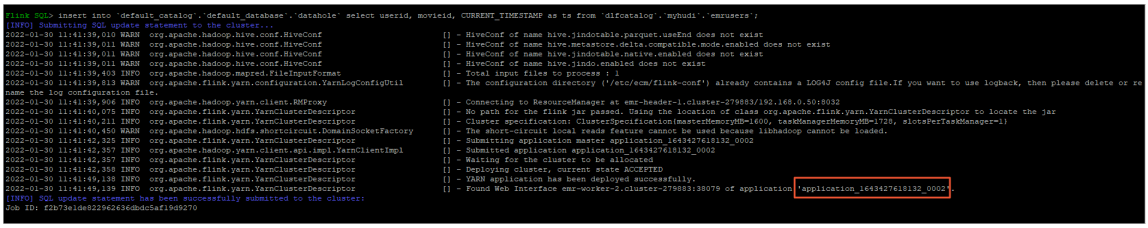
   ii. 执行以下命令，创建表。

```
create table default_catalog.default_database.datahole(userid int, movieid int, ts timestamp) with ('connector' = 'blackhole');
```

   iii. 执行以下命令，读取Hive全量数据到创建的表。

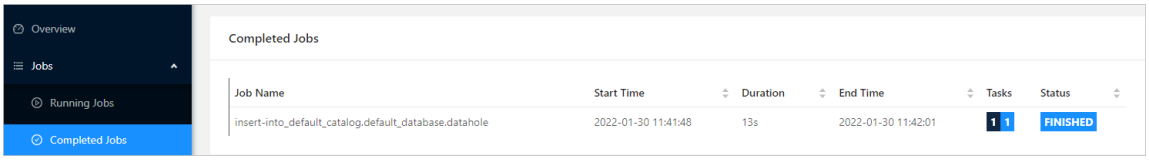
```
insert into `default_catalog`.`default_database`.`datahole` select userid, movieid, CURRENT_TIMESTAMP as ts from `dlfcatalog`.`myhudi`.`emrusers`;
```

   执行成功后，会返回已提交的Flink作业的Application ID。返回如下类似信息。



> **说明** 本示例返回的Application ID为application_1643427618132_0002。

5. 查看作业。
   i. 通过Web UI查看作业状态，详情请参见通过Web UI查看作业状态。
   ii. 在Hadoop控制台，单击目标作业的Application ID，可以查看作业运行的详情。
   iii. 单击**Tracking URL**所在行的链接。
   iv. 在左侧导航栏中，选择**Jobs > Completed Jobs**。
   可以查看已完成的作业。



6.2.26.5.2. 通过DataFlow集群将数据流式写入OSS

在EMR 3.37.1及之后的版本中，DataFlow集群内置了JindoFS相关的依赖。您可以在DataFlow集群中运行Flink作业，将数据流式写入阿里云OSS。本文通过示例为您介绍如何在Dataflow集群中运行Flink作业写入同账号下的阿里云OSS。

背景信息

关于JindoFS的部分高级配置（例如，熵注入），请参见支持Flink可恢复性写入JindoFS或OSS。

前提条件

- 已开通E-MapReduce服务和OSS服务。

- 已完成云账号的授权，详情请参见角色授权。

操作流程

1. 步骤一：准备环境
2. 步骤二：准备JAR包
3. 步骤三：运行Flink作业
4. 步骤四：在OSS上查看输出的结果

步骤一：准备环境

1. 创建Dataflow集群，详情请参见创建集群。

   > **说明** 本文以EMR-3.39.1版本为例。

2. 在OSS上创建与Dataflow集群相同区域的Bucket，详情请参见创建存储空间。

步骤二：准备JAR包

1. 下载DataFlowOSSDemo.tar.gz。

   基于JindoFS，您可以在Flink作业中，像写HDFS一样将数据以流式的方式写入OSS中（路径需要以oss://为前缀）。本示例中使用了Flink的StreamingFileSink方法来演示开启了检查点（Checkpoint）之后，Flink如何以Exactly-Once语义写入OSS。

```
import org.apache.flink.api.common.serialization.SimpleStringEncoder;
import org.apache.flink.api.common.typeinfo.TypeInformation;
import org.apache.flink.configuration.Configuration;
import org.apache.flink.core.fs.Path;
import org.apache.flink.streaming.api.CheckpointingMode;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.api.functions.sink.filesystem.StreamingFileSink;
import org.apache.flink.streaming.api.functions.sink.filesystem.rollingpolicies.OnCheckpointRollingPolicy;
import org.apache.flink.streaming.api.functions.source.RichParallelSourceFunction;
import java.util.Random;
public class OssDemoJob {
    public static void main(String[] args) throws Exception {
        // Set up the streaming execution environment
        final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        // Checkpoint is required
        env.enableCheckpointing(30000, CheckpointingMode.EXACTLY_ONCE);
        // Change to your oss path
        String outputPath = "oss://targetBucket/targetDir";
        DataStream<String> StringStream =
            env.addSource(new RandomStringSource()).returns(TypeInformation.of(String.class));
        StreamingFileSink<String> sink =
            StreamingFileSink.forRowFormat(
            new Path(outputPath), new SimpleStringEncoder<String>("UTF-8"))
            .withRollingPolicy(OnCheckpointRollingPolicy.build())
            .build();
        StringStream.addSink(sink);
        // Compile and submit the job
        env.execute();
    }
    public static class RandomStringSource extends RichParallelSourceFunction<String> {
        private volatile boolean cancelled = false;
        private Random random;
        @Override
        public void open(Configuration parameters) throws Exception {
            super.open(parameters);
            random = new Random();
        }
        @Override
        public void run(SourceContext<String> ctx) throws Exception {
            while (!cancelled) {
                long nextLong = random.nextLong();
                synchronized (ctx.getCheckpointLock()) {
                    ctx.collect("s" + nextLong);
                }
            }
        }
        @Override
        public void cancel() {
            cancelled = true;
        }
    }
}
```

> **说明** 本示例代码片段给出了主要的示例程序，您可以根据自身环境进行修改（例如，添加包名，修改代码中的OSS路径）后，进行编译。关于如何构建Flink作业的JAR包，可以参见Flink官方文档。

2. 在CMD命令行中，进入到下载文件中*pom.xml*所在的目录下，执行以下命令打包文件。

```
mvn clean package
```

根据您*pom.xml*文件中artifactId的信息，下载文件中的target目录下会出现*OssDemoJob.jar*的JAR包。

步骤三：运行Flink作业

1. 通过SSH方式连接Dataflow集群，详情请参见登录集群。
2. 上传打包好的*OssDemoJob.jar*至Dataflow集群的根目录下。

> **说明** 本文示例中*OssDemoJob.jar*是上传至*root*根目录下，您也可以自定义上传路径。

3. 执行以下命令，提交作业。

   本示例通过Per-Job Cluster模式提交作业，其他方式请参见基础使用。

```
flink run -t yarn-per-job -d OssDemoJob.jar
```

```
[root@emr-header-1 ~]# flink run -t yarn-per-job -d OssDemoJob.jar
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/apps/ecm/service/flink/1.13-vvr-4.0.11-2/package/flink-1.13-vvr-4.0.11-2/lib/log4j-slf4j-impl-2.17.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/apps/ecm/service/hadoop/2.8.5-1.6.5/package/hadoop-2.8.5-1.6.5/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
2022-03-23 14:13:20,075 WARN  org.apache.flink.yarn.configuration.YarnLogConfigUtil        [] - The configuration directory ('/etc/ecm/flink-conf') already contains a LOG4J config file.If you want
name the log configuration file.
2022-03-23 14:13:20,303 INFO  org.apache.hadoop.yarn.client.RMProxy                        [] - Connecting to ResourceManager at emr-header-1.cluster-290222/192.168.[redacted]8032
2022-03-23 14:13:20,502 INFO  org.apache.flink.yarn.YarnClusterDescriptor                  [] - No path for the flink jar passed. Using the location of class org.apache.flink.yarn.YarnClusterDescri
2022-03-23 14:13:20,807 INFO  org.apache.flink.yarn.YarnClusterDescriptor                  [] - Cluster specification: ClusterSpecification{masterMemoryMB=1600, taskManagerMemoryMB=1728, slotsPerTa
2022-03-23 14:13:21,343 WARN  org.apache.hadoop.hdfs.shortcircuit.DomainSocketFactory      [] - The short-circuit local reads feature cannot be used because libhadoop cannot be loaded.
2022-03-23 14:13:23,349 INFO  org.apache.flink.yarn.YarnClusterDescriptor                  [] - Submitting application master application_1647310741576_0001
2022-03-23 14:13:23,794 INFO  org.apache.hadoop.yarn.client.api.impl.YarnClientImpl        [] - Submitted application application_1647310741576_0001
2022-03-23 14:13:23,794 INFO  org.apache.flink.yarn.YarnClusterDescriptor                  [] - Waiting for the cluster to be allocated
2022-03-23 14:13:23,797 INFO  org.apache.flink.yarn.YarnClusterDescriptor                  [] - Deploying cluster, current state ACCEPTED
2022-03-23 14:13:33,884 INFO  org.apache.flink.yarn.YarnClusterDescriptor                  [] - YARN application has been deployed successfully.
2022-03-23 14:13:33,886 INFO  org.apache.flink.yarn.YarnClusterDescriptor                  [] - The Flink YARN session cluster has been started in detached mode. In order to stop Flink gracefully,
$ echo "stop" | ./bin/yarn-session.sh -id application_1647310741576_0001
If this should not be possible, then you can also kill Flink via YARN's web interface or via:
$ yarn application -kill application_1647310741576_0001
Note that killing Flink might not clean up all job artifacts and temporary files.
2022-03-23 14:13:33,891 INFO  org.apache.flink.yarn.YarnClusterDescriptor                  [] - Found Web Interface emr-worker-1.cluster-290222:34475 of application 'application_1647310741576_0001'
Job has been submitted with JobID 17a841b5b1aa6c3ec5f45df3ec368b7e
```

   您可以执行以下命令，查看作业状态。

```
flink list -t yarn-per-job -Dyarn.application.id=<appId>
```

> **说明** `<appId>` 为作业运行后返回的Application ID。例如，本示例截图中的application_1647310741576_0001。

步骤四：在OSS上查看输出的结果

作业正常运行后，您可以在OSS控制台查看输出结果。

1. 登录 OSS管理控制台。
2. 单击创建的存储空间。
3. 在文件管理页面指定的输出目录下查看输出结果。

   输出结果类似下图所示。

> **说明** 该作业为流式作业会持续运行，产生较多输出文件。在完成验证后，应及时在命令行中通过 `yarn application -kill <appId>` 命令终止该作业。

6.2.26.6. 常见问题

本文汇总了DataFlow集群使用时的常见问题。

- DataFlow集群外的机器，如何提交作业到DataFlow集群？
- 在DataFlow集群外机器上，如何解析DataFlow集群中的hostname？
- 如何查看Flink作业的运行状态？
- 如何访问Flink作业的日志？
- 如何访问DataFlow集群中的Flink HistoryServer？
- 如何使用DataFlow集群中所支持的商业化Connector？
- 如何使用GeminiStateBackend？
- 如何开启Flink作业JobManager的HA？
- 如何处理上下游存储（Connector）问题？

DataFlow集群外的机器，如何提交作业到DataFlow集群？

您可以根据以下步骤，通过DataFlow集群外的机器，提交作业到DataFlow集群：

1. 确保DataFlow集群和DataFlow集群外的机器网络互通。
2. 配置提交Flink作业的客户端的Hadoop环境。

   DataFlow集群中的Hadoop的软件安装目录是*/usr/lib/hadoop-current*，配置文件的目录是*/etc/ecm/hadoop-conf*，您需要将*hadoop-current*目录及*hadoop-conf*目录下载到提交Flink作业的客户端上。

   然后，在提交Flink作业的客户端上，配置如下环境变量。

   ```
   export HADOOP_HOME=/path/to/hadoop-current && \
   export PATH=${HADOOP_HOME}/bin/:$PATH && \
   export HADOOP_CLASSPATH=$(hadoop classpath) && \
   export HADOOP_CONF_DIR=/path/to/hadoop-conf
   ```

   > **注意** Hadoop的配置文件中（例如yarn-site.xml等）配置的服务地址（例如ResourceManager等），使用的是hostname，例如emr-header-1.cluster*****等。因此，如果您通过集群外的机器提交作业，需要能够解析这些hostname或者将配置文件中的hostname修改成对应的IP地址。

在DataFlow集群外机器上，如何解析DataFlow集群中的hostname？

您可以通过以下方式，在DataFlow集群外的机器上，解析DataFlow集群中的hostname：

- 修改提交Flink作业的客户端上的*/etc/hosts*文件，添加相应的hostname到IP的映射。
- 通过什么是PrivateZone提供的DNS解析服务。

  如果您有自己的域名解析服务，也可以通过如下方式，配置JVM的运行参数，使用自己的域名解析服务。

  ```
  env.java.opts.client: "-Dsun.net.spi.nameservice.nameservers=xxx -Dsun.net.spi.nameservice.provider.1=dns,sun -Dsun.net.spi.nameservice.domain=yyy"
  ```

如何查看Flink作业的运行状态？

- 通过EMR控制台查看。

  EMR支持Knox，可以通过公网方式访问YARN、Flink等的Web UI界面，Flink的Web UI可以通过YARN进行查看，详细信息请参见通过Web UI查看作业状态。
- 通过SSH隧道的方式查看，详情信息请参见通过SSH隧道方式访问开源组件Web UI。
- 直接访问YARN REST接口。

  ```
  curl --compressed -v  -H "Accept: application/json" -X GET "http://emr-header-1:8088/ws/v1/cluster/apps?states=RUNNING&queue=default&user.name=***"
  ```

  > **说明** 需确保安全组开放了8443和8088端口，可以访问到YARN的REST接口或者DataFlow集群和访问的节点处于同一内网中。

如何访问Flink作业的日志？

- 对于运行中的作业，可以通过Flink Web UI，访问Flink作业的日志。
- 对于已经运行结束的作业，可以通过Flink History Server查看作业的统计信息或者通过命令 `yarn logs -applicationId application_xxxx_yyyy` 访问作业的日志，已经运行结束的作业的日志默认保存在HDFS集群的*hdfs:///tmp/logs/$USERNAME/logs/*目录下。

如何访问DataFlow集群中的Flink HistoryServer？

DataFlow集群会默认在emr-header-1节点的8082端口启动Flink HistoryServer，用于收集已运行结束的作业的统计信息，具体访问方式如下：

1. 配置安全组规则，开放emr-header-1节点的8082端口的访问权限。
2. 直接访问*http://$emr-header-1-ip:8082*。

如何使用DataFlow集群中所支持的商业化Connector？

DataFlow集群提供了很多商业化Connector，例如Hologres、SLS、MaxCompute、DataHub、Elasticsearch、ClickHouse、Hudi和Hive等，您在Flink作业中除了可以使用开源的Connector之外，还可以使用这些商业化Connector。下面以Hologres Connector为例，介绍如何在Flink作业中使用DataFlow集群所携带的商业化Connector。

- 作业开发
  i. 下载DataFlow集群所携带的商业化Connector的JAR包（位于DataFlow集群的*/usr/lib/flink-current/opt/connectors*目录下），并通过如下方式将商业化Connector安装在本地Maven环境中。

  ```
  mvn install:install-file -Dfile=/path/to/ververica-connector-hologres-1.13-vvr-4.0.7.jar -DgroupId=com.alibaba.ververica -DartifactId=ververica-connector-hologres -Dversion=1.13-vvr-4.0.7 -Dpackaging=jar
  ```

  ii. 在项目的*pom.xml*文件中添加以下依赖。

```
<dependency>
    <groupId>com.alibaba.ververica</groupId>
    <artifactId>ververica-connector-hologres</artifactId>
    <version>1.13-vvr-4.0.7</version>
    <scope>provided</scope>
</dependency>
```

- 运行作业
  - 方式一：
    a. 拷贝Hologres Connector到一个独立的目录。

    ```
    hdfs mkdir hdfs:///flink-current/opt/connectors/hologres/
    hdfs cp hdfs:///flink-current/opt/connectors/ververica-connector-hologres-1.13-vvr-4.0.7.jar  hdfs:///flink-current
    /opt/connectors/hologres/ververica-connector-hologres-1.13-vvr-4.0.7.jar
    ```

    b. 提交作业时，命令中添加以下参数。

    ```
    -D yarn.provided.lib.dirs=hdfs:///flink-current/opt/connectors/hologres/
    ```

  - 方式二：
    a. 拷贝Hologres Connector到提交Flink作业的客户端的*/usr/lib/flink-current/opt/connectors/ververica-connector-hologres-1.13-vvr-4.0.7.jar*目录下，与DataFlow集群中的目录结构保持一致。
    b. 提交作业时，命令中添加以下参数。

    ```
    -C file:///usr/lib/flink-current/opt/connectors/ververica-connector-hologres-1.13-vvr-4.0.7.jar
    ```

  - 方式三：将Hologres Connector打包到作业的JAR包中。

如何使用GeminiStateBackend?

DataFlow集群提供了企业版StateBackend（即GeminiStateBackend），性能是开源版本的3～5倍。关于如何使用GeminiStateBackend以及更详细的信息，请参见Flink（VVR）作业配置。

如何开启Flink作业JobManager的HA?

DataFlow集群基于YARN模式部署并运行Flink作业，您可以按照社区中的Configuration开启JobManager的HA，从而使Flink作业可以更稳定的运行。配置示例如下所示。

```
high-availability: zookeeper
high-availability.zookeeper.quorum: 192.168.**.**:2181,192.168.**.**:2181,192.168.**.**:2181
high-availability.zookeeper.path.root: /flink
high-availability.storageDir: hdfs:///flink/recovery
```

> **注意** 开启HA后，默认情况下，JobManager在失败后最多重启一次。如果您想让JobManager重启多次，还需要设置YARN的yarn.resourcemanager.am.max-attempts参数和Flink的yarn.application-attempts参数，详情请参见Flink官方文档。除此之外，根据经验，通常还需要调整yarn.application-attempt-failures-validity-interval参数的值，将其从默认的10000毫秒（10秒）调整到一个比较大的值，例如调大为300000毫秒（5分钟），防止JobManager不停的重启。

如何处理上下游存储（Connector）问题?

关于上下游存储方面的常见问题，请参见上下游存储。

6.2.27. Kafka

6.2.27.1. 概述

EMR-3.4.0及后续版本支持Kafka服务。

创建Kafka集群

在E-MapReduce控制台，创建Kafka集群，详情请参见创建集群。

本地盘Kafka集群

本地盘详情信息请参见本地盘机型概述。

当在本地盘上部署Kafka服务时，您需要在E-MapReduce控制台的**配置**页面，配置如下参数。

配置项	描述
default.replication.factor	固定值为3，表示Topic的副本数为3。
min.insync.replicas	固定值为2，表示副本数大于等于2。 当Producer设定request.required.acks为all或-1，且写入副本数大于等于2时，数据写入才能成功。

参数说明

您可以在E-MapReduce控制台的**配置**页面，查看Kafka的服务配置。

配置项	描述
zookeeper.connect	Kafka集群Zookeeper的连接地址。
kafka.heap.opts	Kafka Broker的堆内存大小。
num.io.threads	Kafka Broker的IO线程数，默认为主节点CPU核数的2倍。
num.network.threads	Kafka Broker的网络线程数，默认为主节点的CPU核数。

6.2.27.2. 跨集群访问Kafka

本文介绍当您单独部署Kafka集群时，如何跨集群访问Kafka服务。

背景信息

跨集群访问Kafka场景分为两种：

- 阿里云内网环境访问E-MapReduce Kafka集群。
- 公网环境访问E-MapReduce Kafka集群。

EMR-3.11.x及后续版本

- 阿里云内网访问Kafka

  直接使用Kafka集群节点的内网IP访问即可，内网访问Kafka请使用9092端口。

  访问Kafka前请保证网络互通，VPC访问VPC的配置请参见配置VPC到VPC连接。

- 公网环境访问Kafka

  Kafka集群的Core节点默认无法通过公网访问，所以如果您需要公网环境访问Kafka集群。

  Kafka集群部署在VPC网络环境，有两种方式：

  - 集群Core节点挂载弹性公网IP：
    a. 在E-MapReduce集群管理页面，单击对应集群操作栏中的**详情**。
    b. 在**集群基础信息**页面，单击右上角的**网络管理 > 挂载公网**。
    c. 在**集群基础信息**页面的**网络信息**区域，根据安全组ID去查找并配置安全组规则，具体请参见添加安全组规则。
    d. 在**集群基础信息**页面，单击页面右上角的**实例状态管理 > 同步主机信息**。
    e. 在左侧导航栏，单击**集群服务 > Kafka**。单击**配置**页签，在**服务配置**中，修改**kafka.public-access.enable**为**true**。
    f. 生效配置并重启Kafka服务。
    g. 公网环境使用Kafka集群节点的EIP访问9093端口。
  - 部署高速通道打通内网和公网网络，详情请参见高速通道。

EMR-3.11.x之前版本

- 阿里云内网中访问Kafka

  您需要在Master主机上配置Kafka集群节点的Host信息。示例如下。

```
/etc/hosts
# kafka cluster
10.0.1.23 emr-header-1.cluster-48742
10.0.1.24 emr-worker-1.cluster-48742
10.0.1.25 emr-worker-2.cluster-48742
10.0.1.26 emr-worker-3.cluster-48742
```

> 注意 请在Client端的主机上配置Kafka集群节点的**长域名**，否则访问不到Kafka服务。

- 公网环境访问Kafka

  因为Kafka集群的Core节点默认无法通过公网访问，所以当您需要在公网环境访问Kafka集群时，需要执行以下操作：

  i. Kafka集群和公网主机网络互通。

  Kafka集群部署在VPC网络环境，有两种方式：

  - 集群Core节点挂载弹性公网IP。
  - 部署高速通道打通内网和公网网络，详情请参见高速通道。

  ii. 在VPC控制台VPC 控制台申请EIP，根据您Kafka集群Core节点个数购买相应的EIP。

  iii. 根据安全组ID查找并配置安全组规则，详情请参见添加安全组规则。

  iv. 修改Kafka集群配置项的listeners.address.principal为HOST，并重启Kafka集群。

  v. 配置本地客户端主机的*hosts*文件。

6.2.27.3. 使用Kafka Ranger

本文介绍如何将Kafka集成到Ranger，以及如何配置权限。

前提条件

Kafka集成Ranger

1. Ranger启用Kafka。

   i.

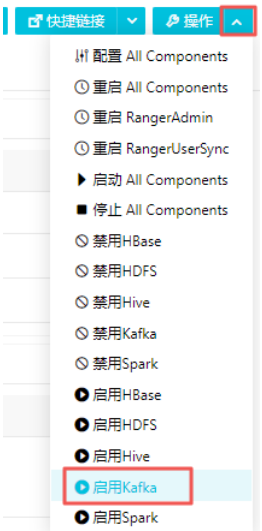
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。

   iii. 单击上方的**集群管理**页签。

   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。

   v. 在左侧导航栏单击**集群服务 > RANGER**。

   vi. 单击右侧的**操作**，选择**启用Kafka**。

   vii. 执行集群操作。

   a. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。

   b. 在**确认**对话框，单击**确定**。

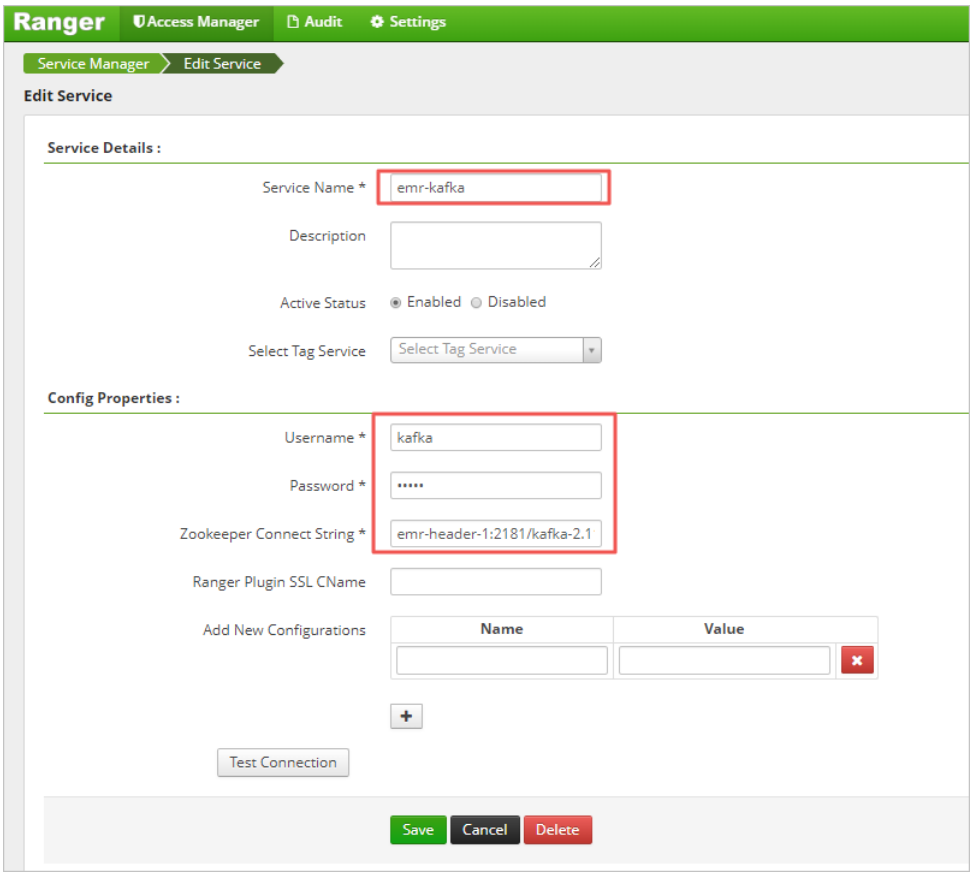
   c. 单击右上角**查看操作历史**查看任务进度，等待任务完成。

2. Ranger UI页面添加Kafka Service。

   i. 进入Ranger UI页面，详情请参见概述。

ii. 在Ranger UI页面，添加Kafka Service。

iii. 配置Kafka Service。



参数	说明
Service Name	固定值emr-kafka。
Username	固定值kafka。
Password	可自定义。
Zookeeper Connect String	填写格式emr-header-1:2181/kafka-x.xx。 说明 其中 kafka-x.xx 根据Kafka实际版本填写。

iv. 单击**Add**。

3. 重启Kafka Broker。
   i. 左侧导航栏单击**集群服务 > Kafka**。
   ii. 单击右上角**操作**下拉菜单，选择 **重启 Kafka Broker**。
   iii. 执行集群操作。
      a. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
      b. 在**确认**对话框，单击**确定**。
      c. 单击右上角**查看操作历史**查看任务进度，等待任务完成。

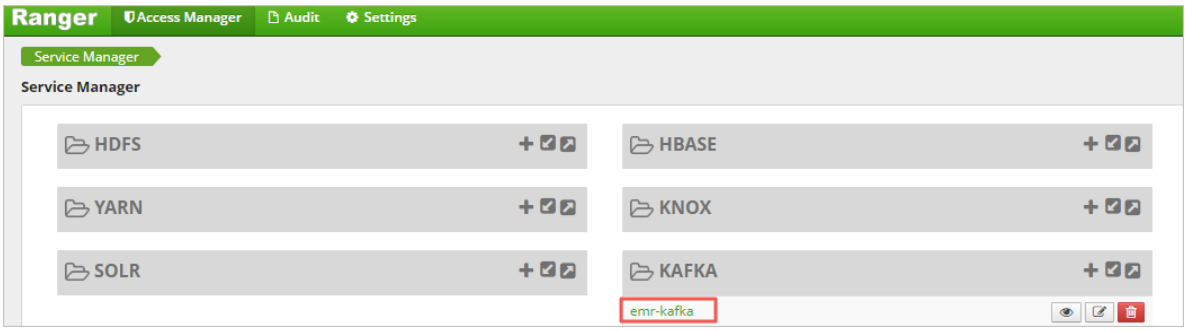
权限配置示例

上面一节中已经将Ranger集成到Kafka，现在可以设置权限。

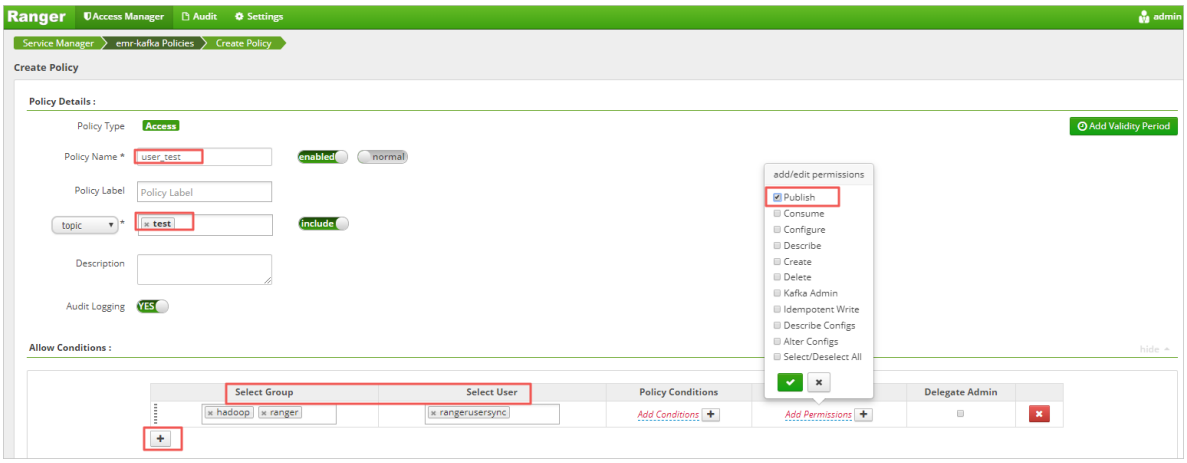
> 注意 标准集群中，在添加了Kafka Service后，Ranger会默认生成规则all - topic，不作任何权限限制（即允许所有用户进行所有操作），此时Ranger无法通过用户进行权限识别。

以test用户为例，添加Publish权限。

1. 进入Ranger UI页面，详情请参见概述。
2. 在Ranger UI页面，单击配置好的**emr-kafka**。



3. 单击右上角的**Add New Policy**。
4. 填写参数。



参数	说明
Policy Name	策略名称，可以自定义。
topic	自定义。可填写多个，填写一个需按一次Enter键。
Select Group	指定添加此策略的用户组。
Select User	指定添加此策略的用户。例如，test。
Permissions	单击+，选择**Publish**。

单击**Select Group**下方的+，可以对多个Group进行授权。

5. 单击**Add**。

添加Policy后，实现对**test**的授权。test用户即可以对名为**test**的topic执行写入操作。

> **说明** 添加、删除或修改Policy后，需要等待约一分钟至授权生效。

6.2.27.4. 使用SSL连接Kafka

本文为您介绍如何开启E-MapReduce Kafka集群的SSL功能，并通过SSL连接Kafka。

前提条件

已创建E-MapReduce的Kafka类型的集群。

创建集群详情，请参见创建集群。

开启SSL服务

Kafka集群的SSL功能默认关闭，您可以执行以下步骤开启SSL功能。

1. 进入详情页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
2. 在左侧导航栏中，选择**集群服务 > Kafka**。
3. 在Kafka服务页面，单击上方的**配置**页签。
4. 修改配置。
   i. 在配置搜索中，输入配置项**kafka.ssl.enable**，单击 图标。
   ii. 设置**kafka.ssl.enable**的值为**true**。
   iii. 单击**保存**。
   iv. 在**确认修改**页面，输入**执行原因**，开启**自动更新配置**，单击**确定**。
5. 重启配置。
   i. 单击右上角的**操作 > 重启All Componens**。
   ii. 在**执行集群操作**页面，输入**执行原因**，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

使用SSL连接Kafka

使用SSL连接Kafka时，需要配置参数security.protocol、ssl.truststore.password和ssl.keystore.password。

例如，在未开启高安全模式的Kafka集群中，使用Kafka自带的Producer和Consumer执行作业，操作步骤如下：

1. 创建配置文件*ssl.properties*，添加配置项。

```
security.protocol=SSL
ssl.truststore.location=/etc/ecm/kafka-conf/truststore
ssl.truststore.password=${password}
ssl.keystore.location=/etc/ecm/kafka-conf/keystore
ssl.keystore.password=${password}
```

> **说明** `${password}` 的值，您可以在Kafka集群的*/etc/ecm/kafka-conf/server.properties*文件中获取。如果是在Kafka集群以外的环境执行作业，您可以将Kafka集群中任意节点*/etc/ecm/kafka-conf/*目录下的*truststore*和*keystore*文件，拷贝至运行环境进行相应配置。

2. 创建Topic。

```
/usr/lib/kafka-current/bin/kafka-topics.sh --partitions 10 --replication-factor 2 --zookeeper emr-header-1:2181 /kafka-1.0.0 --topic test --create
```

3. 使用SSL配置文件产生数据。

```
kafka-producer-perf-test.sh --topic test --num-records 123456 --throughput 10000 --record-size 1024 --producer-props bootstrap.servers=emr-worker-1:9092 --producer.config ssl.properties
```

4. 使用SSL配置文件消费数据。

```
kafka-consumer-perf-test.sh --broker-list emr-worker-1:9092 --messages 100000000 --topic test --consumer.config ssl.properties
```

6.2.27.5. Kafka常见问题

本文介绍使用Kafka时可能遇到的问题及解决方法。

- 报错“ERROR: Wile executing topic command : Replication factor: 1 larger than available brokers: 0.”
- 报错“java.net.BindException: Address already in use (Bind failed)”

报错“ERROR: Wile executing topic command : Replication factor: 1 larger than available brokers: 0.”

问题分析：

- Kafka服务异常，集群Broker退出进程。
- Kafka服务的ZooKeeper地址错误。

解决方法：

- 请结合日志排查问题。
- 请您使用集群配置管理中Kafka组件的ZooKeeper连接地址。

报错“java.net.BindException: Address already in use (Bind failed)”

JMX端口被占用，您可以在命令行前手动指定一个JMX端口即可。示例如下。

```
JMX_PORT=10101 kafka-topics --zookeeper emr-header-1:2181/kafka-1.0.0 --list
```

6.2.28. Kafka Manager

E-MapReduce支持通过Kafka Manager服务对Kafka集群进行管理。

前提条件

已创建Kafka类型的集群，创建详情请参见创建集群。

> **说明** 创建Kafka集群时，默认安装Kafka Manager软件服务，并开启Kafka Manager的认证功能。

操作步骤

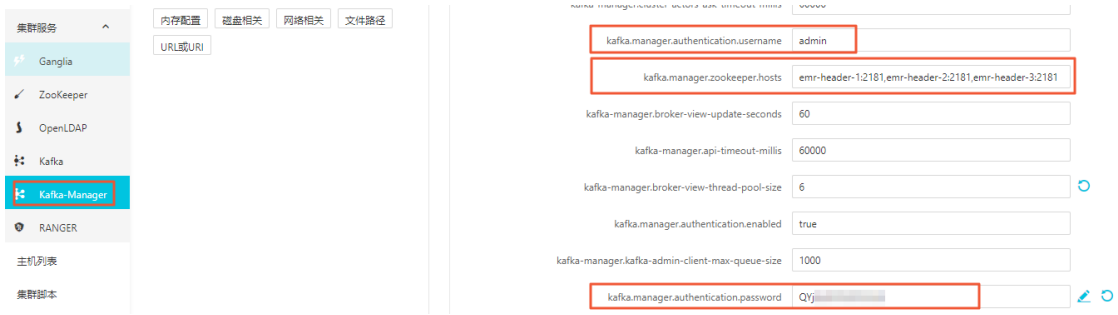
1. 使用SSH隧道方式访问Web页面，详情请参见通过SSH隧道方式访问开源组件Web UI。

   > **说明**
   > - 建议您首次使用Kafka Manager时修改默认密码。
   > - 为了防止8085端口暴露，建议使用SSH隧道方式来访问Web界面。如果使用*http://localhost:8085*方式访问Web界面，请做好IP白名单保护，避免数据泄漏。

2. 在登录页面，输入用户名和密码。

   用户名和密码可以通过以下步骤获取：

   i. 登录阿里云E-MapReduce控制台
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面的集群列表中，单击对应集群后面的**详情**。
   v. 在左侧导航栏，选择**集群服务 > Kafka-Manager**。
   vi. 单击**配置**，在**服务配置**区域，查看以下参数的值：
   - **kafka.manager.authentication.username**：登录Kafka Manager页面的用户名。
   - **kafka.manager.authentication.password**：登录Kafka Manager页面的密码。
   - kafka.manager.zookeeper.hosts：Kafka集群的Zookeeper地址。



3. 添加已创建的Kafka集群。

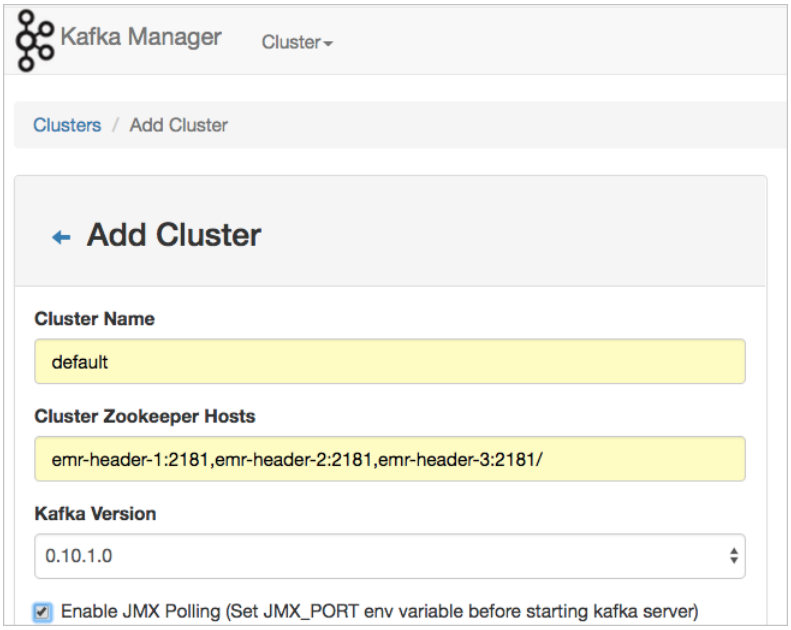
   i. 输入集群名称。

ii. 配置Kafka集群的Zookeeper地址。

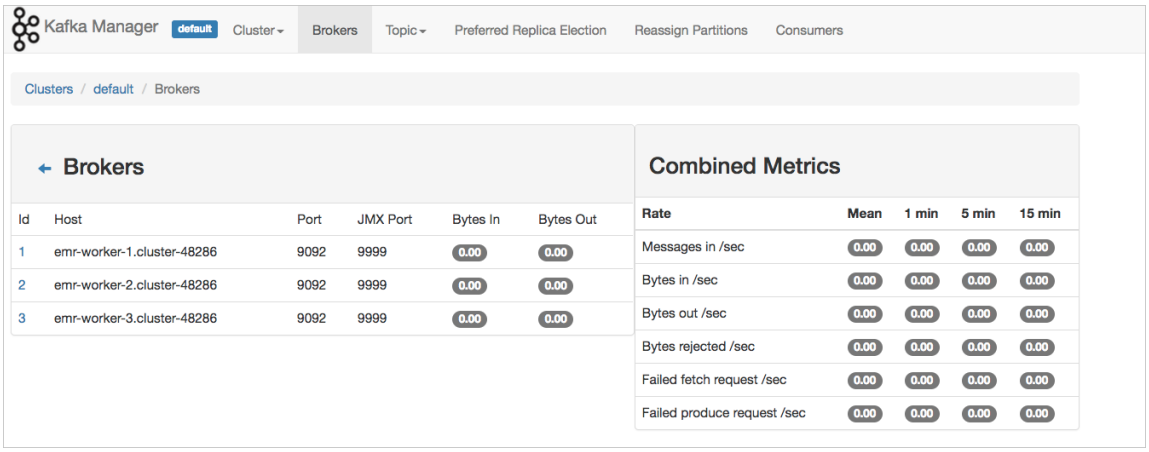
填写在步骤2中获取kafka.manager.zookeeper.hosts的值。

iii. 选择对应的Kafka版本。

iv. （可选）建议打开JMX功能。



创建好之后即可使用常见的Kafka功能。



6.2.29. DeltaLake

6.2.29.1. Delta Lake概述

Delta Lake是DataBricks公司推出的一种数据湖方案。Delta Lake以数据为中心，围绕数据流走向（数据从流入数据湖、数据组织管理和数据查询到流出数据湖）推出了一系列功能特性，协助您搭配第三方上下游工具，搭建快捷、易用和安全的数据湖。

背景信息

通常的数据湖方案是选取大数据存储引擎构建数据湖（例如，阿里云对象存储OSS产品或云下HDFS），然后将产生的各种类型数据存储在该存储引擎中。在使用数据时，通过Spark或Presto对接数据分析引擎并进行数据解析。但该套方案存在如下问题：

- 数据导入可能会失败，失败后清理脏数据和恢复作业困难。
- 方案中没有ETL（Extract Transform Load）过程，缺少必要的数据质量监管。
- 方案中没有事务将读和写隔离，致使流式和批式读写无法相互隔离。

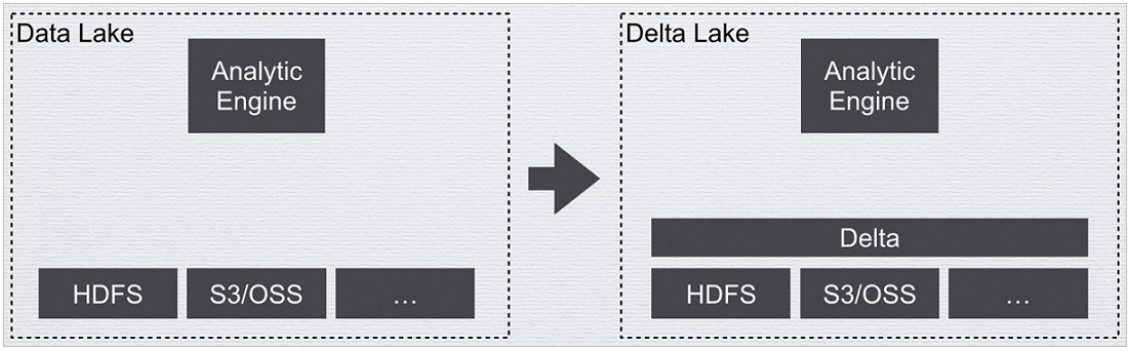
Delta数据湖方案如下：

- 在大数据存储层之上提供了数据管理层，该数据管理层等同于数据库中的元数据管理，其元数据随着数据一起存放并对用户可见（例如数据仓库与数据湖所示）。
- Delta基于元数据管理引入了ACID，解决了因数据导入失败而产生脏数据和数据导入时的读写隔离问题。

- 元数据存储了数据的字段信息，Delta提供了数据导入时数据校验功能，保证数据质量。
- 事务功能使得批式读写和流式读写能够互相隔离。

> **说明** ACID指数据库事务正确执行的四个基本要素的缩写。包含：原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）和持久性（Durability）。

数据仓库与数据湖



Data Warehouse、Data Lake和Delta Lake对比如下所示。

对比项	Data Warehouse	Data Lake	Delta Lake
架构	计算存储一体或分离	计算存储分离	计算存储分离
存储管理	严格、非通用	原生格式	通用格式、轻量级
场景	报表、分析	报表、分析、数据科学	报表、分析、数据科学
灵活性	低	高	较高
数据质量和可靠性	很高	低	较高
事务性	支持	不支持	支持
性能	高	低	较高
扩展性	依赖于具体实现	高	高
面向人员	管理人员	管理人员、数据科学家	管理人员、数据科学家
成本	高	低	低

适用场景

Delta适用于云上数据湖数据管理解决方案。如果您存在以下场景，可以使用Delta：

- 实时查询：数据实时从上游流入Delta，查询侧即可查询该数据，例如，在CDC场景下，Spark Streaming实时消费binlog时，使用Delta merge功能，实时将上游的数据通过merge更新到Delta Lake，然后可以使用Hive、Spark或Presto实时查询。同时，由于支持ACID，保证了数据的流入和查询的隔离性，不会产生脏读数据。
- 删除或更新，GDPR（General Data Protection Regulation）：通常数据湖方案不支持数据的删除或更新。如果需要删除或更新数据，则需要把原始数据清理掉，然后把更新后的数据写入存储。而Delta支持数据的删除或更新。
- 数据实时同步，CDC（Change Data Capture）：使用Delta merge功能，启动流作业，实时将上游的数据通过merge更新到Delta Lake。
- 数据质量控制：借助于Delta Schema校验功能，在数据导入时剔除异常数据，或者对异常数据做进一步处理。
- 数据演化：数据的Schema并非固定不变，Delta支持通过API方式改变数据的Schema。
- 实时机器学习：在机器学习场景中，通常需要花费大量的时间用于处理数据，例如数据清洗、转换、提取特征等等。同时，您还需要对历史和实时数据分别处理。而Delta简化了工作流程，整条数据处理过程是一条完整的、可靠的实时流，其数据的清洗、转换、特征化等操作都是流上的节点动作，无需对历史和实时数据分别处理。

与开源Delta Lake对比

EMR-Delta Lake丰富了开源Delta Lake的特性，例如对SQL和Optimize的支持等。下表列出了Delta Lake的基本特性，并对比EMR-Delta Lake与开源Delta Lake（0.6.1）。

特性	EMR-Delta	开源Delta

特性	EMR-Delta	开源Delta
SQL	• ALTER • CONVERT • CREATE • CTAS • DELETE • DESC HISTORY • INSERT • MERGE • OPTIMIZE • UPDATE • VACUUM • SAVEPOINT • ROLLBACK	• CREATE 说明 建表示例：CREATE TABLE <tbl> USING delta LOCATION <delta_table_path> ◦ 仅支持基于已有的Delta目录建表。 ◦ 建表时请不要指定Schema。 • CONVERT • DESC HISTORY • VACUUM
API	• batch read/write • streaming read/write • optimize • delete • update • merge • convert • history • vacuum • savepoint • rollback	• batch read/write • streaming read/write • delete • update • merge • convert • history • vacuum
Hive connector	支持	支持
Presto connector	支持	支持
Parquet	支持	支持
ORC	不支持	不支持
文本格式	不支持	不支持
Data Skipping	支持	不支持
ZOrder	支持	不支持
Native DeltaLog	支持	不支持

6.2.29.2. 基础使用

本文为您介绍E-MapReduce中DeltaLake的配置信息及其常用命令的示例。

DeltaLake配置信息

EMR中DeltaLake的默认配置信息如下：

- Spark 2.X环境

```
spark.sql.extensions io.delta.sql.DeltaSparkSessionExtension
```

- Spark 3.X环境

```
spark.sql.extensions io.delta.sql.DeltaSparkSessionExtension
spark.sql.catalog.spark_catalog org.apache.spark.sql.delta.catalog.DeltaCatalog
```

常用命令

- 创建表

```
CREATE TABLE delta_table (id INT) USING delta;
```

- 插入数据

```
INSERT INTO delta_table VALUES 0,1,2,3,4;
```

- 覆盖写数据

```
INSERT OVERWRITE TABLE delta_table VALUES 5,6,7,8,9;
```

- 查询数据

```
SELECT * FROM delta_table;
```

- 更新数据

```
UPDATE delta_table SET id = id + 100 WHERE mod(id, 2) = 0;--给偶数ID加100。
```

- 删除数据

```
DELETE FROM delta_table WHERE mod(id, 2) = 0;--删除偶数ID的记录。
```

- Merge
  i. 创建Source表用于Merge操作。

  ```
  CREATE TABLE newData(id INT) USING delta;
  ```

  ii. 向表中插入数据。

  ```
  INSERT INTO newData VALUES 0,1,2,3,4,5,6,7,8,9;
  ```

  iii. 使用newData作为Source Merge进delta_table表，如果匹配到相同id的记录，id加100，没有匹配直接插入；

  ```
  MERGE INTO delta_table AS target
    USING newData AS source
    ON target.id = source.id
    WHEN MATCHED THEN UPDATE SET target.id = source.id + 100
    WHEN NOT MATCHED THEN INSERT *;
  ```

- 流式读数据
  i. 创建流式目标表。

  ```
  CREATE TABLE stream_debug_table (id INT);
  ```

  ii. 创建流。

  ```
  CREATE SCAN stream_delta_table on delta_table USING STREAM;
  ```

  > **说明** 本文示例中的delta_table为您已存在的delta表。

  iii. 流式写入目标表。

  ```
  CREATE STREAM job
  options (
    triggerType='ProcessingTime',
    checkpointLocation = '/tmp/streaming_read_cp'
  )
  INSERT INTO stream_debug_table
  SELECT *
  FROM stream_delta_table;
  ```

- 流式写数据
  i. 创建Kafka的管道表。

  ```
  CREATE TABLE IF NOT EXISTS kafka_topic
  USING kafka
  OPTIONS (
  kafka.bootstrap.servers = "${BOOTSTRAP_SERVERS}",
  subscribe = "${TOPIC_NAME}"
  );
  ```

  > **说明** 上述代码中的 `kafka.bootstrap.servers` 为Kafka集群中任一Kafka Broker组件的内网IP地址和端口。 `subscribe` 为Topic名称。

ii. 创建流。

```
CREATE SCAN stream_kafka_topic on kafka_topic USING STREAM;
```

iii. 流式写入delta表。

```
CREATE STREAM job
options (
  triggerType='ProcessingTime',
  checkpointLocation = '/tmp/streaming_read_cp'
)
INSERT INTO stream_debug_table
SELECT *
FROM stream_delta_table;
```

流式代码示例

1. 通过SSH方式连接集群，详情请参见登录集群。
2. 执行以下命令，启动streaming-sql。

```
streaming-sql
```

> **说明** 如果您已添加DeltaLake组件，则可以直接执行 `streaming-sql` 命令。如果集群内没有默认配置，您可以通过以下配置来使用Delta Lake。

```
streaming-sql --jars /path/to/delta-core_2.11-0.6.1.jar --conf spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension
```

3. 执行以下命令，创建流式目标表。

```
CREATE TABLE stream_debug_table (id INT) USING DELTA;
```

4. 执行以下命令，创建流。

```
CREATE SCAN stream_delta_table on delta_table USING STREAM;
```

5. 执行以下命令，以delta_table作为Source，流式写入目标表。

```
CREATE STREAM job
options (
  triggerType='ProcessingTime',
  checkpointLocation = '/tmp/streaming_read_cp'
)
INSERT INTO stream_debug_table
SELECT *
FROM stream_delta_table;
```

6. 您可以新开启一个streaming-sql客户端，向Source中插入新数据，并查询目标表的数据。
   i. 执行以下命令，验证Source存量写入。

   ```
   SELECT * FROM stream_debug_table;
   ```

   ii. 执行以下命令，插入新数据。

   ```
   INSERT INTO delta_table VALUES 801, 802;
   ```

   iii. 执行以下命令，查询插入的数据。

   ```
   SELECT * FROM stream_debug_table;
   ```

   iv. 执行以下命令，插入新数据。

   ```
   INSERT INTO delta_table VALUES 901, 902;
   ```

   v. 执行以下命令，查询插入的数据。

   ```
   SELECT * FROM stream_debug_table;
   ```

6.2.29.3. 高阶使用

6.2.29.3.1. 访问Delta表数据

在E-MapReduce中通过Spark创建的Delta表将自动同步到Hive元数据，您可以像使用其它表一样查询Delta表。您也可以通过Hive创建外表来查询Delta表。本文为您介绍如何通过Spark创建表和Hive创建外表的方式，在Hive、Presto和Impala访问Delta表的数据。

背景信息

- Delta表的DDL和DML等操作只能在Spark中执行。您可以通过Hive、Presto和Impala引擎查询Delta表数据，详情请参见通过Spark创建表访问查询Delta表。
- E-MapReduce也支持对已经存在的Delta数据，通过创建Hive外表来查询数据，详情请参见创建Hive外表。

通过Spark创建表访问查询Delta表

1. 进入Spark命令行。
   i. 使用SSH方式登录到集群主节点，详情请参见登录集群。
   ii. 执行以下命令，进入Spark命令行。

   ```
   spark-sql
   ```

2. 在Spark中创建并查询表数据。
   i. 执行以下命令，在Spark中创建Delta表。

   ```
   create table delta_table (id int) using delta location "/tmp/delta_table";
   ```

   ii. 执行以下命令，向表中插入数据。

   ```
   insert into delta_table values 0,1,2,3,4;
   ```

   iii. 执行以下命令，查看表数据。

   ```
   select * from delta_table;
   ```

   返回包含如下的信息。

   ```
   2
   3
   4
   0
   1
   Time taken: 1.847 seconds, Fetched 5 row(s)
   ```

3. 访问Delta表数据。
   - 通过Hive访问Delta表数据

     注意 EMR 3.x系列的EMR-3.37.0及后续版本，您可以在EMR控制台添加自定义参数，也可以在运行命令时设置参数。控制台添加参数时，在Hive服务的配置页面，如果使用的是Hive on MR，则添加参数名为hive.input.format，参数值为io.delta.hive.HiveInputFormat的配置项，如果使用的是Hive on Tez，则添加参数名为hive.tez.input.format，参数值为io.delta.hive.HiveInputFormat的配置项。添加操作请参见添加组件参数。

     a. 执行以下命令，进入Hive命令行。

     ```
     hive
     ```

     b. 执行以下命令，在Hive中查看Delta表的数据。

     ```
     select * from delta_table;
     ```

     返回如下信息。

     ```
     2
     3
     4
     0
     1
     Time taken: 2.937 seconds, Fetched: 5 row(s)
     ```

     说明 查看数据与在Spark中插入的数据一致，说明Hive已经成功访问了Delta表的数据。

   - 通过Presto访问Delta表数据

     a. 执行以下命令，进入Presto命令行。

     ```
     presto --server emr-header-1:9090 --catalog hive --schema default
     ```

b. 执行以下命令，在Presto中查看Delta表的数据。

```
select * from delta_table;
```

返回如下信息。

```
 id
----
  2
  3
  4
  0
  1
(5 rows)
```

> **说明** 查看数据与在Spark中插入的数据一致，说明Presto已经成功访问了Delta表的数据。

- 通过Impala访问Delta表数据

a. 执行以下命令，进入Impala命令行。

```
impala-shell
```

b. 执行以下命令，在Impala中查看Delta表的数据。

```
select * from delta_table;
```

返回如下信息。

```
+----+
| id |
+----+
| 2  |
| 3  |
| 4  |
| 0  |
| 1  |
+----+
```

> **说明** 查看数据与在Spark中插入的数据一致，说明Impala已经成功访问了Delta表的数据。

创建Hive外表

E-MapReduce也支持对已存在的Delta数据创建Hive外表来查询。示例如下。

```
CREATE EXTERNAL TABLE delta_tbl (name string, age int, city string)
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
WITH SERDEPROPERTIES("path" = "/path/to/delta")
STORED AS INPUTFORMAT 'io.delta.hive.DeltaInputFormat'
OUTPUTFORMAT 'io.delta.hive.DeltaOutputFormat'
LOCATION '/path/to/delta'
TBLPROPERTIES("spark.sql.sources.provider" = "delta");
```

如果您创建的集群是EMR 3.x系列的EMR-3.37.0及后续版本，您也可以通过Hive StorageHandler的方式创建外表。示例如下。

```
CREATE EXTERNAL TABLE delta_tbl123 (name string, age int, city string)
STORED BY 'io.delta.hive.DeltaStorageHandler'
WITH SERDEPROPERTIES("path" = "/path/to/delta")
LOCATION '/path/to/delta'
TBLPROPERTIES("spark.sql.sources.provider" = "delta");
```

> **说明** 建表语句中 `SERDEPROPERTIES` 设置 `path` ，以及 `TBLPROPERTIES` 中设置 `spark.sql.sources.provider` 是为了兼容后续通过SparkSQL的方式查询。通过该方式创建Delta外表后，在Hive、Presto和Impala查询表的方式，与通过Spark创建表访问查询Delta表方式相同，详情请参见步骤3。

6.2.29.4. 开发指南

6.2.29.4.1. Delta Lake参数

本文介绍一些Delta中比较重要的参数。

Delta的设置参数分为三类：

- Spark SQL设置，即设置SQL运行时的参数。
- 运行时参数，即可以在Session中动态设置的参数，以 `spark.databricks.delta.` 前缀开头。
- 非运行时参数，只能够在Spark的配置文件中配置为全局参数，或者建表时在 `TBLPROPERTIES` 中指定为表参数。表参数的优先级高于全局参数。当设置为全局参数时，配置前缀为 `spark.databricks.delta.properties.defaults.` ，当设置为 `TBLPROPERTIES` 时，前缀为 `delta.` 。

参数	描述
`spark.databricks.delta.snapshotPartitions`	默认值为10。 此参数为delta log元数据的partition数量。当delta log特别大时，需要增大此值，反之减小此值的设置。该值的大小对于delta table的解析性能影响较大。
`spark.databricks.delta.retentionDurationCheck.enabled`	默认值为true。 清理墓碑文件时是否进行安全期检查。 **警告** 如果您想删除近期合并过的小文件，可以设置此参数为false，来关闭此安全检查，但不建议您关闭此检查，这样可能会删除近期的数据而造成数据读写失败。
`spark.databricks.delta.schema.autoMerge.enabled`	默认值为false。 Delta有校验写入数据是否符合表定义Schema的功能，用于保证写入数据的正确性。当您数据的Schema发生变更后，需要在写入数据时在option中显示指定 `mergeSchema` 为true。如果您期望当数据Schema发生变化自动进行Schema的合并，请设置该值为true。但是我们仍然建议您使用显示指定的方式，而不是让它自动合并Schema。
`spark.databricks.delta.properties.defaults.deletedFileRetentionDuration` 或 `delta.deletedFileRetentionDuration`	默认值为interval 1 week。 Delta墓碑文件的安全期。清空未超过安全期内的墓碑文件将会抛出异常（如果 `spark.databricks.delta.retentionDurationCheck.enabled` 为true的话）。 **说明** 此值应当大于等于1个小时。
`spark.databricks.delta.properties.defaults.logRetentionDuration` 或 `delta.logRetentionDuration`	默认值为interval 30 days。 Delta log文件过期时间。Delta log过期被定义为： - 该log文件对应的数据文件已经做过了compaction。 - 该log文件超过了上述文件过期时间。每当Delta log进行checkpoint动作时，会检查是否有需要删除的过期文件，如果有，则删除这些过期文件以防Delta log文件无限增长。
`spark.sql.sources.parallelPartitionDiscovery.parallelism`	默认值为1000。 此参数为Delta扫描文件时所用的并行度。如果文件数量较少，则减小此值。目前仅使用在Vacuum中。如果设置不当，影响Vacuum扫描文件的效率。 **说明** 此参数为Spark SQL参数。

6.2.29.4.2. 批式读写

本文介绍Delta Lake如何批式读写数据。

建表并写入数据

- Scala

```
// 非分区表
data.write.format("delta").save("/tmp/delta_table")
// 分区表
data.write.format("delta").partitionedBy("date").save("/tmp/delta_table")
```

- SQL

```
-- 非分区表
CREATE TABLE delta_table (id INT) USING delta LOCATION "/tmp/delta_table";
INSERT INTO delta_table VALUES 0,1,2,3,4;
-- 分区表
CREATE TABLE delta_table (
id INT, date STRING) USING delta PARTITIONED BY (date) LOCATION "/tmp/delta_table";
INSERT INTO delta_table PARTITION (date='2019-11-11') VALUES 0,1,2,3,4;
-- 或者使用动态分区写入
INSERT INTO delta_table PARTITION (date) VALUES (0,'2019-11-01'),(1,'2019-11-02'),(2,'2019-11-05'),(3,'2019-11-08'),(4,'2
019-11-11');
```

追加数据

- Scala

```
// 非分区表
data.write.format("delta").mode("append").save("/tmp/delta_table")
// 分区表
data.write.format("delta").mode("append").save("/tmp/delta_table")
```

- SQL

```
-- 非分区表
INSERT INTO delta_table VALUES 0,1,2,3,4;
-- 分区表
INSERT INTO delta_table PARTITION (date='2019-11-11') VALUES 0,1,2,3,4;
-- 或者使用动态分区写入
INSERT INTO delta_table PARTITION (date) VALUES (0,'2019-11-01'),(1,'2019-11-02'),(2,'2019-11-05'),(3,'2019-11-08'),(4,'2
019-11-11');
```

覆盖数据

- Scala

```
// 非分区表
data.write.format("delta").mode("overwrite").save("/tmp/delta_table")
// 分区表
data.write.format("delta").mode("overwrite").option("replaceWhere", "date >= '2019-11-01' AND date <= '2019-11-11'").save
("/tmp/delta_table")
```

- SQL

```
INSERT OVERWRITE TABLE delta_table VALUES 0,1,2,3,4;
-- 分区表
INSERT OVERWRITE delta_table PARTITION (date='2019-11-11') VALUES 0,1,2,3,4;
-- 或者使用动态分区写入
INSERT OVERWRITE delta_table PARTITION (date) VALUES (0,'2019-11-01'),(1,'2019-11-02'),(2,'2019-11-05'),(3,'2019-11-08'),
(4,'2019-11-11');
```

读表

- Scala

```
spark.read.format("delta").load("/tmp/delta_table")
```

- SQL

```
SELECT * FROM delta_table;
```

历史版本访问

- Scala

```
df1 = spark.read.format("delta").option("timestampAsOf", timestamp_string).load("/tmp/delta_table")
df2 = spark.read.format("delta").option("versionAsOf", version).load("/tmp/delta_table")
```

- SQL

不支持。

6.2.29.4.3. 流式读写

本文介绍Delta Lake作为数据源和数据接收端如何流式读写数据。

Delta Table作为数据源（Source）

```
spark.readStream
  .format("delta")
  .option("maxFilesPerTrigger", 1000)
  .load("/tmp/delta_table")
```

`maxFilesPerTrigger` 指定了一个批次最多处理的文件数量，默认值为1000。

通常作为数据源的组件，数据一旦产生就会被下游消费，数据不会发生变化。但是Delta还兼顾了数据湖的角色，数据可能会被删除、更新，或者合并。目前Delta提供了两个选项来应对这种情况：

- ignoreDeletes：设置该选项为true后，对分区的删除动作不会有任何影响。
- ignoreChanges：设置该选项为true后，删除、更新或合并动作不会被特殊处理，但是这些动作产生的新文件会被当成新数据发送到下游。例如，某一个文件包含10000条数据，更新其中一条数据后，新文件有9999条旧数据和1条新数据。这9999条旧数据和1条新数据会被发送到下游。

Delta Table作为数据接收端（Sink）

- Append模式：该模式是Spark Streaming的默认工作模式。

```
df.writeStream
  .format("delta")
  .outputMode("append")
  .option("checkpointLocation", "/tmp/delta_table/_checkpoints")
  .start("/tmp/delta_table")
```

- Complete模式：在该模式下每一次batch的执行都会以全表覆盖的形式写目标表。例如，对于（id LONG, date DATE, name STRING, sales DOUBLE）这张表，您可以统计每个人的总销售额，将统计结果写入目标表，每个批次更新一次。

```
spark.readStream
  .format("delta")
  .load("/tmp/delta_table")
  .select("name","sales")
  .groupBy("name")
  .agg(sum("sales"))
  .writeStream
  .format("delta")
  .outputMode("complete")
  .option("checkpointLocation", "/tmp/delta_table_summary/_checkpoints")
  .start("/tmp/delta_table_summary")
```

6.2.29.4.4. 管理数据

本文为您介绍E-MapReduce（简称EMR）上的DeltaLake如何管理数据，包括删除、更新与合并数据等操作。

背景信息

EMR的DeltaLake支持以下操作管理数据。

功能	描述
删除数据	删除数据。
更新数据	更新数据。
合并数据	合并数据。
DESCRIBE HISTORY	该命令用于显示DeltaLake的详细操作历史。
CONVERT	该命令用于将Parquet格式的表转成Delta表。
OPTIMIZE	该命令通过合并小文件或ZOrder排序优化Delta表的数据布局，提升查询效率。

功能	描述
VACUUM	该命令可以删除表路径中不需要的，且超过指定时间的数据文件。
SAVEPOINT	该命令可以永久保存DeltaLake的历史版本。
ROLLBACK	该命令可以恢复到DeltaLake某个历史版本。

删除数据

- Scala

```
import io.delta.tables.
val deltaTable = DeltaTable.forPath(spark, "/tmp/delta_table")
deltaTable.delete("date < '2019-11-11'")
import org.apache.spark.sql.functions.
import spark.implicits.
deltaTable.delete(col("date") < "2019-11-11")
```

- SQL

```
DELETE FROM delta_table [AS t] [WHERE t.date < '2019-11-11'];
```

  - 暂不支持带有子查询的WHERE条件。但如果子查询为标量子查询且使用SQL，可以设置 `spark.sql.uncorrelated.scalar.subquery.preexecution.enabled` 为 `true` 后进行查询，例如：

    ```
    DELETE FROM delta_table WHERE t.date < (SELECT date FROM ref_table WHERE ....)
    ```

  - 如果您需要根据另一张表对目标表的匹配行进行删除（例如 `DELETE FROM target WHERE target.col = ref.col ...` ），请使用Merge语法。

> **说明** 使用 `DELETE` 命令，如果没有条件限制，则会删除所有数据。

更新数据

- Scala

```
import io.delta.tables._
val deltaTable = DeltaTable.forPath(spark, "/tmp/delta_table")
deltaTable.updateExpr(            //使用SQL字符串。
  "name = 'Robet'",
  Map("name" -> "'Robert'")
import org.apache.spark.sql.functions._
import spark.implicits._
deltaTable.update(                //使用SQL函数和隐式转换。
  col("name") === "Robet"),
  Map("name" -> lit("Robert"));
```

- SQL

```
UPDATE delta_table [AS t] SET t.id = t.id + 1 [WHERE t.date < '2019-11-11'];
```

  - 暂不支持带有子查询的WHERE条件。但如果子查询为标量子查询且使用SQL，可以设置 `spark.sql.uncorrelated.scalar.subquery.preexecution.enabled` 为 `true` 后进行查询，例如：

    ```
    UPDATE delta_table SET t.id = t.id + 1 WHERE t.date < (SELECT date FROM ref_table WHERE ....)
    ```

  - 如果要根据另一张表对目标表的匹配行进行更新（例如， `UPDATE target SET target.col = ref.col ...` 或 `WHERE target.col = ref.col ...` ），请使用Merge语法。

合并数据

- Scala

```
import io.delta.tables._
import org.apache.spark.sql.functions._
val updatesDF = ...  // define the updates DataFrame[date, id, name]
DeltaTable.forPath(spark, "/tmp/delta_table")
  .as("target")
  .merge(updatesDF.as("source"), "target.id = source.id")
  .whenMatched("target.name = 'should_update'")
  .updateExpr(Map("target.name" -> "source.name"))
  .whenMatched("target.name = 'should_delete'")
  .delete()
  .whenNotMatched("source.name = 'shoulde_insert'")
  .insertExpr(
    Map(
      "date" -> "updates.date",
      "eventId" -> "updates.eventId",
      "data" -> "updates.data"))
  .execute()
```

- SQL

```
MERGE INTO target AS t
USING source AS s
ON t.date = s.date
WHEN MATCHED [AND t.name = 'should_update'] THEN UPDATE SET target.name = source.name
WHEN MATCHED [AND t.name = 'should_delete'] THEN DELETE
WHEN NOT MATCHED [AND s.name = 'should_insert'] THEN INSERT (t.date, t.name, t.id) VALUES (s.date, s.name.s.id)
```

  - UPDATE子句和INSERT子句支持 `*` 语法，如果设置为 `UPDATE SET *` 或者 `INSERT *` ，则会更新或插入所有字段。
  - 暂不支持带有子查询的ON条件，但如果子查询为标量子查询的形式且使用SQL，可以设置 `spark.sql.uncorrelated.scalar.subquery.preexecution.enabled` 为 `true` 后进行查询。

DESCRIBE HISTORY

该命令用于显示DeltaLake的详细操作历史。

按照顺序展示版本号、操作时间、用户ID、用户名、操作类型、操作参数、作业信息、Notebook信息、集群、操作基于的前置版本、隔离等级、是否直接追加和操作Metrics等信息。

> **说明** 通常大多数信息显示为Null。

示例如下：

- 显示所有的操作记录。

```
DESC HISTORY dbName.tableName;
```

- 显示最新一条的操作记录。

```
DESC HISTORY dbName.tableName limit 1;
```

CONVERT

该命令用于将Parquet格式的表转成Delta表。

CONVERT遍历指定路径下的Parquet数据文件，推测表的Schema，生成Delta表需要的元数据信息。如果Parquet表本身是分区表，则需要额外指定分区字段和类型。

示例如下：

- 转换指定路径下的Parquet数据文件。

```
CONVERT TO DELTA parquet.`oss://region/path/to/tbl_without_partition`;
```

- 转换指定路径下的Parquet数据文件，并按照dt和hour进行分区。

```
CONVERT TO DELTA parquet.`oss://region/path/to/tbl_with_partition` PARTITIONED BY (dt string, hour int);
```

使用CONVERT后，仅将表路径构建为Delta表需要的格式，还没有将其注册成表，需要继续使用CREATE TABLE命令，且不需要指定建表字段和分区字段，具体示例如下。

```
CREATE TABLE tbl_without_partition
USING delta
LOCATION "oss://region/path/to/tbl_without_partition";
```

OPTIMIZE

该命令通过合并小文件或ZOrder排序优化Delta表的数据布局，提升查询效率。OPTIMIZ命令支持如下操作：

- 针对分区表，可以指定分区来进行优化。
- 可以指定非分区字段进行ZOrder排序，在进行正常Compact优化时调整数据布局。

示例如下：

- 进行全局优化。

```
OPTIMIZE dbName.tableName;
```

- 对2021-04-01之前的分区进行优化。

```
OPTIMIZE dbName.tableName WHERE date < '2021-04-01';
```

- 对2021-04-01之前的分区使用col2和col3列进行优化。

```
OPTIMIZE dbName.tableName WHERE date < '2021-04-01' ZORDER BY (col2, col3);
```

> 说明
> - 对于Streaming入湖场景，通常每个batch较小，会导致小文件较多，可以定期执行Optimize命令合并小文件。
> - 对于查询模式相对固定的场景，例如，除分区字段外，仅指定几个列作为查询条件时，可以采用Zorder方式优化。

VACUUM

该命令可以删除表路径中不需要的，且超过指定时间的数据文件。

EMR的DeltaLake定义数据文件不需要包含以下两部分：

- 当前最新版本关联到的数据文件。
- 执行过Savepoint的特定版本关联到的数据文件。

VACUUM命令可以通过两种方式指定删除多久前的数据文件：

- 通过参数 `delta.deletedFileRetentionDuration` 配置表属性，默认值为1周。
- 通过VACUUM命令指定，单位为小时。
- 语法

```
VACUUM (path=STRING | table=tableIdentifier) (RETAIN number HOURS)? (DRY RUN)?
```

- 示例如下：
  - 删除7天之前的数据文件。

    ```
    VACUUM dbName.tableName;
    ```

  - 删除24小时之前的数据文件。

    ```
    VACUUM dbName.tableName RETAIN 24 HOURS;
    ```

  - 显示待删除24小时之前的文件列表。

    ```
    VACUUM dbName.tableName RETAIN 24 HOURS DRY RUN;
    ```

    > 说明
    > - 根据您创建表的实际情况，可以定期执行VACUUM命令，节省存储空间。
    > - 实际执行VACUUM命令前，可以先通过 `DRY RUN` 命令，确认删除内容。

SAVEPOINT

该命令可以永久保存DeltaLake的历史版本。

DeltaLake会在每次执行CheckPoint（固定版本间隔，由参数 `delta.checkpointInterval` 决定）时清理掉log元数据文件（默认保留30天内的log元数据，由参数 `delta.logRetentionDuration` 决定）。通过VACUUM也会删除历史版本不再需要的数据文件。执行SAVEPOINT命令，可以永久避免log元数据和数据文件被删除，同时配合time-travel的能力，可以读取历史版本数据。

示例如下：

- 保存ID为0的版本。

```
CREATE SAVEPOINT delta.`/path/to/delta_tbl` VERSION AS OF 0;
```

- 保存指定时间之前最近的版本。

```
CREATE SAVEPOINT dbName.tableName TIMESTAMP AS OF "2021-04-01 10:00:00";
```

删除或查看SAVEPOINT操作记录的示例如下：

- 删除记录

```
DROP SAVEPOINT dbName.tableName TIMESTAMP AS OF "2021-04-01 10:00:00";
```

- 查看记录

  可以显示SAVEPOINT的版本号、版本提交时间、SAVEPOINT时间及其他信息。

```
SHOW SAVEPOINT dbName.tableName;
```

ROLLBACK

该命令可以恢复到DeltaLake某个历史版本。

如果指定要恢复到的历史版本不可重建（即缺失log元数据或者对应的数据文件），则抛出异常。

- 回滚到ID为0的版本。

```
ROLLBACK delta.`/path/to/delta_tbl` VERSION AS OF 0;
```

- 回滚到指定时间之前最近的版本。

```
ROLLBACK dbName.tableName TIMESTAMP AS OF "2021-04-01 10:00:00";
```

6.2.29.5. 最佳实践

6.2.29.5.1. 场景一：流式入库

目前支持流式入库的系统都基本遵循了一个思路，流式数据按照小批量数据写小文件到存储系统，然后定时对这些文件进行合并，从HIVE到Delta Lake无一例外（Kudu也可以做到流式入库，但是Kudu的存储是自己设计的，不属于前述的基于大数据存储系统之上的解决方案）。

流式入库演变

阶段	详细情况
以前	以前针对流式入库的需求，一般都是自己动手，事实表按照时间划分Partition，粒度比较细。例如，五分钟一个Partition，每当一个Partition运行完成，触发一个INSERT OVERWRITE动作，将该Partition内的文件合并重新写入分区。但是这么做有以下几个问题： • 缺少读写隔离，易造成读端失败或者产生数据准确性问题。 • 流式作业没有Exactly-Once保证，入库作业失败后需要人工介入，确保数据不会写重或者写漏（如果是SparkStreaming，有At-Least-Once保证）。 HIVE从0.13版本提供了事务支持，并且从2.0版本开始提供了HIVE Streaming功能来实现流式入库的支持。但是在实际使用HIVE Streaming功能的案例并不多见。其主要原因如下： • HIVE事务的实现修改了底层文件，导致公共的存储格式等仅能够被HIVE读取，导致很多使用SparkSQL、Presto等进行数据分析的用户无法使用该功能。 • HIVE事务目前仅支持ORC。 • HIVE的模式为Merge-on-read，需要对小文件进行Sort-Merge。小文件数量增多之后读性能急剧下降，所以用户需要及时进行小文件的合并。而小文件的合并作业经常失败，影响用户业务效率。 • HIVE这种模式无法拓展到Data Lake场景，仅仅停留在Data Warehouse场景。在Data Lake场景中，数据来源以及数据需求都是多样性的。
现在	有了Delta，可以很方便地应对流式入库的场景。只需要以下四个动作： 1. 建表。 2. 启动Spark Streaming任务写入数据。 3. 定时Optimize（例如：每个Partition写入完成）。 4. 定时Vacuum（例如：每天）。

Delta实例展示

从上游Kafka中读取数据，写入Delta表。上游Kafka准备一个Python脚本，不断向Kafka内发送数据。

```
#! /usr/bin/env python3
import json
import time
from kafka import KafkaProducer
from kafka.errors import KafkaError
bootstrap = ['emr-header-1:9092']
topic = 'delta_stream_sample'
def gnerator():
    id = 0
    line = {}
    while True:
        line['id'] = id
        line['date'] = '2019-11-11'
        line['name'] = 'Robert'
        line['sales'] = 123
        yield line
        id = id + 1
def sendToKafka():
    producer = KafkaProducer(bootstrap_servers=bootstrap)
    for line in gnerator():
        data = json.dumps(line).encode('utf-8')
        # Asynchronous by default
        future = producer.send(topic, data)
        # Block for 'synchronous' sends
        try:
            record_metadata = future.get(timeout=10)
        except KafkaError as e:
            # Decide what to do if produce request failed
            pass
        time.sleep(0.1)
sendToKafka()
```

为了方便，数据只有 `id` 不一样。

```
{"id": 0, "date": "2019-11-11", "name": "Robert", "sales": 123}
{"id": 1, "date": "2019-11-11", "name": "Robert", "sales": 123}
{"id": 2, "date": "2019-11-11", "name": "Robert", "sales": 123}
{"id": 3, "date": "2019-11-11", "name": "Robert", "sales": 123}
{"id": 4, "date": "2019-11-11", "name": "Robert", "sales": 123}
{"id": 5, "date": "2019-11-11", "name": "Robert", "sales": 123}
```

启动一个Spark Streaming作业，从Kafka读数据，写入Delta表。

- Scala
  - bash

    ```
    spark-shell --master local --use-emr-datasource
    ```

- scala

```
import org.apache.spark.sql.{functions, SparkSession}
import org.apache.spark.sql.types.DataTypes
import org.apache.spark.sql.types.StructField
val targetDir = "/tmp/delta_table"
val checkpointLocation = "/tmp/delta_table_checkpoint"
val bootstrapServers = "192.168.XX.XX:9092"
val topic = "delta_stream_sample"
val schema = DataTypes.createStructType(Array[StructField](
  DataTypes.createStructField("id", DataTypes.LongType, false),
  DataTypes.createStructField("date", DataTypes.DateType, false),
  DataTypes.createStructField("name", DataTypes.StringType, false),
  DataTypes.createStructField("sales", DataTypes.StringType, false)))
val lines = spark
    .readStream
    .format("kafka")
    .option("kafka.bootstrap.servers", bootstrapServers)
    .option("subscribe", topic)
    .option("maxOffsetsPerTrigger", 1000)
    .option("startingOffsets", "earliest")
    .option("failOnDataLoss", value = false)
    .load()
    .select(functions.from_json(functions.col("value").cast("string"), schema).as("json"))
    .select("json.*")
val query = lines.writeStream
    .outputMode("append")
    .format("delta")
    .option("checkpointLocation", checkpointLocation)
    .start(targetDir)
query.awaitTermination()
```

- SQL
  - bash

```
streaming-sql --master local --use-emr-datasource
```

  - SQL

```
CREATE TABLE IF NOT EXISTS kafka_table
USING kafka
OPTIONS(
kafka.bootstrap.servers='192.168.XX.XX:9092',
subscribe='delta_stream_sample'
);
CREATE TABLE IF NOT EXISTS delta_table (id LONG, `date` DATE, name STRING, sales STRING)
USING delta
LOCATION '/tmp/delta_table';
CREATE SCAN stream_kafka_table on kafka_table USING STREAM
OPTIONS(
maxOffsetsPerTrigger='1000',
startingOffsets='earliest',
failOnDataLoss=false
);
CREATE STREAM job
OPTIONS(
checkpointLocation='/tmp/delta_table_checkpoint'
)
INSERT INTO delta_table
SELECT
    content.id as id,
    content.date as date,
    content.name as name,
    content.sales as sales
FROM (
    SELECT from_json(CAST(value as STRING), 'id LONG, `date` DATE, name STRING, sales STRING') as content
    FROM stream_kafka_table
);
```

另新建一个spark-shell，确认已经读到数据。

- Scala

```
val df = spark.read.format("delta").load("/tmp/delta_table")
df.select("*").orderBy("id").show(10000)
```

- SQL

```
SELECT * FROM delta_table ORDER BY id LIMIT 10000;
```

现在已经写入了2285条数据。

```
|2295|2019-11-11|Robert|  123|
|2296|2019-11-11|Robert|  123|
|2297|2019-11-11|Robert|  123|
|2275|2019-11-11|Robert|  123|
|2276|2019-11-11|Robert|  123|
|2277|2019-11-11|Robert|  123|
|2278|2019-11-11|Robert|  123|
|2279|2019-11-11|Robert|  123|
|2280|2019-11-11|Robert|  123|
|2281|2019-11-11|Robert|  123|
|2282|2019-11-11|Robert|  123|
|2283|2019-11-11|Robert|  123|
|2284|2019-11-11|Robert|  123|
|2285|2019-11-11|Robert|  123|
+----+----------+------+-----+
```

Exactly-Once测试

将Spark Streaming作业停掉，再重新启动。重新读一下表，读数据正常的话，数据能够从上次断掉的地方衔接上。

- Scala

```
df.select("*").orderBy("id").show(10000)
```

- SQL

```
SELECT * FROM delta_table ORDER BY id LIMIT 10000;
```

```
|2878|2019-11-11|Robert|  123|
|2879|2019-11-11|Robert|  123|
|2880|2019-11-11|Robert|  123|
|2881|2019-11-11|Robert|  123|
|2882|2019-11-11|Robert|  123|
|2883|2019-11-11|Robert|  123|
|2884|2019-11-11|Robert|  123|
|2885|2019-11-11|Robert|  123|
|2886|2019-11-11|Robert|  123|
|2887|2019-11-11|Robert|  123|
|2888|2019-11-11|Robert|  123|
|2889|2019-11-11|Robert|  123|
|2890|2019-11-11|Robert|  123|
|2891|2019-11-11|Robert|  123|
```

6.2.29.5.2. 场景二：数据同步

数据同步是指数仓或者数据湖内的数据与上游业务库内的数据保持同步的状态。当上游业务库内的数据发生变更之后，下游的数仓/数据湖立即感知到数据变化，并将数据变化同步过来。在数据库中，这类场景称为Change Data Capture（CDC）场景。

背景信息

CDC的实现方案比较多，但是大多是在数据库领域，相应的工具也比较多。在大数据领域，这方面的实践较少，也缺乏相应的标准和技术实现。通常您需要选择已有的引擎，利用它们的能力自己搭建一套CDC方案。常见的方案大概分为下面两类：

- 定期批量Merge方式：上游原始表捕获增量更新，将更新的数据输出到一个新的表中，下游仓库利用MERGE或UPSERT语法将增量表与已有表进行合并。这种方式要求表具有主键或者联合主键，且实时性也较差。另外，这种方法一般不能处理DELETE的数据，实际上用删除原表重新写入的方式支持了DELETE，但是相当于每次都重新写一次全量表，性能不可取，还需要有一个特殊字段来标记数据是否属于增量更新数据。
- 上游源表输出binlog（这里我们指广义的binlog，不限于MySQL），下游仓库进行binlog的回放。这种方案一般需要下游仓库能够具有实时回放的能力。但是可以将row的变化作为binlog输出，这样，只要下游具备INSERT、UPDATE、DELETE的能力就可以了。不同于第一种方案，这种方案可以和流式系统结合起来。binlog可以实时地流入注入Kafka的消息分发系统，下游仓库订阅相应的Topic，实时拉取并进行回放。

批量更新方式

此方案适用于没有Delete且实时性要求不那么高的场景。

1. 建立一张MySQL表，插入一部分数据。

```
CREATE TABLE sales(id LONG, date DATE, name VARCHAR(32), sales DOUBLE, modified DATETIME);
INSERT INTO sales VALUES (1, '2019-11-11', 'Robert', 323.00, '2019-11-11 12:00:05'), (2, '2019-11-11', 'Lee', 500.00, '
2019-11-11 16:11:46'), (3, '2019-11-12', 'Robert', 136.00, '2019-11-12 10:23:54'), (4, '2019-11-13', 'Lee', 211.00, '20
19-11-13 11:33:27');
SELECT * FROM sales;
```

```
+------+------------+--------+-------+---------------------+
| id   | date       | name   | sales | modified            |
+------+------------+--------+-------+---------------------+
| 1    | 2019-11-11 | Robert |   323 | 2019-11-11 12:00:05 |
| 2    | 2019-11-11 | Lee    |   500 | 2019-11-11 16:11:46 |
| 3    | 2019-11-12 | Robert |   136 | 2019-11-12 10:23:54 |
| 4    | 2019-11-13 | Lee    |   211 | 2019-11-13 11:33:27 |
+------+------------+--------+-------+---------------------+
```

> **说明** `modified` 就是我们上文提到的用于标识数据是否属于增量更新数据的字段。

2. 将MySQL表的内容全量导出到HDFS。

```
sqoop import --connect jdbc:mysql://emr-header-1:3306/test --username root --password EMRroot1234 -table sales -m1 --ta
rget-dir /tmp/cdc/staging_sales
hdfs dfs -ls /tmp/cdc/staging_sales
Found 2 items
-rw-r-----   2 hadoop hadoop          0 2019-11-26 10:58 /tmp/cdc/staging_sales/_SUCCESS
-rw-r-----   2 hadoop hadoop        186 2019-11-26 10:58 /tmp/cdc/staging_sales/part-m-00000
```

3. 建立delta表，并导入MySQL表的全量数据。

```
-- `LOAD DATA INPATH`语法对delta table不可用，先建立一个临时外部表。
CREATE TABLE staging_sales (id LONG, date STRING, name STRING, sales DOUBLE, modified STRING) USING csv LOCATION '/tmp/
cdc/staging_sales/';
CREATE TABLE sales USING delta LOCATION '/user/hive/warehouse/test.db/test' SELECT * FROM staging_sales;
SELECT * FROM sales;
1    2019-11-11    Robert    323.0    2019-11-11 12:00:05.0
2    2019-11-11    Lee    500.0    2019-11-11 16:11:46.0
3    2019-11-12    Robert    136.0    2019-11-12 10:23:54.0
4    2019-11-13    Lee    211.0    2019-11-13 11:33:27.0
--删除临时表。
DROP TABLE staging_sales;
```

切换到命令行删除临时目录。

```
hdfs dfs -rm -r -skipTrash /tmp/cdc/staging_sales/    # 删除临时目录。
```

4. 在原MySQL表做一些操作，插入更新部分数据。

```
-- 注意DELETE的数据无法被Sqoop导出，因而没办法合并到目标表中
-- DELETE FROM sales WHERE id = 1;
UPDATE sales SET name='Robert',modified=now() WHERE id = 2;
INSERT INTO sales VALUES (5, '2019-11-14', 'Lee', 500.00, now());
SELECT * FROM sales;
```

```
+------+------------+--------+-------+---------------------+
| id   | date       | name   | sales | modified            |
+------+------------+--------+-------+---------------------+
| 1    | 2019-11-11 | Robert |   323 | 2019-11-11 12:00:05 |
| 2    | 2019-11-11 | Robert |   500 | 2019-11-26 11:08:34 |
| 3    | 2019-11-12 | Robert |   136 | 2019-11-12 10:23:54 |
| 4    | 2019-11-13 | Lee    |   211 | 2019-11-13 11:33:27 |
| 5    | 2019-11-14 | Lee    |   500 | 2019-11-26 11:08:38 |
+------+------------+--------+-------+---------------------+
```

5. sqoop导出更新数据。

```
sqoop import --connect jdbc:mysql://emr-header-1:3306/test --username root --password EMRroot1234 -table sales -m1 --ta
rget-dir /tmp/cdc/staging_sales --incremental lastmodified --check-column modified --last-value "2019-11-20 00:00:00"
hdfs dfs -ls /tmp/cdc/staging_sales/
Found 2 items
-rw-r-----   2 hadoop hadoop          0 2019-11-26 11:11 /tmp/cdc/staging_sales/_SUCCESS
-rw-r-----   2 hadoop hadoop         93 2019-11-26 11:11 /tmp/cdc/staging_sales/part-m-00000
```

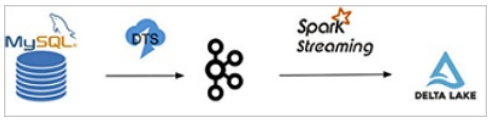
6. 为更新数据建立临时表，然后MERGE到目标表。

```
CREATE TABLE staging_sales (id LONG, date STRING, name STRING, sales DOUBLE, modified STRING) USING csv LOCATION '/tmp/
cdc/staging_sales/';
MERGE INTO sales AS target
USING staging_sales AS source
ON target.id = source.id
WHEN MATCHED THEN UPDATE SET *
WHEN NOT MATCHED THEN INSERT *;
SELECT * FROM sales;
1    2019-11-11    Robert    323.0    2019-11-11 12:00:05.0
3    2019-11-12    Robert    136.0    2019-11-12 10:23:54.0
2    2019-11-11    Robert    500.0    2019-11-26 11:08:34.0
5    2019-11-14    Lee    500.0    2019-11-26 11:08:38.0
4    2019-11-13    Lee    211.0    2019-11-13 11:33:27.0
```

实时同步

实时同步的方式对场景的限制没有第一种方式多，例如，DELETE数据也能处理，不需要修改业务模型增加一个额外字段。但是这种方式实现较为复杂，如果binlog的输出不标准的话，您还需要写专门的UDF来处理binlog数据。例如RDS MySQL输出的binlog，以及Log Service输出的binlog格式上就不相同。

在这个例子中，我们使用阿里云RDS MySQL版作为源库，使用阿里云DTS服务将源库的binlog数据实时导出到Kafka集群，您也可以选择开源的Maxwell或Canal等。 之后我们定期从Kafka读取binlog并存放到OSS或HDFS，然后用Spark读取该binlog并解析出Insert、Update、Delete的数据，最后用Delta的Merge API将源表的变动更新到Delta表，其链路如下图所示。



1. 首先开通RDS MySQL服务，设置好相应的用户、Database和权限（RDS的具体使用请参见概述）。建立一张表并插入一些数据。

```
-- 该建表动作可以在RDS控制台页面方便地完成，这里展示最后的建表语句。
CREATE TABLE `sales` (
  `id` bigint(20) NOT NULL,
  `date` date DEFAULT NULL,
  `name` varchar(32) DEFAULT NULL,
  `sales` double DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
-- 插入部分数据，并确认。
INSERT INTO sales VALUES (1, '2019-11-11', 'Robert', 323.00), (2, '2019-11-11', 'Lee', 500.00), (3, '2019-11-12', 'Robe
rt', 136.00), (4, '2019-11-13', 'Lee', 211.00);
SELECT * FROM `sales` ;
```

```
<img src="./img/rds_tbl.png" width = "400" alt="RDS Table" align=center />
```

	id	date	name	sales
1	1	2019-11-11	Robert	323
2	2	2019-11-11	Lee	500
3	3	2019-11-12	Robert	136
4	4	2019-11-13	Lee	211

2. 建立一个EMR Kafka集群（如果已有EMR Kafka集群的话请跳过），并在Kafka集群上创建一个名为sales的topic：

   bash

```
kafka-topics.sh --create --zookeeper emr-header-1:2181,emr-header-2:2181,emr-header-3:2181 --partitions 1 --replication
-factor 1 --topic sales
```

3. 开通DTS服务（如果未开通的话），并创建一个同步作业，源实例选择RDS MySQL，目标实例选择Kafka。
4. 配置DTS的同步链路，将RDS的sales table同步至Kafka，目标topic选择sales。正常的话，可以在Kafka的机器上看到数据。
5. 编写Spark Streaming作业，从Kafka中解析binlog，利用Delta的MERGE API将binlog数据实时回放到目标Delta表。DTS导入到Kafka的binlog数据的样子如下，其每一条记录都表示了一条数据库数据的变更。 详情请参见附录：Kafka内binlog格式窥探。

```
|字段名称|值|
|:--|:--|
|recordid|1|
|source|{"sourceType": "MySQL", "version": "0.0.0.0"}|
|dbtable|delta_cdc.sales|
|recordtype|INIT|
|recordtimestamp|1970-01-01 08:00:00|
|extratags|{}|
|fields|["id","date","name","sales"]|
|beforeimages|{}|
|afterimages|{"sales":"323.0","date":"2019-11-11","name":"Robert","id":"1"}|
```

说明 这里最重要的字段是 `recordtype` 、 `beforeimages` 、 `afterimages` 。其中 `recordtype` 是该行记录对应的动作，包含 `INIT` 、 `UPDATE` 、 `DELETE` 、 `INSERT` 几种。 `beforeimages` 为该动作执行前的内容， `afterimages` 为动作执行后的内容。

- Scala
  - bash

```
spark-shell --master yarn --use-emr-datasource
```

  - scala

```
import io.delta.tables._
import org.apache.spark.internal.Logging
import org.apache.spark.sql.{AnalysisException, SparkSession}
import org.apache.spark.sql.expressions.Window
import org.apache.spark.sql.functions._
import org.apache.spark.sql.types.{DataTypes, StructField}
val schema = DataTypes.createStructType(Array[StructField](
  DataTypes.createStructField("id", DataTypes.StringType, false),
  DataTypes.createStructField("date", DataTypes.StringType, true),
  DataTypes.createStructField("name", DataTypes.StringType, true),
  DataTypes.createStructField("sales", DataTypes.StringType, true)
))
//初始化delta表中INIT类型的数据。
def initDeltaTable(): Unit = {
  spark.read
      .format("kafka")
      .option("kafka.bootstrap.servers", "192.168.XX.XX:9092")
      .option("subscribe", "sales")
      .option("failOnDataLoss", value = false)
      .load()
      .createTempView("initData")
  // 对于DTS同步到Kafka的数据，需要avro解码，EMR提供了dts_binlog_parser的UDF来处理此问题。
  val dataBatch = spark.sql(
      """
        |SELECT dts_binlog_parser(value)
        |AS (recordID, source, dbTable, recordType, recordTimestamp, extraTags, fields, beforeImages, afterImages)
        |FROM initData
      """.stripMargin)
  // 选择INIT类型的数据作为初始数据。
  dataBatch.select(from_json(col("afterImages").cast("string"), schema).as("jsonData"))
      .where("recordType = 'INIT'")
      .select(
        col("jsonData.id").cast("long").as("id"),
        col("jsonData.date").as("date"),
        col("jsonData.name").as("name"),
        col("jsonData.sales").cast("decimal(7,2)")).as("sales")
      .write.format("delta").mode("append").save("/delta/sales")
}
try {
  DeltaTable.forPath("/delta/sales")
} catch {
  case e: AnalysisException if e.getMessage().contains("is not a Delta table") =>
    initDeltaTable()
}
spark.readStream
    .format("kafka")
    .option("kafka.bootstrap.servers", "192.168.XX.XX:9092")
    .option("subscribe", "sales")
```

```
    .option("startingOffsets", "earliest")
    .option("maxOffsetsPerTrigger", 1000)
    .option("failOnDataLoss", value = false)
    .load()
    .createTempView("incremental")
// 对于DTS同步到Kafka的数据，需要avro解码，EMR提供了dts_binlog_parser的UDF来处理此问题。
val dataStream = spark.sql(
  """
    |SELECT dts_binlog_parser(value)
    |AS (recordID, source, dbTable, recordType, recordTimestamp, extraTags, fields, beforeImages, afterImages)
    |FROM incremental
  """.stripMargin)
val task = dataStream.writeStream
    .option("checkpointLocation", "/delta/sales_checkpoint")
    .foreachBatch(
      (ops, id) => {
        // 该window function用于提取针对某一记录的最新一条修改。
        val windowSpec = Window
            .partitionBy(coalesce(col("before_id"), col("id")))
            .orderBy(col("recordId").desc)
        // 从binlog中解析出recordType, beforeImages.id, afterImages.id, afterImages.date, afterImages.name, afterIma
ges.sales。
        val mergeDf = ops
            .select(
              col("recordId"),
              col("recordType"),
              from_json(col("beforeImages").cast("string"), schema).as("before"),
              from_json(col("afterImages").cast("string"), schema).as("after"))
            .where("recordType != 'INIT'")
            .select(
              col("recordId"),
              col("recordType"),
              when(col("recordType") === "INSERT", col("after.id")).otherwise(col("before.id")).cast("long").as("be
fore_id"),
              when(col("recordType") === "DELETE", col("before.id")).otherwise(col("after.id")).cast("long").as("id
"),
              when(col("recordType") === "DELETE", col("before.date")).otherwise(col("after.date")).as("date"),
              when(col("recordType") === "DELETE", col("before.name")).otherwise(col("after.name")).as("name"),
              when(col("recordType") === "DELETE", col("before.sales")).otherwise(col("after.sales")).cast("decimal
(7,2)").as("sales")
            )
            .select(
              dense_rank().over(windowSpec).as("rk"),
              col("recordType"),
              col("before_id"),
              col("id"),
              col("date"),
              col("name"),
              col("sales")
            )
            .where("rk = 1")
        //merge条件，用于将incremental数据和delta表数据做合并。
        val mergeCond = "target.id = source.before_id"
        DeltaTable.forPath(spark, "/delta/sales").as("target")
            .merge(mergeDf.as("source"), mergeCond)
            .whenMatched("source.recordType='UPDATE'")
            .updateExpr(Map(
              "id" -> "source.id",
              "date" -> "source.date",
              "name" -> "source.name",
              "sales" -> "source.sales"))
            .whenMatched("source.recordType='DELETE'")
            .delete()
            .whenNotMatched("source.recordType='INSERT' OR source.recordType='UPDATE'")
            .insertExpr(Map(
              "id" -> "source.id",
              "date" -> "source.date",
              "name" -> "source.name",
              "sales" -> "source.sales"))
            .execute()
      }
    ).start()
```

```
task.awaitTermination()
```

- SQL
  - bash

    ```
    streaming-sql --master yarn --use-emr-datasource
    ```

  - SQL

```
CREATE TABLE kafka_sales
USING kafka
OPTIONS(
kafka.bootstrap.servers='192.168.XX.XX:9092',
subscribe='sales'
);
CREATE TABLE delta_sales(id long, date string,  name string, sales decimal(7, 2))
USING delta
LOCATION '/delta/sales';
INSERT INTO delta_sales
SELECT CAST(jsonData.id AS LONG), jsonData.date, jsonData.name, jsonData.sales
FROM (
  SELECT from_json(CAST(afterImages as STRING), 'id STRING, date DATE, name STRING, sales STRING') as jsonData
  FROM (
    SELECT dts_binlog_parser(value) AS (recordID, source, dbTable, recordType, recordTimestamp, extraTags, fields,
beforeImages, afterImages)
    FROM kafka_sales
  ) binlog WHERE recordType='INIT'
) binlog_wo_init;
CREATE SCAN incremental on kafka_sales USING STREAM
OPTIONS(
startingOffsets='earliest',
maxOffsetsPerTrigger='1000',
failOnDataLoss=false
);
CREATE STREAM job
OPTIONS(
checkpointLocation='/delta/sales_checkpoint'
)
MERGE INTO delta_sales as target
USING (
  SELECT
  recordId,
  recordType,
  before_id,
  id,
  date,
  name,
  sales
  FROM(
    SELECT
    recordId,
    recordType,
    CASE WHEN recordType = "INSERT" then after.id else before.id end as before_id,
    CASE WHEN recordType = "DELETE" then CAST(before.id as LONG) else CAST(after.id as LONG) end as id,
    CASE WHEN recordType = "DELETE" then before.date else after.date end as date,
    CASE WHEN recordType = "DELETE" then before.name else after.name end as name,
    CASE WHEN recordType = "DELETE" then CAST(before.sales as DECIMAL(7, 2)) else CAST(after.sales as DECIMAL(7, 2)
) end as sales,
    dense_rank() OVER (PARTITION BY coalesce(before.id,after.id) ORDER BY recordId DESC) as rank
    FROM (
      SELECT
      recordId,
      recordType,
      from_json(CAST(beforeImages as STRING), 'id STRING, date STRING, name STRING, sales STRING') as before,
      from_json(CAST(afterImages as STRING), 'id STRING, date STRING, name STRING, sales STRING') as after
      FROM (
        select dts_binlog_parser(value) as (recordID, source, dbTable, recordType, recordTimestamp, extraTags, fiel
ds, beforeImages, afterImages) from incremental
      ) binlog WHERE recordType != 'INIT'
    ) binlog_wo_init
  ) binlog_extract WHERE rank=1
) as source
ON target.id = source.before_id
WHEN MATCHED AND source.recordType='UPDATE' THEN
UPDATE SET id=source.id, date=source.date, name=source.name, sales=source.sales
WHEN MATCHED AND source.recordType='DELETE' THEN
DELETE
WHEN NOT MATCHED AND (source.recordType='INSERT' OR source.recordType='UPDATE') THEN
INSERT (id, date, name, sales) values (source.id, source.date, source.name, source.sales);
```

6. 待上一步骤中的Spark Streaming作业启动后，我们尝试读一下这个Delta Table。

   scala

```
spark.read.format("delta").load("/delta/sales").show
+---+----------+------+------+
| id|      date|  name| sales|
+---+----------+------+------+
|  1|2019-11-11|Robert|323.00|
|  2|2019-11-11|   Lee|500.00|
|  3|2019-11-12|Robert|136.00|
|  4|2019-11-13|   Lee|211.00|
+---+----------+------+------+
```

   在RDS控制台执行下列四条命令并确认结果，注意我们对于 `id = 2` 的记录update了两次，理论上最终结果应当为最新一次修改。

```
DELETE FROM sales WHERE id = 1;
UPDATE sales SET sales = 150 WHERE id = 2;
UPDATE sales SET sales = 175 WHERE id = 2;
INSERT INTO sales VALUES (5, '2019-11-14', 'Robert', 233);
SELECT * FROM sales;
```

	id	date	name	sales
1	2	2019-11-11	Lee	150
2	3	2019-11-12	Robert	136
3	4	2019-11-13	Lee	211
4	5	2019-11-14	Robert	233

   重新读一下Delta表，发现数据已经更新了， `且id=2` 的结果为最后一次的修改：

```
spark.read.format("delta").load("/delta/sales").show
+---+----------+------+------+
| id|      date|  name| sales|
+---+----------+------+------+
|  5|2019-11-14|Robert|233.00|
|  3|2019-11-12|Robert|136.00|
|  4|2019-11-13|   Lee|211.00|
|  2|2019-11-11|   Lee|175.00|
+---+----------+------+------+
```

最佳实践

随着数据实时流入，Delta内的小文件会迅速增多。针对这种情况，有两种解决方案：

- 对表进行分区。一方面，写入多数情况下是针对最近的分区，历史分区修改往往频次不是很高，这个时候对历史分区进行compaction操作，compaction因事务冲突失败的可能性较低。另一方面，带有分区谓词的查询效率较不分区的情况会高很多。
- 在流式写入的过程中，定期进行compation操作。例如，每过10个mini batch进行一次compaction。这种方式不存在compaction由于事务冲突失败的问题，但是由于compaction可能会影响到后续mini batch的时效性，因此采用这种方式要注意控制compaction的频次。

对于时效性要求不是那么高的场景，又担心compation因事务冲突失败，可以采用如下所示处理。在这种方式中，binlog的数据被定期收集到OSS上（可以通过DTS到Kafka然后借助kafka-connect-oss将binlog定期收集到OSS，也可以采用其他工具），然后启动spark批作业读取OSS上的binlog，一次性的将binlog合并到Delta Lake。其流程图如下所示。

> **说明** 虚线部分可替换为其他可能方案。

附录：Kafka内binlog格式窥探

DTS同步到Kafka的binlog是avro编码的。如果要探查其文本形式，我们需要借助EMR提供的一个avro解析的UDF：dts_binlog_parser。

- Scala
  - bash

```
spark-shell --master local --use-emr-datasource
```

- scala

  在启动的spark-shell中执行以下命令。

```
spark.read
    .format("kafka")
    .option("kafka.bootstrap.servers", "192.168.XX.XX:9092")
    .option("subscribe", "sales")
    .option("maxOffsetsPerTrigger", 1000)
    .load()
    .createTempView("kafkaData")
val kafkaDF = spark.sql("SELECT dts_binlog_parser(value) FROM kafkaData")
kafkaDF.show(false)
```

- SQL
  - bash

```
streaming-sql --master local --use-emr-datasource
```

  - SQL

```
CREATE TABLE kafkaData
USING kafka
OPTIONS(
kafka.bootstrap.servers='192.168.XX.XX:9092',
subscribe='sales'
);
SELECT dts_binlog_parser(value) FROM kafkaData;
```

最终显示结果如下所示。

```
+--------+----------------------------------------------+--------------+----------+-------------------+---------+----------
-----------------+------------+-------------------------------------------------------------------------+
|recordid|source                                        |dbtable       |recordtype|recordtimestamp    |extratags|fields
|beforeimages|afterimages                                                                                 |
+--------+----------------------------------------------+--------------+----------+-------------------+---------+----------
-----------------+------------+-------------------------------------------------------------------------+
|1       |{"sourceType": "MySQL", "version": "0.0.0.0"}|delta_cdc.sales|INIT      |1970-01-01 08:00:00|{}       |["id","dat
e","name","sales"]|{}          |{"sales":"323.0","date":"2019-11-11","name":"Robert","id":"1"}|
|2       |{"sourceType": "MySQL", "version": "0.0.0.0"}|delta_cdc.sales|INIT      |1970-01-01 08:00:00|{}       |["id","dat
e","name","sales"]|{}          |{"sales":"500.0","date":"2019-11-11","name":"Lee","id":"2"}   |
|3       |{"sourceType": "MySQL", "version": "0.0.0.0"}|delta_cdc.sales|INIT      |1970-01-01 08:00:00|{}       |["id","dat
e","name","sales"]|{}          |{"sales":"136.0","date":"2019-11-12","name":"Robert","id":"3"}|
|4       |{"sourceType": "MySQL", "version": "0.0.0.0"}|delta_cdc.sales|INIT      |1970-01-01 08:00:00|{}       |["id","dat
e","name","sales"]|{}          |{"sales":"211.0","date":"2019-11-13","name":"Lee","id":"4"}   |
+--------+----------------------------------------------+--------------+----------+-------------------+---------+----------
-----------------+------------+-------------------------------------------------------------------------+
```

6.2.29.5.3. 场景三：冷热分层

本文介绍冷数据的特点和适应场景，通过表格存储Tablestore和Delta Lake结合示例，演示数据的冷热分层。冷热分层可以充分利用计算和存储资源，以低成本承载更优质服务。

背景信息

在海量大数据场景下，随着业务和数据量的不断增长，性能和成本的权衡成为大数据系统设计面临的关键挑战。

Delta Lake是新型数据湖方案，推出了数据流入、数据组织管理、数据查询和数据流出等特性，同时提供了数据的ACID和CRUD操作。通过结合Delta Lake和上下游组件，您可以搭建出一个便捷、易用、安全的数据湖架构。在数据湖架构设计中，通常会应用HTAP（Hybrid Transaction and Analytical Process）体系结构，通过合理地选择分层存储组件和计算引擎，既能支持海量数据分析和快速的事务更新写入，又能有效地降低冷热数据分离的成本。

更多介绍请参见结构化大数据分析平台设计、面向海量数据的极致成本优化-云HBase的一体化冷热分离和云上如何做冷热数据分离。

冷热数据

数据按照实际访问的频率可以分为热数据、温数据和冷数据。其中冷数据的数据量较大，很少被访问，甚至整个生命周期都不会被访问。

冷热数据的区分方式如下：

- 按照数据的创建时间：通常，数据写入初期，用户的关注度较高且访问频繁，此时的数据为热数据。但随着时间的推移，旧数据访问频率会越来越低，仅存在少量查询，甚至完全不查询，此时数据为冷数据。

  常见于交易类数据、时序监控和IM聊天等场景。

- 按照访问热度：采用业务打标或系统自动识别等方式，按照数据的访问热度来区分冷热数据。

  例如，某旧博客突然被大量访问。此时不应该按照时间区分，而是应该按照具体的业务和数据分布规律来区分冷热数据。

  > **说明** 本文主要讨论按照数据创建时间的冷热数据分层。

冷数据特点

- 数据量大：相对于热数据，冷数据通常需要保存较长时间，甚至永久保存。
- 成本管控：数据量大且访问频率较低，不宜投入过多成本。
- 性能要求低：相较于普通的TP请求查询，无需在毫秒级别返回。冷数据的查询可以接受数十秒甚至更长时间返回结果，或者可以进行异步处理。
- 操作简单：通常，冷数据都是执行批量写入和删除操作，没有更新操作。

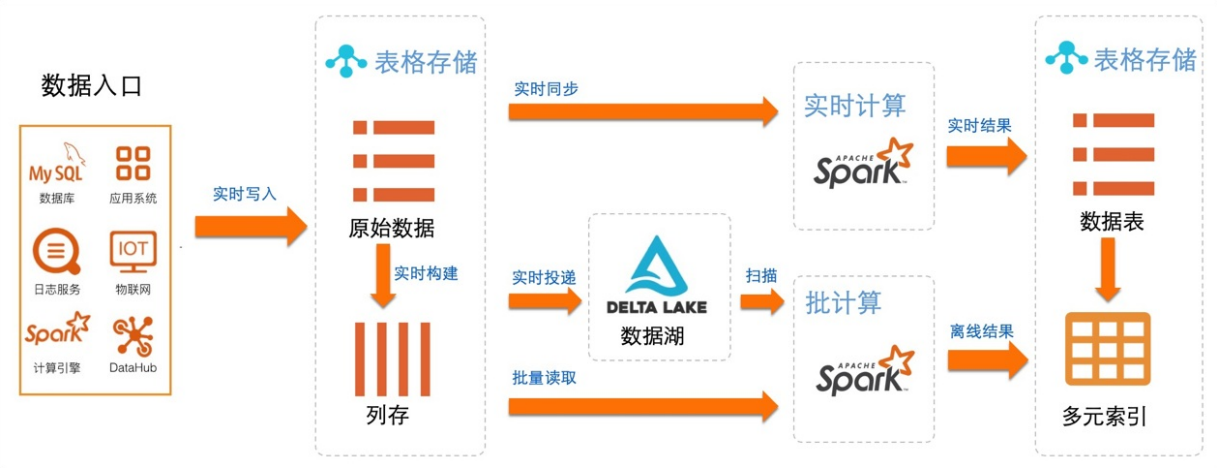
  当查询数据时，您只需要读取指定条件的数据，且查询条件不会过于复杂。

适用场景

- 时序类数据场景：时序类数据天然具备时间属性，数据量大，且仅执行追加操作。示例如下：
  - IM场景：通常用户会查询最近若干条聊天记录，只有在特殊需求的时候才会查询历史数据。例如钉钉。
  - 监控场景：通常用户只会查看近期的监控，只有在调查问题或者制定报表时才会查询历史数据。例如云监控。
  - 账单场景：通常用户只会查询最近几天或者一个月内的账单，不会查询超过一年以上的账单。例如支付宝。
  - 物联网场景：通常设备近期上报的数据是热点数据，会经常被分析，而历史数据的分析频率都较低。例如IoT。
- 归档类场景：对于读写简单，查询复杂的数据，您可以定期归档数据至成本更低的存储组件或更高压缩比的存储介质中，以达到降低成本的目的。

海量结构化数据Delta Lake架构

针对结构化冷热分层的数据场景，阿里巴巴集团推出了海量结构化数据的Delta Lake架构。



基于Tablestore的通道服务，原始数据可以利用变更数据捕获CDC（Change Data Capture）技术写入多种存储组件中。

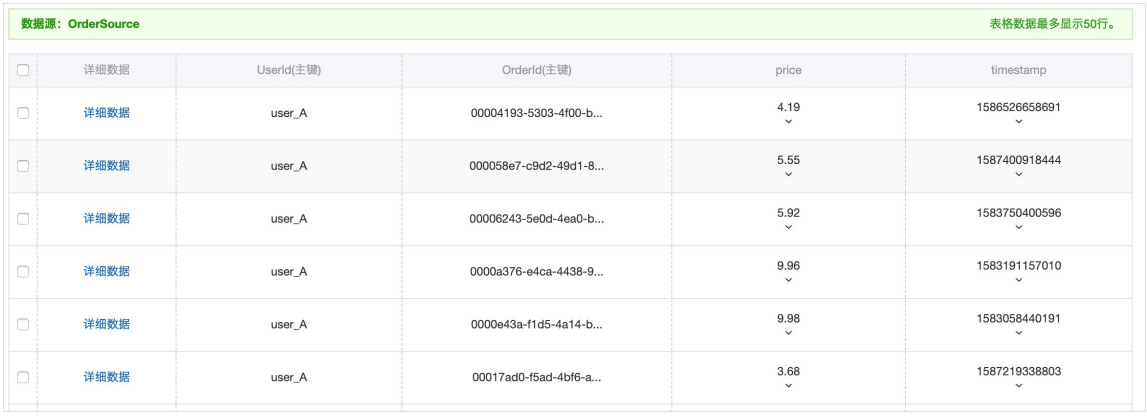
示例

本示例结合Tablestore和Delta Lake，进行数据的冷热分层。

1. 实时流式投递。

i. 创建数据源表。

数据源表是原始订单表OrderSource，有两个主键UserId（用户ID）和OrderId（订单ID），两个属性列price（价格）和timestamp（订单时间）。使用Tablestore SDK的BatchWrite接口写入订单数据，订单的时间戳的时间范围为最近90天（本示例的模拟时间范围为2020-02-26~2020-05-26），共计写入3112400条。



数据源：OrderSource　　表格数据最多显示50行。

详细数据	UserId(主键)	OrderId(主键)	price	timestamp
详细数据	user_A	00004193-5303-4f00-b...	4.19	1586526658691
详细数据	user_A	000058e7-c9d2-49d1-8...	5.55	1587400918444
详细数据	user_A	00006243-5e0d-4ea0-b...	5.92	1583750400596
详细数据	user_A	0000a376-e4ca-4438-9...	9.96	1583191157010
详细数据	user_A	0000e43a-f1d5-4a14-b...	9.98	1583058440191
详细数据	user_A	00017ad0-f5ad-4bf6-a...	3.68	1587219338803

在模拟订单写入时，对应Tablestore表属性列的版本号也会被设置为相应的时间戳。通过配置表上的TTL属性，当写入数据的保留时长超过设置的TTL时，系统会自动清理对应版本号的数据。

ii. 在Tablestore控制台上创建增量通道。

利用增量通道提供的CDC技术，同步新增的主表数据至Delta。通道ID用于后续的SQL配置。

通道列表　　刷新　创建通道

服务说明：通道服务是基于TableStore数据接口之上的全增量一体化服务，它通过一组Tunnel Service API和SDK为用户提供了增量、全量和增量加全量三种类型的分布式数据实时消费通道。通过为数据表建立Tunnel Service数据通道，用户可以简单地实现对表中历史存量和新增数据的消费处理。

通道名	通道ID	通道类型	通道状态	增量通道最新同步时间	是否过期	操作
test	324c6...	增量	增量处理	2020-05-26 21:01:26	false	展示通道分区列表 | 刷新 | 删除

通道分区列表

通道名：test　　通道分区总数：14

排序：默认 | 通道分区ID | 客户端ID | 类型 | 状态 | 消费统计 | 同步时间

通道分区ID	客户端ID	类型	状态	消费统计	增量通道分区最新同步时间	操作
0dceb759-...		增量	打开	183240	2020-05-26 21:01:26	模拟消费
1a5bc96b-...		增量	打开	182464	2020-05-26 21:01:26	模拟消费
3c3e6254-...		增量	打开	182034	2020-05-26 21:01:26	模拟消费
3d72694c-...		增量	打开	182375	2020-05-26 21:01:26	模拟消费
44de40f8-...		增量	打开	182901	2020-05-26 21:01:26	模拟消费

iii. 在EMR集群的Header节点，启动 `streaming-sql` 交互式命令行。

```
streaming-sql --master yarn --use-emr-datasource --num-executors 16 --executor-memory 4g --executor-cores 4
```

执行以下命令，创建源表和目的表。

```
// 1. 创建源表。
DROP TABLE IF EXISTS order_source;
CREATE TABLE order_source
USING tablestore
OPTIONS(
endpoint="http://vehicle-test.cn-hangzhou.vpc.tablestore.aliyuncs.com",
access.key.id="",
access.key.secret="",
instance.name="vehicle-test",
table.name="OrderSource",
catalog='{"columns": {"UserId": {"col": "UserId", "type": "string"}, "OrderId": {"col": "OrderId", "type": "string"
},"price": {"col": "price", "type": "double"}, "timestamp": {"col": "timestamp", "type": "long"}}}',
);
// 2. 创建Delta Lake Sink: delta_orders
DROP TABLE IF EXISTS delta_orders;
CREATE TABLE delta_orders(
UserId string,
OrderId string,
price double,
timestamp long
)
USING delta
LOCATION '/delta/orders';
// 3. 在源表上创建增量SCAN视图。
CREATE SCAN incremental_orders ON order_source USING STREAM
OPTIONS(
tunnel.id="324c6bee-b10d-4265-9858-b829a1b71b4b",
maxoffsetsperchannel="10000");
// 4. 执行Stream作业，实时同步Tablestore CDC数据至Delta Lake。
CREATE STREAM orders_job
OPTIONS (
checkpointLocation='/delta/orders_checkpoint',
triggerIntervalMs='3000'
)
MERGE INTO delta_orders
USING incremental_orders AS delta_source
ON delta_orders.UserId=delta_source.UserId AND delta_orders.OrderId=delta_source.OrderId
WHEN MATCHED AND delta_source.__ots_record_type__='DELETE' THEN
DELETE
WHEN MATCHED AND delta_source.__ots_record_type__='UPDATE' THEN
UPDATE SET UserId=delta_source.UserId, OrderId=delta_source.OrderId, price=delta_source.price, timestamp=delta_sour
ce.timestamp
WHEN NOT MATCHED AND delta_source.__ots_record_type__='PUT' THEN
INSERT (UserId, OrderId, price, timestamp) values (delta_source.UserId, delta_source.OrderId, delta_source.price, d
elta_source.timestamp);
```

各操作含义如下。

操作	描述
创建Tablestore源表	创建order_source源表。 OPTIONS参数中的 `catalog` 是表字段的Schema定义（本示例对应UserId、OrderId、price和timestamp四列）。
创建Delta Lake Sink表	创建delta_orders目的表。 LOCATION中指定的是Delta文件存储的位置。
在Tablestore源表上创建增量SCAN视图	创建incremental_orders的流式视图。 ■ `tunnel.id`：步骤1.b中创建的增量通道ID。 ■ `maxoffsetsperchannel`：通道每个分区可以写入数据的最大数据量。
启动Stream作业进行实时投递	根据Tablestore的主键列（UserId和OrderId）进行聚合，同时根据CDC日志的操作类型（PUT，UPDATE，DELETE），转化为对应的Delta操作。 `__ots_record_type__` 是Tablestore流式Source提供的预定义列，表示行操作类型。

2. 查询冷热数据。

   通常，您可以保存热数据至Tablestore表中进行高效的TP查询，保存冷数据或是全量数据至Delta中。通过配置Tablestore表的生命周期（TTL），您可以灵活地控制热数据量。

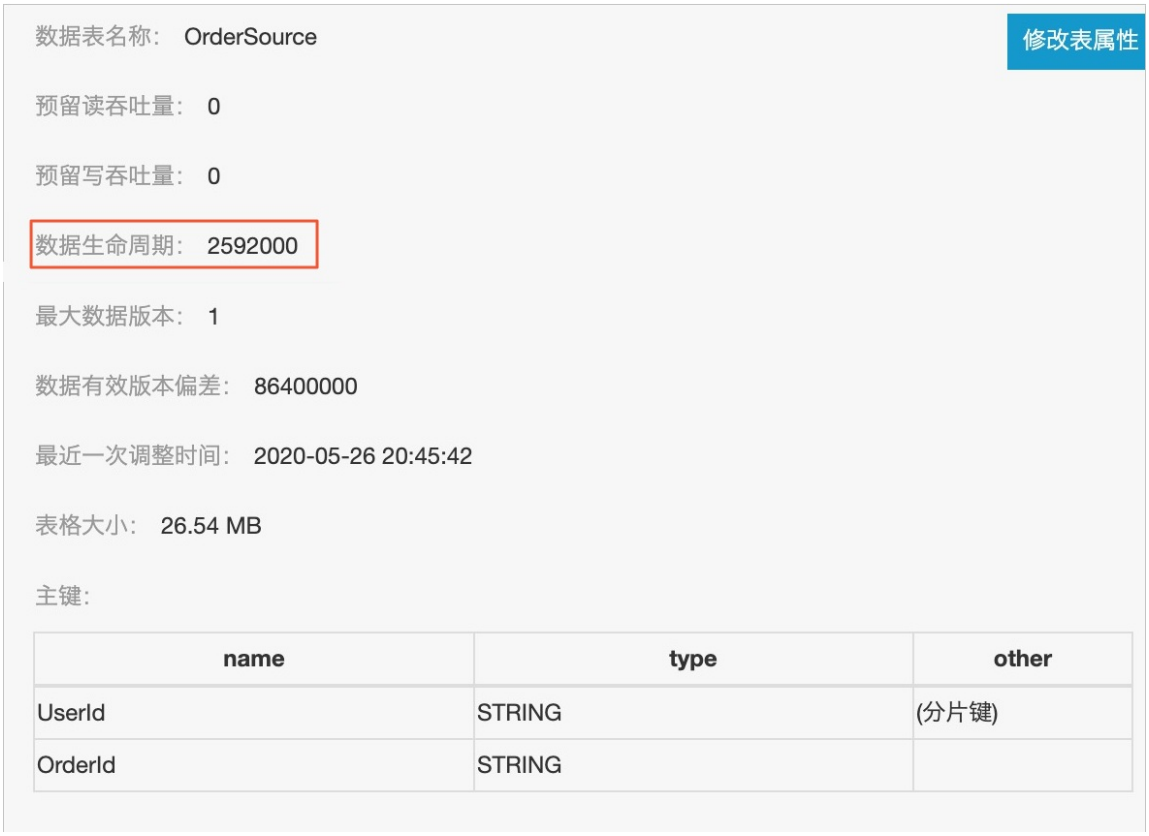
   i. 配置主表的TTL前，查询源表（order_source）和目的表（delta_orders）。

   此时两边的查询结果一致。

```
spark-sql> SELECT COUNT(*) FROM order_source;
3112400
Time taken: 7.85 seconds, Fetched 1 row(s)
spark-sql> SELECT COUNT(*) FROM delta_orders;
3112400
Time taken: 5.004 seconds, Fetched 1 row(s)
spark-sql> SELECT COUNT(*) FROM order_source WHERE price > 5;
1554430
Time taken: 9.51 seconds, Fetched 1 row(s)
spark-sql> SELECT COUNT(*) FROM delta_orders WHERE price > 5;
1554430
Time taken: 6.45 seconds, Fetched 1 row(s)
spark-sql>
```

   ii. 配置Tablestore的TTL为最近30天。

   Tablestore中的热数据只有最近30天的数据，而Delta中依旧保留的是全量数据，以达到冷热分层的目的。



iii. 冷热分层后，再次查询源表（order_source）和目的表（delta_orders）。

冷热分层后热数据为1017004条，冷数据（全量数据）保持不变，仍然为3112400条。

```
spark-sql> SELECT COUNT(*) FROM delta_orders;
3112400
Time taken: 5.823 seconds, Fetched 1 row(s)
spark-sql> SELECT COUNT(*) FROM order_source;
1016912
Time taken: 4.083 seconds, Fetched 1 row(s)
spark-sql> SELECT COUNT(*) FROM order_source where timestamp < 1587868748000;
0
Time taken: 5.316 seconds, Fetched 1 row(s)
spark-sql> SELECT COUNT(*) FROM delta_orders where timestamp < 1587868748000;
2060732
Time taken: 8.846 seconds, Fetched 1 row(s)
spark-sql>
```

6.2.29.5.4. 场景四：Slowly Changing Dimension

业务数据随着时间在不断变化，如果您要对数据进行分析，则需要考虑如何存储和管理数据。其中数据中随着时间变化的维度被称为Slowly Changing Dimension（SCD）。E-MapReduce根据实际的数仓场景定义了基于固定粒度的缓慢变化维（G-SCD）。本文为您介绍G-SCD的具体解决方案及如何通过G-SCD处理维度的数据。

背景信息

Slowly Changing Dimension（SCD）即缓慢变化维，是随着时间变化的维度。在数据仓库中存储和管理当前和历史的数据，就需要考虑如何处理缓慢变化维，因此SCD被认为是跟踪维度变化的关键ETL任务之一。

根据处理维度新值的方式，SCD被分为以下三种类型。

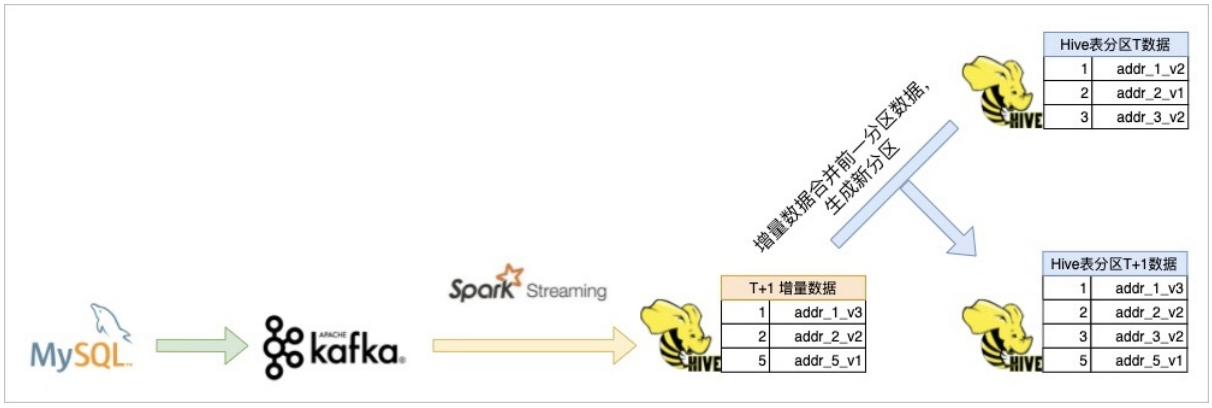
类型	描述
直接覆盖（Type 1）	直接覆盖原值，不保留历史记录。该方式无法分析历史变化的信息。
添加维度行（Type 2）	保留所有历史值。当属性值有变化时，都会新增一条记录，并且需要标记当前记录有效，同时修改前一个有效记录的有效性字段。通常可以通过起始时间、截止时间标识记录的有效性。
添加属性列（Type 3）	通过额外的字段仅保留前一个版本的值。针对需要分析历史信息的属性添加一列，记录该属性变化前的值，而本属性字段则记录最新的值。

SCD处理维度新值的三种方式不能覆盖业务的实际场景，所以E-MapReduce根据业务实际数仓场景提出了G-SCD（Based-Granularity Slowly Changing Dimension），即基于固定粒度（或者业务快照）的缓慢变化维。G-SCD按照固定的时间粒度生成一份业务快照数据，其中时间粒度可以是天、小时或者分钟等，同时支持按照时间粒度查询对应时间段的数据。

在传统的数仓体系下，基于Hive表的实现有以下两个解决方案可以考虑，但各有弊端。

解决方案	存在的问题
流式构建T+1时刻的增量数据表，和离线表的T时刻分区数据做合并，生成离线表T+1分区。	存储资源浪费。
保存离线的基础表，每个业务时刻的增量数据独立保存，在查询数据时合并基础表和增量表。	查询性能差。

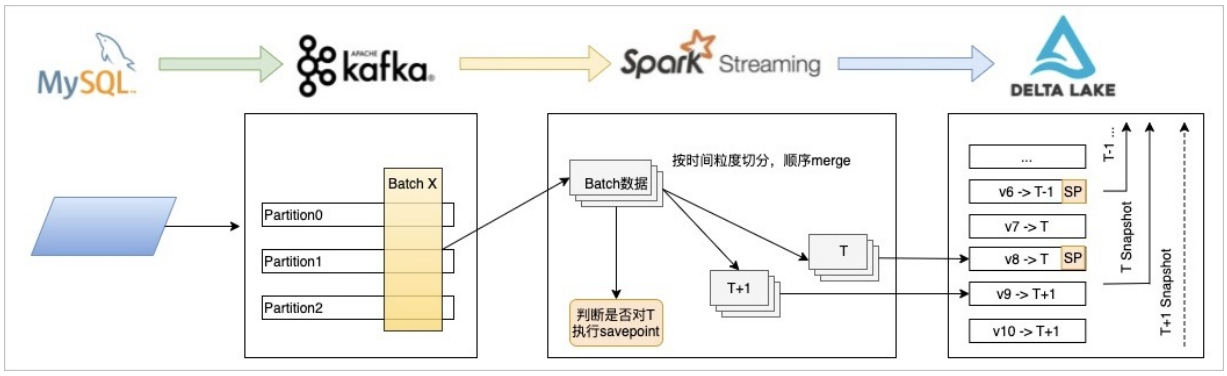
其中按T保留全量数据的解决方案如下图所示。



为了解决上述两个解决方案存在的问题，阿里云E-MapReduce团队基于Delta Lake提供了G-SCD的解决方案，即G-SCD on Delta Lake。G-SCD on Delta Lake方案与SCD的Type 2方案类似，两者之间的相同点和不同点如下表所示。

方案	相同点	不同点
SCD的Type 2方案	保留历史所有信息。	每次属性值有变化，都会新增一条记录。
G-SCD on Delta Lake方案		GSCD on Delta Lake在具体实现上不是通过新增记录的形式保留信息，而是借助Delta Lake本身的Versioning特性，通过Time-Travel的能力追溯具体的快照数据。

G-SCD on Delta Lake方案如下图所示。



G-SCD解决方案的优势如下：

- 流批一体：不需要增量表和基础表两张表。
- 存储资源节省：不需要按时间粒度保留历史全量数据。
- 查询性能高：借助Delta Lake的Optimize、Zorder和DataSkipping的能力，提升查询性能。
- SQL使用兼容性高：保留原来实现的SQL语句，和利用分区实现快照的方式一样，可以使用类似的分区字段查询对应时间粒度内的快照数据。

SCD简介

G-SCD概念和解决方案

前提条件

已创建集群，详情请参见创建集群。

使用限制

- 需要保证Kafka内同一个Partition内的数据严格有序。
- 数据按Key分区，保证同一Key必须落到同一个Kafka的Partition。

操作流程

1. 步骤一：创建G-SCD表

   创建G-SCD表，按照要求配置需要的参数。

2. 步骤二：处理数据

   您可以根据业务数据的情况，选择使用流式写入或者批量写入的方式进行数据的处理。示例中通过两次批量写入代替流式写入的方式模拟G-

SCD on Delta Lake的数据处理。

3. 步骤三：验证数据写入结果

   通过查询语句，验证数据是否写入成功。

步骤一：创建G-SCD表

创建G-SCD表的示例如下，该表会在步骤二：处理数据使用。

```
CREATE TABLE target (id Int, body String, dt string)
USING delta
TBLPROPERTIES (
  "delta.gscdTypeTable" = "true",
  "delta.gscdGranularity" = "1 day",
  "delta.gscdColumnFormat" = "yyyy-MM-dd",
  "delta.gscdColumn" = "dt"
);
```

参数说明如下表所示。

参数	说明
delta.gscdTypeTable	定义当前表是否为G-SCD Delta Lake表，本文示例需要设置为true。当该值设置为false时则表示该表为普通表，无法使用G-SCD的相关功能。
delta.gscdGranularity	业务快照粒度，例如：1 day、1 hour、30 minutes等。
delta.gscdColumnFormat	业务快照粒度的格式，支持格式如下： • yyyyMM • yyyyMMdd • yyyyMMddHH • yyyyMMddHHmm • yyyy-MM • yyyy-MM-dd • yyyy-MM-dd HH • yyyy-MM-dd HH:mm
delta.gscdColumn	定义查询时，表示业务快照版本的字段。当前字段也需要在Schema内定义，并且必须为String类型。

步骤二：处理数据

您可以根据业务数据的情况，选择使用流式写入或者批量写入的方式进行数据的处理。

- 流式写入

```
CREATE TABLE IF NOT EXISTS gscd_kafka_table
USING kafka
OPTIONS(
  kafka.bootstrap.servers = 'localhost:9092',
  subscribe = 'xxxxxx'
);
CREATE SCAN gscd_stream ON gscd_kafka_table USING STREAM
OPTIONS (
  `watermark.time` = 'floor(ts/1000)'  --- 定义源头watermark时间表达式，单位为秒。
);
CREATE STREAM delta_job
OPTIONS (
  triggerType = 'ProcessingTime',
  checkpointLocation = '/path/to/checkpoint'
)
MERGE INTO gscd_target_table AS target
USING (
  SELECT *, from_unixtime(ts/1000, 'yyyy-MM-dd') AS dt FROM gscd_stream
) AS source
ON source.id = target.id AND target.dt = source.dt
WHEN MATCHED THEN update set *
WHEN NOT MATCHED THEN insert *;
```

> 注意 在上述SQL语句中，watermark的使用原理及注意事项如下：
> - 为了在流作业中自动触发Savepoint，需要在 `CREATE SCAN` 语句中指定watermark时间表达式。
> - watermark表示流作业源头的时间值，单位为秒。
> - watermark时间会生成作为delta.gscdColumn字段的值，当watermark时间达到delta.gscdGranularity边界时（示例中定义的为1 day），会自动触发Savepoint。
> - watermark时间要求在同一个Partition内递增有序。

- 批量写入

  一般场景下，通过流式写入已经可以满足。但当数据异常时，G-SCD on Delta Lake的方案同时提供了回滚Rollback的能力，并可以使用批量离线写入修复数据。修复完成后，执行Savepoint，永久保留当前Version。

```
MERGE INTO GSCD("2021-01-01") gscd_target_table
USING gscd_source_table
ON source.id = target.id
WHEN MATCHED THEN UPDATE SET body = source.body
WHEN NOT MATCHED THEN INSERT(id, body) VALUES(source.id, source.body);
```

  在上述SQL语句中，`GSCD ("2021-01-01")` 的语法表示要写入的数据所属的业务粒度值。

> 注意 批量写入不支持同一个作业写入多个业务粒度数据。如果存在这种情况，需要提前进行拆分。

为了帮助您快速使用G-SCD处理维度数据，本文给出详细的示例，具体操作步骤如下。

1. 模拟源数据。

```
CREATE TABLE s1 (id Int, body String) USING delta;
CREATE TABLE s2 (id Int, body String) USING delta;
INSERT INTO s1 VALUES (1, "addr_1_v1"), (2, "addr_2_v1"), (3, "addr_3_v1");
INSERT INTO s2 VALUES (2, "addr_2_v2"), (4, "addr_1_v1");
```

2. 通过使用两次批量写入，代替流式写入的方式模拟G-SCD on Delta Lake的数据处理。
   i. 第一次批量写入，然后创建对应时间粒度的Savepoint。

```
-- 第一次批量写入。
MERGE INTO GSCD ("2021-01-01") target as target
USING s1 as source
ON source.id = target.id
WHEN MATCHED THEN UPDATE SET body = source.body
WHEN NOT MATCHED THEN INSERT(id, body) VALUES(source.id, source.body);
-- 创建2021-01-01的Savepoint。
CREATE SAVEPOINT target GSCD('2021-01-01');
```

   ii. 第二次批量写入，然后创建对应时间粒度的Savepoint。

```
-- 第二次批量写入。
MERGE INTO GSCD ("2021-01-02") target as target
USING s2 as source
ON source.id = target.id
WHEN MATCHED THEN UPDATE SET body = source.body
WHEN NOT MATCHED THEN INSERT(id, body) VALUES(source.id, source.body);
-- 创建2021-01-02的Savepoint。
CREATE SAVEPOINT target GSCD('2021-01-02');
```

步骤三：验证数据写入结果

通过查询语句，验证数据是否写入成功。查询在步骤二：处理数据示例中两次批量写入的数据，具体操作如下。

1. 执行以下命令，查询第一次批量写入的数据。

```
select id, body from target where dt = '2021-01-01';
```

> 注意
> - 查询数据时，可以使用正常的SQL语法。
> - 查询数据时，必须指定gscdColumn字段作为查询条件，并且必须为 `=` 表达式，例如 `dt = '2021-01-01'` 。

2. 执行以下命令，查询第二次批量写入的数据。

```
select id, body from target where dt = '2021-01-02';
```

如果能够查询到写入的数据，则表明数据写入成功。执行上述查询命令后，返回结果如下图所示。

```
spark-sql> select id, body from target where dt='2021-01-01';
1       addr_1_v1
2       addr_2_v1
3       addr_3_v1
Time taken: 4.282 seconds, Fetched 3 row(s)
spark-sql> select id, body from target where dt='2021-01-02';
1       addr_1_v1
2       addr_2_v2
3       addr_3_v1
4       addr_1_v1
Time taken: 1.1 seconds, Fetched 4 row(s)
```

6.2.29.6. 常见问题

本文汇总了DeltaLake使用时的常见问题。

- 为什么建表失败？
- 流式写入Delta时产生了很多的小文件怎么办？
- Optimize执行时间很长是什么原因？
- 为什么Optimize失败了？应该如何处理？
- 执行了Optimize，为什么还有很多小文件？
- 执行了Vacuum，为什么还有很多小文件？
- 如果想删除最近产生的小文件（这些小文件已经被合并），应该如何处理？
- 执行了Vacuum，为什么还有很多的Delta log文件？
- 有没有自动触发Optimize或Vacuum的机制？

为什么建表失败？

Delta建表需要制定LOCATION，这种表在Spark中为外表。建表时，如果目标目录不存在，即创建一张全新的表，理论上不会出现这种情况。如果LOCATION已经存在，那么基于此LOCATION建表应当确保，建表语句的Schema与LOCATION内Delta log中定义的Schema相同。

流式写入Delta时产生了很多的小文件怎么办？

用Spark Streaming写数据到Delta，本质上是执行一系列的mini batch，一个batch会产生一个或者多个文件。由于batch size通常较小，因此Spark Streaming连续运行会产生相当数量的小文件。解决方法有两种：

- 如果实时性要求不高，建议增大mini batch的trigger size。
- 定期运行Optimize，对表进行合并小文件的操作。

Optimize执行时间很长是什么原因？

如果长时间没有进行Optimize操作，Delta内可能会累积相当数量的小文件，此时运行Optimize可能执行时间会比较长。因此建议设置定时任务来定期触发Optimize动作。

为什么Optimize失败了？应该如何处理？

Optimize会有删除历史数据和写新数据的动作。由于Delta采用的乐观锁机制，写事务在提交的时候，其中一个写事务会失败。尤其是一个流式作业在不断地更新Delta内的数据（例如：CDC场景），此时Optimize失败的概率会更大（注意：如果流式作业仅仅是新增数据而不涉及删除或者更新，Optimize不会失败）。建议用户对表进行基于时间的分区，每当一个分区完成，对该分区进行 Optimize操作。

执行了Optimize，为什么还有很多小文件？

Optimize是合并小文件，但是被合并的小文件不会被立即删除。因为Delta有访问历史的功能，因此如果要访问合并之前的历史版本，这些小文件会被用到。如果要删除这些小文件，请使用Vacuum命令。

执行了Vacuum，为什么还有很多小文件？

Vacuum动作是清理已经合并过的且已经超出了安全期的小文件。默认安全期为7天。如果小文件没有被合并过，或者合并过的小文件尚在安全期之内，Vacuum不会将之删除。

如果想删除最近产生的小文件（这些小文件已经被合并），应该如何处理？

不建议删除时间过近的小文件，因为Delta的历史访问功能可能会用到这些小文件。如果确实要这么做，有两种做法：

- 关闭安全期检查：`spark.databricks.delta.retentionDurationCheck.enabled=false`，这个设置可以在启动spark任务时作为参数传入。

- 修改全局的安全期为一个较小的值：例如在 `spark-defaults.conf` 中设置 `spark.databricks.delta.properties.defaults.deletedFileRetentionDuration interval 1 hour` 。

执行了Vacuum，为什么还有很多的Delta log文件？

Vacuum动作是合并数据文件，并非合并Delta log文件。Delta log文件的合并和清理是Delta自动做的，每经历10个提交，会自动触发一次Delta log的合并，合并之后同时检查超出安全期的log文件，如果超出，则删除。默认Delta log的安全期为30天。

有没有自动触发Optimize或Vacuum的机制？

Delta仅仅是一个库，而非运行时，因此尚没有自动化的机制，但可以设置定时任务定期来触发Optimize或Vacuum的机制。

6.2.30. Presto

6.2.30.1. 概述

Presto是一个开源的分布式SQL查询引擎，适用于交互式分析查询。

背景信息

本文为您介绍以下内容：

- 基本特性
- 系统组成
- 应用场景
- 产品优势
- 基本概念
- 更多参考

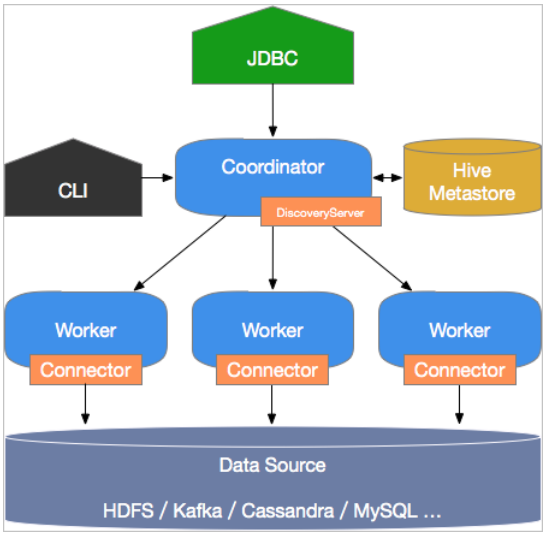
基本特性

Presto使用Java语言进行开发，具备易用、高性能和强扩展能力等特点，具体如下：

- 完全支持ANSI SQL。
- 支持丰富的数据源：
  - Hive
  - Cassandra
  - Kafka
  - MongoDB
  - MySQL
  - PostgreSQL
  - SQL Server
  - Redis
  - Redshift
  - 本地文件
- 支持高级数据结构，具体如下：
  - 数组和Map数据
  - JSON数据
  - GIS数据
  - 颜色数据
- 功能扩展能力强，提供了多种扩展机制：
  - 扩展数据连接器
  - 自定义数据类型
  - 自定义SQL函数
- 流水线：基于Pipeline处理模型数据在处理过程中实时返回给用户。
- 监控接口完善：
  - 提供友好的Web UI，可视化的呈现查询任务执行过程。
  - 支持JMX协议。

系统组成

Presto的系统组成如下图所示。



Presto是典型的M/S架构的系统，由一个Coordinator节点和多个Worker节点组成。 Coordinator负责如下工作：

- 接收用户查询请求，解析并生成执行计划，下发Worker节点执行。
- 监控Worker节点运行状态，各个Worker节点与Coordinator节点保持心跳连接，汇报节点状态。
- 维护MetaStore数据。

Worker节点负责执行下发到任务，通过连接器读取外部存储系统到数据，进行处理，并将处理结果发送给Coordinator节点。

应用场景

Presto是定位在数据仓库和数据分析业务的分布式SQL引擎，适合以下应用场景：

- ETL
- Ad-Hoc查询
- 海量结构化数据或半结构化数据分析
- 海量多维数据聚合或报表分析

> 注意 Presto是一个数仓类产品，因为其对事务支持有限，所以不适合在线业务场景。

产品优势

E-MapReduce（简称EMR）中的Presto与开源Presto比较，还具备如下优势：

- 即买即用，快速完成上百节点的Presto集群搭建。
- 弹性扩容简单操作。
- 与EMR软件栈完美结合，支持处理存储在OSS的数据。
- 无需运维，E-MapReduce提供一站式服务。

基本概念

数据模型即数据的组织形式。Presto使用Catalog、Schema和Table三层结构来管理数据。

- Catalog

  一个Catalog可以包含多个Schema，物理上指向一个外部数据源，可以通过Connector访问该数据源。一次查询可以访问一个或多个Catalog。

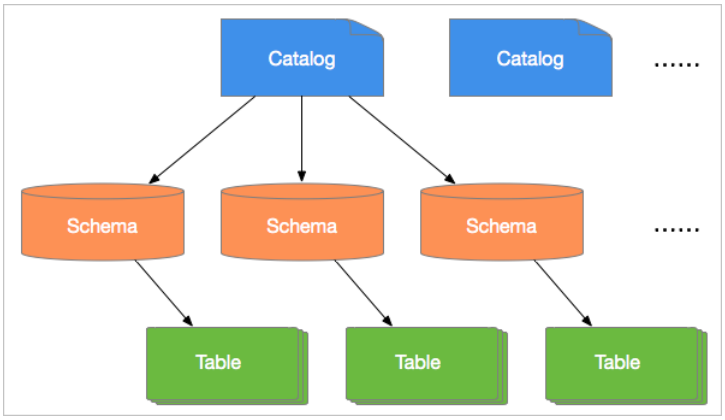
- Schema

  相当于一个数据库实例，一个Schema包含多张数据表。

- Table

  数据表，与一般意义上的数据库表相同。

Catalog、Schema和Table之间的关系如下图所示。



Presto通过各种Connector来接入多种外部数据源。Presto提供了一套标准的SPI接口，用户可以使用这套接口开发自己的Connector，以便访问自定义的数据源。

一个 Catalog一般会绑定一种类型的Connector（在Catalog的Properties文件中设置）。Presto内置了多种Connector。

数据模型

Connector

更多参考

根据集群版本，获取Presto组件的版本号，详情请参见版本概述。

请根据Presto组件的版本号，查看开源Presto文档：

- 当Presto版本是3XX时，修改*http://trino.io/docs/3XX/*中的版本号，在浏览器访问该链接。

  例如，当Presto版本是331时，访问*http://trino.io/docs/331/*，详情请参见Presto 331 Documentation。
- 当Presto版本是0.2XX时，修改*http://prestodb.io/docs/0.2XX/*中的版本号，在浏览器访问该链接。

  例如，当Presto版本是0.228时，访问*http://prestodb.io/docs/0.228/*，详情请参见Presto 0.228 Documentation。

在Zeppelin中使用Presto的详情，请参见Zeppelin概述。

6.2.30.2. 基础使用

6.2.30.2.1. 通过命令行工具访问Presto

本文为您介绍如何通过命令行工具访问Presto控制台。

前提条件

已创建E-MapReduce的Hadoop集群，并且选择了Presto服务。创建详情请参见创建集群。

背景信息

- 如果您创建的Hadoop集群，未打开**Kerberos集群模式**开关，则创建的集群即为普通集群，访问Presto控制台的操作步骤请参见普通集群。
- 如果您创建的Hadoop集群，打开了**Kerberos集群模式**开关，则创建的集群即为高安全集群，访问Presto控制台的操作步骤请参见高安全集群。

普通集群

1. 通过SSH方式登录集群。

   详情请参见登录集群。
2. 执行如下命令，进入Presto控制台。

   ```
   presto --server emr-header-1:9090 --catalog hive --schema default --user hadoop
   ```

   执行 `presto --help` 命令，可以获取Presto控制台的帮助信息。
3. 执行如下命令，查看当前Catalog下的Schema。

   ```
   show schemas;
   ```

   返回如下信息。

```
       Schema
--------------------
 default
 emr_presto_init__
 information_schema
(3 rows)
```

4. （可选）执行 `quit;` ，可以退出Presto控制台。

高安全集群

1. 通过SSH方式登录集群。

   详情请参见登录集群。

2. 添加Principal并导出keytab文件。

   i. 执行如下命令，进入Kerberos的admin工具。

   - EMR-3.30.0及后续版本和EMR-4.5.1及后续版本：

     ```
     sh /usr/lib/has-current/bin/admin-local.sh /etc/ecm/has-conf -k /etc/ecm/has-conf/admin.keytab
     ```

   - EMR-3.30.0之前版本和EMR-4.5.1之前版本：

     ```
     sh /usr/lib/has-current/bin/hadmin-local.sh /etc/ecm/has-conf -k /etc/ecm/has-conf/admin.keytab
     ```

   ii. 执行如下命令，添加指定key的Principal。

   ```
   addprinc -randkey test
   ```

   > **说明** 本文示例为添加test的Principal。

   iii. 执行如下命令，导出keytab文件。

   ```
   xst -k /root/test.keytab test
   ```

   keytab文件默认导出至*/root/*目录下。

3. 执行如下命令，进入Presto控制台。

```
presto --server https://<hostname>:7778 \
       --catalog hive \
       --schema default \
       --keystore-path /etc/ecm/presto-conf/keystore \
       --keystore-password <passwd> \
       --krb5-keytab-path <keytab_file> \
       --krb5-principal <username>@EMR.<cluster_id>.COM \
       --krb5-remote-service-name presto \
       --user <username>
```

参数	描述
hostname	需要您在集群emr-header-1节点上执行 `hostname` 命令获取，格式为*emr-header-1.cluster-xxx*。
passwd	需要您在集群emr-header-1节点上执行 `sed -n 's/http-server.https.keystore.key=\([^;]*\)/\1/p' /etc/ecm/presto-conf/config.properties` 命令获取。
keytab_file	导出的keytab文件的路径。本文示例为*/root/test.keytab*。
username	创建的keytab文件的Principal。本文示例为test。
cluster_id	需要您在集群emr-header-1节点上执行 `hostname | grep -Eo '[0-9]+$'` 命令获取。

示例截图如下。

4. 执行如下命令，查看当前Catalog下的Schema。

```
show schemas;
```

返回如下信息。

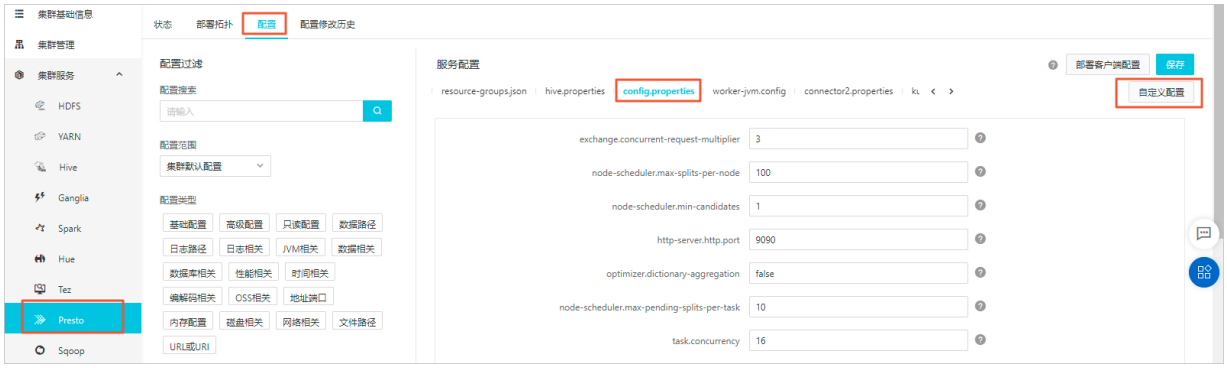
```
       Schema
--------------------
 default
 emr_presto_init__
 information_schema
(3 rows)
```

5. （可选）执行 `quit;` ，可以退出Presto控制台。

常见问题

Q：错误信息“Access Denied: User xxx@EMR.xxx.COM cannot impersonate user xxx”。

A：您需要在EMR控制台Presto服务配置的**config.properties**页面，先添加自定义配置，设置**http-server.authentication.krb5.user-mapping.pattern**的值为**(.+)@EMR\\.[0-9]+\\.COM**，然后重启PrestoMaster。



重启服务截图如下。

6.2.30.2.2. 通过Gateway访问Presto

本文为您介绍如何通过配置HAProxy反向代理，实现通过Gateway节点访问Presto服务。该方法也可以扩展到其他组件，例如Impala。

前提条件

- 已创建普通集群或者高可用集群。

  创建集群详情，请参见创建集群。
- 已创建Gateway集群。

  创建Gateway集群详情，请参见创建Gateway集群。

普通集群

普通集群配置Gateway时，只需要配置HAProxy反向代理，以便于对E-MapReduce（EMR）集群上Master节点的Presto Coodrinator的9090端口实现反向代理。

1. 配置HAProxy。
   i. 通过SSH登录Gateway节点，详情请参见登录集群。
   ii. 执行以下命令，编辑HAProxy的配置文件*haproxy.cfg*。

   ```
   vim /etc/haproxy/haproxy.cfg
   ```

iii. 添加以下内容。

```
#---------------------------------------------------------------------
# Global settings
#---------------------------------------------------------------------
global
......
## 配置代理，将Gateway的9090端口映射到emr-header-1.cluster-xxxx的9090端口。
listen prestojdbc :9090
    mode tcp
    option tcplog
    balance source
    server presto-coodinator-1 emr-header-1.cluster-xxxx:9090
```

修改完成后按*Esc*键，并输入 `:wq` 后按下回车键，保存并退出。

> **说明** 本文中的 `emr-header-1.cluster-xxxx` ，您可以通过 `hostname` 命令获取。

2. 执行以下命令，重启HAProxy服务。

```
service haproxy restart
```

3. 配置以下安全组。

方向	配置规则	说明
公网入	自定义TCP，开放9090端口。	该端口用于HAProxy代理Master节点Coodinator端口。

访问Presto服务示例：

- 命令行使用Presto，示例请参见通过命令行工具访问Presto。
- JDBC访问Presto，示例请参见使用JDBC。

高安全集群

EMR高安全集群中的Presto服务使用Kerberos服务进行认证，其中Kerberos KDC服务位于emr-header-1上，端口为88，支持TCP/UDP协议。使用Gateway访问高安全集群中的Presto服务，需要同时对Presto Coodinator服务端口和Kerberos KDC实现代理。EMR Presto Coodinator集群，默认使用Keystore配置的CN为emr-header-1，只能在内网使用，因此需要重新生成*CN=emr-header-1.cluster-xxx*的Keystore。

- HTTPS认证相关
  i. 通过SSH方式登录集群，详情请参见登录集群。
  ii. 创建服务端*CN=emr-header-1.cluster-xxx*的Keystore。

```
keytool -genkey -dname "CN=emr-header-1.cluster-xxx,OU=Alibaba,O=Alibaba,L=HZ, ST=zhejiang, C=CN" -alias server -keya
lg RSA -keystore keystore -keypass 81ba14ce6084 -storepass 81ba14ce6084 -validity 36500
```

  iii. 导出证书。

```
keytool -export -alias server -file server.cer -keystore keystore -storepass 81ba14ce6084
```

  iv. 制作客户端Keystore。

```
keytool -genkey -dname "CN=myhost,OU=Alibaba,O=Alibaba,L=HZ, ST=zhejiang, C=CN" -alias client -keyalg RSA -keystore c
lient.keystore -keypass 123456 -storepass 123456 -validity 36500
```

  v. 导入证书到客户端Keystore。

```
keytool -import -alias server -keystore client.keystore -file server.cer -storepass 123456
```

  根据提示信息输入 `yes` 并回车，信任此证书。

  vi. 拷贝生成的文件到客户端。

```
scp root@xxx.xxx.xxx.xxx:/etc/ecm/presto-conf/client.keystore ./
```

  > **说明** 本文中的 `xxx.xxx.xxx.xxx` 为主节点的公网IP地址。

  根据提示信息输入 `yes` 并回车。

- Kerberos认证相关
  i. 通过SSH方式登录集群，详情请参见登录集群。
  ii. 添加Principal并导出keytab文件。

a. 执行如下命令，进入Kerberos的admin工具。

- EMR-3.30.0及后续版本和EMR-4.5.1及后续版本：

```
sh /usr/lib/has-current/bin/admin-local.sh /etc/ecm/has-conf -k /etc/ecm/has-conf/admin.keytab
```

- EMR-3.30.0之前版本和EMR-4.5.1之前版本：

```
sh /usr/lib/has-current/bin/hadmin-local.sh /etc/ecm/has-conf -k /etc/ecm/has-conf/admin.keytab
```

b. 执行如下命令，添加指定key的Principal。

```
addprinc -pw 123456 clientuser
```

c. 执行如下命令，导出keytab文件。

```
ktadd -k /root/clientuser.keytab clientuser
```

keytab文件默认导出至*/root/*目录下。

iii. 拷贝生成的文件到客户端。

a. 拷贝文件*clientuser.keytab*到客户端。

```
scp root@xxx.xxx.xxx.xxx:/root/clientuser.keytab ./
```

根据提示信息输入主节点的密码。

b. 拷贝文件*krb5.conf*到客户端。

```
scp root@xxx.xxx.xxx.xxx:/etc/krb5.conf ./
```

根据提示信息输入主节点的密码。

iv. 修改拷贝到客户端的*krb5.conf*文件，修改如下两处。

a. 修改udp_preference_limit为1。因为HAProxy不支持UDP协议，所以修改使客户端使用TCP协议与KDC通信。

b. 修改kdc 的值为Gateway的公网IP地址。

```
[libdefaults]
    kdc_realm = EMR.***.COM
    default_realm = EMR.***.COM
    # 修改参数为1，使客户端使用TCP协议与KDC通信（因为HAProxy不支持UDP协议）。
    udp_preference_limit = 1
    kdc_tcp_port = 88
    kdc_udp_port = 88
    dns_lookup_kdc = false
[realms]
    EMR.xxx.COM = {
        # 设置为Gateway的外网IP。
        kdc = xxx.xxx.xxx.xxx:88
    }
```

v. 修改客户端主机的*hosts*文件，添加以下内容。

```
#  gateway ip
xxx.xxx.xxx.xxx emr-header-1.cluster-xxx
```

- 配置Gateway HAProxy。

i. 通过SSH方式登录Gateway节点，详情请参见登录集群。

ii. 编辑文件*/etc/haproxy/haproxy.cfg*，添加以下内容。

```
#---------------------------------------------------------------------
# Global settings
#---------------------------------------------------------------------
global
......
listen prestojdbc :7778
    mode tcp
    option tcplog
    balance source
    server presto-coodinator-1 emr-header-1.cluster-xxx:7778
listen kdc :88
    mode tcp
    option tcplog
    balance source
    server emr-kdc emr-header-1:88
```

编辑完成后，保存退出。

iii. 执行以下命令，重启HAProxy服务。

```
service haproxy restart
```

iv. 配置以下安全组规则。

方向	配置规则	说明
公网入	自定义UDP，开放88端口。	该端口用于HAProxy代理Master节点上的KDC。
公网入	自定义TCP，开放88端口。	该端口用于HAProxy代理Master节点上的KDC。
公网入	自定义TCP，开放7778端口。	该端口用于HAProxy代理Master节点的Coodinator端口。

- 使用JDBC访问Presto，代码示例如下。

```
try {
    Class.forName("com.facebook.presto.jdbc.PrestoDriver");
} catch(ClassNotFoundException e) {
    LOG.error("Failed to load presto jdbc driver.", e);
    System.exit(-1);
}
Connection connection = null;
Statement statement = null;
try {
    String url = "jdbc:presto://emr-header-1.cluster-5****:7778/hive/default";
    Properties properties = new Properties();
    properties.setProperty("user", "hadoop");
    // https相关配置。
    properties.setProperty("SSL", "true");
    properties.setProperty("SSLTrustStorePath", "resources/5****/client.keystore");
    properties.setProperty("SSLTrustStorePassword", "123456");
    // Kerberos相关配置。
    properties.setProperty("KerberosRemoteServiceName", "presto");
    properties.setProperty("KerberosPrincipal", "clientuser@EMR.5****.COM");
    properties.setProperty("KerberosConfigPath", "resources/5****/krb5.conf");
    properties.setProperty("KerberosKeytabPath", "resources/5****/clientuser.keytab");
    // 创建连接对象。
    connection = DriverManager.getConnection(url, properties);
    // 创建Statement对象。
    statement = connection.createStatement();
    // 执行查询。
    ResultSet rs = statement.executeQuery("select * from table1");
    // 获取结果。
    int columnNum = rs.getMetaData().getColumnCount();
    int rowIndex = 0;
    while (rs.next()) {
        rowIndex++;
        for(int i = 1; i <= columnNum; i++) {
            System.out.println("Row " + rowIndex + ", Column " + i + ": " + rs.getString(i));
        }
    }
} catch(SQLException e) {
    LOG.error("Exception thrown.", e);
} finally {
    // 销毁Statement对象。
    if (statement != null) {
        try {
            statement.close();
            } catch(Throwable t) {
              // No-ops
            }
    }
   // 关闭连接。
   if (connection != null) {
          try {
           connection.close();
       } catch(Throwable t) {
         // No-ops
       }
    }
}
```

6.2.30.2.3. 使用JDBC

本文为您介绍如何使用Presto或Trino提供的JDBC Driver连接数据库。Java应用可以使用Presto或Trino提供的JDBC Driver连接数据库。

在Maven中引入JDBC Driver

您可以根据E-MapReduce集群的版本，在*pom.xml*中添加如下配置引入Presto或Trino JDBC Driver。

EMR版本	组件版本	JDBC Driver	Driver类名

EMR版本	组件版本	JDBC Driver	Driver类名
• EMR-3.x系列：EMR-3.38.0及以上 • EMR-5.x系列：EMR-5.5.0及以上	3XX	<dependency> <groupId>io.trino</groupId> <artifactId>trino-jdbc</artifactId> <version>3XX</version> </dependency>	io.trino.jdbc.TrinoDriver
• EMR-3.x系列：EMR-3.25.0~EMR-3.37.x • EMR-4.x系列：EMR-4.3.0~EMR-4.9.0 • EMR-5.x系列：EMR-5.2.1~EMR-5.4.3	3XX	<dependency> <groupId>io.prestosql</groupId> <artifactId>presto-jdbc</artifactId> <version>3XX</version> </dependency>	io.prestosql.jdbc.PrestoDriver
其他EMR版本	0.2XX	<dependency> <groupId>com.facebook.presto</groupId> <artifactId>presto-jdbc</artifactId> <version>0.2XX</version> </dependency>	com.facebook.presto.jdbc.PrestoDriver

连接数据库

- 当EMR集群为EMR-3.38.0及以上版本或EMR-5.5.0及以上版本时，您可以通过如下JDBC URL，使用JDBC Driver连接数据库。

```
jdbc:trino://<COORDINATOR>:<PORT>/[CATALOG]/[SCHEMA]
```

连接示例如下所示。

```
jdbc:trino://emr-header-1:9090              # 连接数据库，使用Catalog和Schema。
jdbc:trino://emr-header-1:9090/hive         # 连接数据库，使用Catalog（hive）和Schema。
jdbc:trino://emr-header-1:9090/hive/default  # 连接数据库，使用Catalog（hive）和Schema（default）。
```

- 其余版本时，您可以通过如下JDBC URL，使用JDBC Driver连接数据库。

```
jdbc:presto://<COORDINATOR>:<PORT>/[CATALOG]/[SCHEMA]
```

连接示例如下所示。

```
jdbc:presto://emr-header-1:9090              # 连接数据库，使用Catalog和Schema。
jdbc:presto://emr-header-1:9090/hive         # 连接数据库，使用Catalog（hive）和Schema。
jdbc:presto://emr-header-1:9090/hive/default  # 连接数据库，使用Catalog（hive）和Schema（default）。
```

连接参数

JDBC Driver支持很多参数，参数的传入方式示例如下：

- 通过Properties对象传入。
  - EMR集群为EMR-3.38.0及以上版本或EMR-5.5.0及以上版本

```
String url = "jdbc:trino://emr-header-1:9090/hive/default";
Properties properties = new Properties();
properties.setProperty("user", "hadoop");
Connection connection = DriverManager.getConnection(url, properties);
......
```

  - 其余版本

```
String url = "jdbc:presto://emr-header-1:9090/hive/default";
Properties properties = new Properties();
properties.setProperty("user", "hadoop");
Connection connection = DriverManager.getConnection(url, properties);
......
```

- 通过URL传入。
  - EMR集群为EMR-3.38.0及以上版本或EMR-5.5.0及以上版本

    ```
    String url = "jdbc:trino://emr-header-1:9090/hive/default?user=hadoop";
    Connection connection = DriverManager.getConnection(url);
    ......
    ```

  - 其余版本

    ```
    String url = "jdbc:presto://emr-header-1:9090/hive/default?user=hadoop";
    Connection connection = DriverManager.getConnection(url);
    ......
    ```

常用参数说明如下。

参数名称	格式	参数说明
user	STRING	用于身份验证和授权的用户名。
password	STRING	用于LDAP身份验证的密码。
socksProxy	STRING:NUMBER	SOCKS代理服务器地址。例如*localhost:1080*。
httpProxy	STRING:NUMBER	HTTP代理服务器地址。例如*localhost:8888*。
SSL	BOOLEAN	是否使用HTTPS连接。默认为false。
SSLTrustStorePath	STRING	Java TrustStore文件路径。
SSLTrustStorePassword	STRING	Java TrustStore密码。
KerberosRemoteServiceName	STRING	Kerberos服务名称。
KerberosPrincipal	STRING	Kerberos Principal。
KerberosUseCanonicalHostname	BOOLEAN	是否使用规范化的主机名。默认为false。
KerberosConfigPath	STRING	Kerberos配置文件路径。
KerberosKeytabPath	STRING	Kerberos KeyTab文件路径。
KerberosCredentialCachePath	STRING	Kerberos Credential缓存信息。

示例

Java使用JDBC Driver连接数据库示例的片段如下。

```
.....
Connection connection = null;
Statement statement = null;
try {
    // 根据组件名称使用正确的JDBC URL
    String url = "jdbc:<trino/presto>://emr-header-1:9090/hive/default";
    Properties properties = new Properties();
    properties.setProperty("user", "hadoop");
    // 创建连接对象。
    connection = DriverManager.getConnection(url, properties);
    // 创建Statement对象。
    statement = connection.createStatement();
    // 执行查询。
    ResultSet rs = statement.executeQuery("select * from t1");
    // 获取结果。
    int columnNum = rs.getMetaData().getColumnCount();
    int rowIndex = 0;
    while (rs.next()) {
        rowIndex++;
        for(int i = 1; i <= columnNum; i++) {
            System.out.println("Row " + rowIndex + ", Column " + i + ": " + rs.getInt(i));
        }
    }
} catch(SQLException e) {
    LOG.ERROR("Exception thrown.", e);
} finally {
  // 销毁Statement对象。
  if (statement != null) {
      try {
        statement.close();
    } catch(Throwable t) {
        // No-ops
    }
  }
  // 关闭连接。
  if (connection != null) {
      try {
        connection.close();
    } catch(Throwable t) {
        // No-ops
    }
  }
}
```

6.2.30.2.4. 使用独立的Presto集群

创建独立的Presto集群后无法直接使用，需要进行相关的配置。本文为您介绍创建独立的Presto集群后，如何配置连接器和数据湖元数据，以使用独立的Presto集群。

背景信息

在使用开源大数据平台E-MapReduce控制台时，您可以通过在Hadoop集群添加Presto服务或者创建独立的Presto集群使用Presto服务。Presto集群仅包含了SmartData、Presto、Hudi、Iceberg和Hue等必要的服务，具备以下特点：

- Presto独享集群资源，受其他组件干扰少。
- 支持弹性伸缩。
- 支持数据湖分析与实时数仓。
- 不存储数据。

> 说明
> - Hudi和Iceberg不是实际的进程，不占集群资源。
> - Hue和SmartData服务，如果不使用，可以选择停止。

如果要使用独立的Presto集群，需要先创建一个Hadoop集群或使用已有的Hadoop集群作为数据集群。

创建完独立的Presto集群后，您需要进行以下配置：

- 配置连接器
- （可选）配置数据湖元数据

如果数据集群（Hadoop集群）的元数据选择的不是**数据湖元数据**，则可以忽略此配置。

前提条件

已在开源大数据平台E-MapReduce控制台创建Presto集群，详情请参见创建集群。

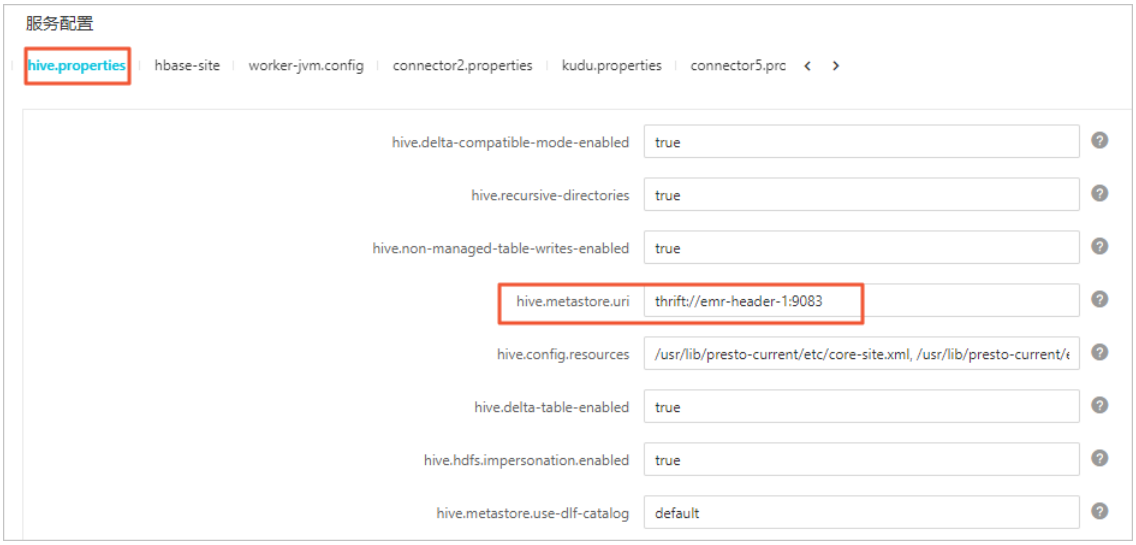
使用限制

- 仅EMR-3.38.0及后续版本支持创建独立的Presto集群。
- Presto集群应与数据集群（Hadoop集群）在同一VPC下。

配置连接器

在待使用的连接器中配置查询对象。本文以Hive连接器为例介绍。

1. 进入独立Presto集群的Presto服务页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏中，选择**集群服务 > Presto**。
2. 修改配置项。
   i. 在Presto服务页面，单击**配置**页签。
   ii. 在**服务配置**区域，单击**hive.properties**页签。
   iii. 找到参数hive.metastore.uri，修改参数值为数据集群（Hadoop集群）的Hive Metastore地址。



> **说明**
> - Hive连接器提供了hive.config.resources配置项，用于加载配置文件。独立的Presto集群没有Hadoop等组件，所以提供了*core-site*、*hdfs-site*和*hbase-site*等配置文件，配置文件默认保存在Presto根路径的*etc*目录下。您可以直接在EMR控制台Presto服务的**配置**页面， 查看和修改配置文件的内容，也可以自定义配置项。
> - Hadoop集群默认会使用集群服务自带的*core-site*、*hdfs-site*和*hbase-site*配置文件，您可以在对应服务的配置中修改，此时Presto服务中提供的这些配置项可以忽略。

3. 保存配置。
   i. 单击右侧的**保存**。
   ii. 在**确认修改**对话框中，配置相关参数，单击**确定**。
4. 生效配置。
   i. 单击右侧的**部署客户端配置**。
   ii. 在**执行集群操作**对话框中，配置相关参数，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。
5. 重启Presto服务，详情请参见重启服务。
6. 配置host。

> 注意 如果待查询的数据全部保存在OSS上，或者建表语句时指定了Location，则可以不用配置host。

Hive的部分表在创建时，会默认指定emr-header-1.cluster开头的路径，因此在查询保存在数据集群上的数据时，为了查询时能够读取到这些表，Presto集群的每一台主机都需要配置host。

- 方式一（推荐）：您可以在EMR控制台，通过添加集群脚本或引导操作，来实现配置host，详情请参见集群脚本或添加引导操作。
- 方式二：直接修改host文件。具体步骤如下所示。
  a. 获取数据集群（Hadoop集群）Master节点的内网IP地址。登录开源大数据平台E-MapReduce控制台，在待操作数据集群的**集群基础信息**页面的**主机信息**区域，可以查看所有实例的内网IP地址。

  b. 登录数据集群（Hadoop集群），登录详情请参见登录集群。
  c. 执行 `hostname` 命令，获取主机名。
  例如，主机名形式为emr-header-1.cluster-26****。
  d. 登录Presto集群，详情请参见登录集群。
  e. 执行以下命令，编辑文件*hosts*。

```
vim /etc/hosts
```

  f. 添加以下内容至文件*hosts*最后一行。
  添加数据集群Master节点的内网IP地址和主机名至Presto集群所有主机*/etc/*目录下的*hosts*文件中。

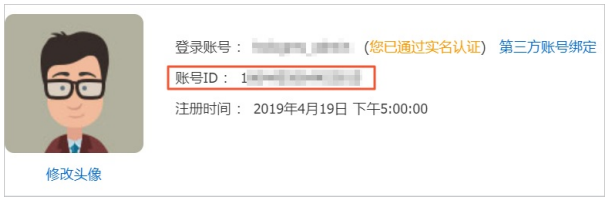
```
192.168.**.** emr-header-1.cluster-26****
```

配置数据湖元数据

如果数据集群（Hadoop集群）的元数据选择为**数据湖元数据**，则还需为Hive、Iceberg和Hudi等连接器进行额外的配置，以访问数据湖。

数据湖元数据配置的详细信息如下表。

参数	描述	备注
hive.metastore	MetaStore类型。	固定值为DLF。
dlf.catalog.region	DLF服务的地域名。	详情请参见已开通的地域和访问域名。 说明 请和dlf.catalog.endpoint选择的地域保持一致。

参数	描述	备注
dlf.catalog.endpoint	DLF服务的Endpoint。	详情请参见已开通的地域和访问域名。 推荐您设置dlf.catalog.endpoint参数为DLF的VPC Endpoint。例如，如果您选择的地域为cn-hangzhou地域，则dlf.catalog.endpoint参数需要配置为dlf-vpc.cn-hangzhou.aliyuncs.com。 说明 您也可以使用DLF的公网Endpoint，如果您选择的地域为cn-hangzhou地域，则dlf.catalog.endpoint参数需要配置为dlf.cn-hangzhou.aliyuncs.com。
dlf.catalog.akMode	DLF服务的Access Key模式。	建议配置为EMR_AUTO。
dlf.catalog.proxyMode	DLF服务的代理模式。	建议配置为DLF_ONLY。
dlf.catalog.uid	阿里云账号的账号ID。	登录账号信息，请通过用户信息页面获取。 

示例：查询表信息

1. 登录Presto集群，详情请参见登录集群。
2. 执行以下命令，启动Presto客户端。

```
presto --server emr-header-1:9090
```

3. 执行以下命令，查询test_hive表信息。

```
select * from hive.default.test_hive;
```

返回如下信息。

```
 id
----
 3
 2
 1
```

6.2.30.2.5. 常用连接器

6.2.30.2.5.1. 配置连接器

本文为您介绍E-MapReduce（简称EMR）的Presto提供的内置连接器，以及如何修改和添加连接器等操作。

背景信息

本文为您介绍连接器相关的内容和操作，具体如下：

- EMR Presto内置连接器
- 修改内置连接器
- 增加自定义连接器
- 测试连通性

前提条件

已创建Hadoop集群，并选择了Presto和相应的服务，详情请参见创建集群。

EMR Presto内置连接器

EMR Presto默认提供以下几种开箱即用的内置连接器。

连接器	功能	对应文档
hive	使用Hive连接器可以查询存储在Hive数据仓库中的数据。	Hive连接器
hive-acc		
kudu	使用Kudu连接器可以查询、插入和删除存储在Kudu里的数据。	Kudu连接器
mysql	使用MySQL连接器可以在外部MySQL实例中查询和创建表。	MySQL连接器
iceberg	使用Iceberg连接器可以查询Iceberg格式的数据文件。	Iceberg连接器
hudi	使用Hudi连接器可以查询COW和MOR表。	Hudi连接器
delta	使用Delta连接器可以查询Delta Lake表。	Delta连接器
jmx	使用JMX连接器可以进行系统监控和调试。	无
system	使用系统连接器可以查询Presto集群的基本信息和度量。 **说明** 系统连接器无需配置，所有信息都可以通过名为 `system` 的目录获取。	无

修改内置连接器

您可以在EMR控制台的Presto服务的**配置**页面，修改各个连接器对应的配置文件。连接器与配置文件对应关系如下表。

连接器	配置文件
hive	hive.properties
hive-acc	hive-acc.properties
kudu	kudu.properties
mysql	mysql.properties
iceberg	iceberg.properties
hudi	hudi.properties
delta	delta.properties
jmx	jmx.properties
system	system.properties

示例：修改Hive连接器

1. 在EMR控制台的Presto服务的**配置**页面，在**服务配置**区域，单击**hive.properties**页签。
2. 根据实际情况修改各配置项。

   修改配置项的详细信息，请参见修改组件参数。
3. 保存配置时，在**确认修改**对话框中开启**自动更新配置**。
4. 保存完配置后，在右上角选择**操作 > 重启All Components**。

   说明 服务重启完成后，即可正常使用连接器。

增加自定义连接器

您可以使用EMR Presto默认提供的3个占位连接器，增加自定义连接器，分别为connector1、connector2和connector3，即EMR控制台的Presto服务的**配置**页面的**connector1.properties**、**connector2.properties**和**connector3.properties**。占位连接器的配置方法与内置连接器一致。

如果你需要增加自定义连接器，修改未被使用的占位连接器配置即可。

示例：增加Mongodb连接器

1. 进入EMR on ECS控制台的Presto服务的**配置**页面，在**服务配置**区域，单击**connector1.properties**页签。

2. 修改connector.name的值为mongodb。
3. 根据实际情况添加自定义配置项。

   添加配置项的详细信息，请参见添加组件参数。
4. 保存配置时，在**确认修改**对话框中开启**自动更新配置**。
5. 保存完配置后，在右上角选择**操作 > 重启All Components**。

   > **说明** 服务重启完成后，即可正常使用连接器。

测试连通性

1. 通过SSH方式连接Presto集群，详情请参见登录集群。
2. 执行以下命令，进入Presto控制台。

   ```
   presto --server emr-header-1:9090 --catalog hive
   ```

   > **说明** hive为您连接器的名称。
3. 执行以下命令，查看Schema。

   ```
   show schemas;
   ```

   当返回信息中，包含FINISHED时，表示连接正常。

6.2.30.2.5.2. MySQL连接器

使用MySQL连接器能够让您在外部MySQL实例中查询和创建表。该连接器可以用于不同系统（例如，MySQL和Hive），或者两个不同MySQL实例间的数据进行JOIN操作。

背景信息

本文为您介绍MySQL连接器相关的内容和操作，具体如下：

- 配置MySQL连接器
- 类型映射
- 示例：查询MySQL
- 下推（Pushdown）

前提条件

- 已创建Hadoop集群，并选择了Presto服务，或者创建单独的Presto集群，详情请参见创建集群。
- 已购买RDS，详情请参见创建RDS MySQL实例。

使用限制

- EMR-3.38.0及后续版本的Hadoop集群或Presto集群，支持配置MySQL连接器。
- 连接的MySQL需要是5.7、8.0版本或更高的版本。
- Presto的Coordinator和所有Worker节点必须能够访问MySQL，默认端口为3306。
- 暂不支持下列SQL命令：
  - DELETE
  - GRANT
  - REVOKE
  - SHOW GRANTS
  - SHOW ROLES
  - SHOW ROLE GRANTS

配置MySQL连接器

修改MySQL连接器配置，详情请参见配置连接器。

进入EMR控制台的Presto服务的**配置**页面，在**服务配置**区域，单击**mysql.properties**页签，您可以看到以下参数，参数值请根据您实际情况修改。

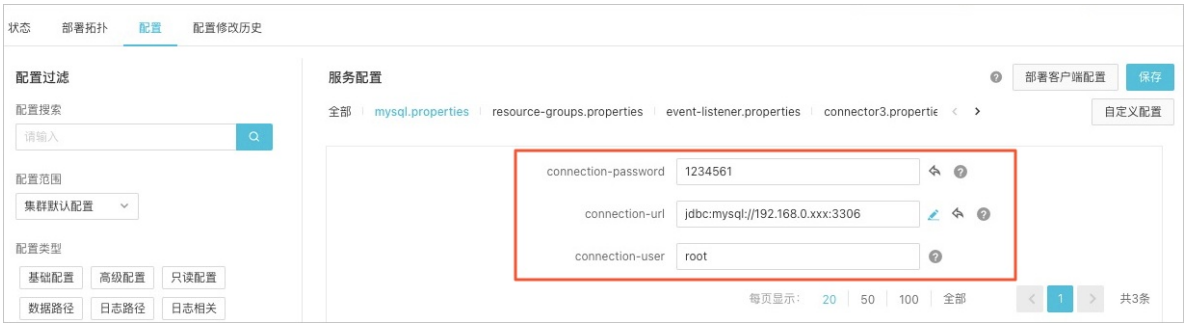
参数	描述

参数	描述
connection-url	数据连接字符串，详情请参见查看或修改内外网地址和端口。 例如，jdbc:mysql://rm-2ze5ipacsu8265q****.mysql.rds.aliyuncs.com:3306。
connection-user	数据库的用户名。该用户具有访问上述RDS MySQL库中表的权限。
connection-password	connection-user对应的密码。

如果您有多个MySQL服务，可以在*etc/catalog*下创建对应数量且不重名的配置文件，确保文件后缀为*.properties*格式即可。例如，创建的配置文件名称为*sales.properties*，Presto将会使用该文件配置的连接器创建一个名为*sales*的Catalog。

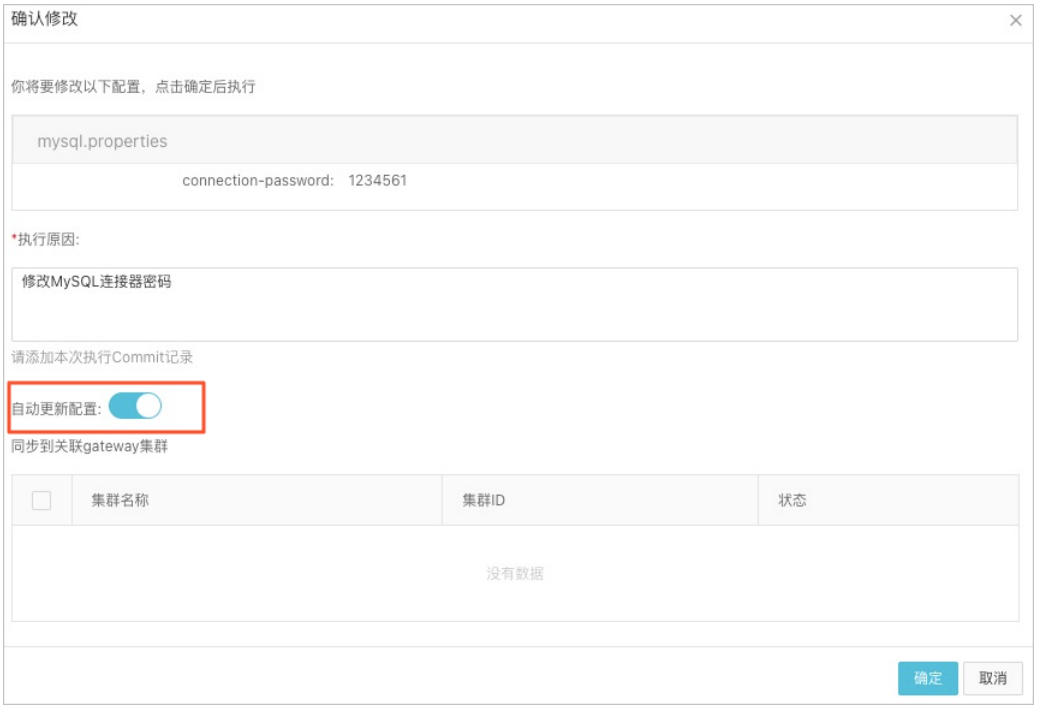
在EMR集群中，如果需要配置多个MySQL服务，可以使用connector[x].properties添加自定义配置项来配置MySQL服务，其中N 为1、2、3……。详细信息请参见以下步骤：

1. 在EMR控制台的Presto服务的配置页面，在**服务配置**区域的**mysql.properties**页签中，修改以下配置。



您也可以单击右上角的**自定义配置**，添加您需要的配置项，详情请参见添加组件参数。

2. 保存配置时，在**确认修改**对话框中开启下**自动更新配置**。



3. 保存完配置后，在右上角选择**操作 > 重启All Components**。

连接器默认配置

配置多个MySQL服务

配置示例

类型映射

精度超过38位的DECIMAL类型可以通过设定配置项decimal_mapping或将decimal_mapping的Session属性设为allow_overflow来映射到Presto的DECIMAL。 结果类型的范围由配置项decimal-default-scale或decimal-rounding-mode控制，其精度恒定为38。

默认情况下，需舍入或截断才能匹配的值在运行时会执行失败。这一行为由配置项decimal-rounding-mode或decimal-rounding-mode的Session属性控制，其值可设为UNNECESSARY（默认值）、UP、DOWN、CEILING、FLOOR、HALF_UP、HALF_DOWN或HALF_EVEN。

通过配置下列属性，可以将来自数据源的数据类型映射到Presto的数据类型，以及可以在Presto中缓存元数据。

属性	描述
unsupported-type-handling	配置如何处理不支持的数据列类型。取值如下： • IGNORE（默认值）：列不可访问。 • CONVERT_TO_VARCHAR：列被转换为无界的VARCHAR。
jdbc-types-mapped-to-varchar	允许将以逗号（,）分隔的列表的数据类型映射强制转换为无界VARCHAR。
case-insensitive-name-matching	数据库和集合名称是否区分大小写。取值如下： • true：不区分大小写。 • false（默认值）：区分大小写。
case-insensitive-name-matching.cache-ttl	不区分大小写的数据库和集合名称的缓存时间。 默认值为1，单位分钟。
metadata.cache-ttl	缓存包括表和列统计信息在内的元数据的持续时间。 默认值0表示禁止缓存。
metadata.cache-missing	是否缓存包括表和列统计信息在内的元数据的状态。取值如下： • true：缓存。 • false（默认值）：不缓存。

DECIMAL类型处理

通用配置属性

示例：查询MySQL

1. 查看RDS的数据库。
   i. 通过SSH方式连接集群，详情请参见登录集群。
   ii. 执行如下命令，连接Presto客户端。

   ```
   presto --server emr-header-1:9090 --catalog iceberg --schema default
   ```

   返回如下信息，表示Presto连接成功。

   ```
   presto:default>
   ```

   iii. 执行如下命令，查看Schema。

   ```
   show schemas from mysql;
   ```

   > **说明** mysql为properties配置文件的名称。

   iv. 执行如下命令，查看数据库。

   ```
   show tables from mysql.web;
   ```

   > **说明** 本文示例中的web是您在MySQL上创建的数据库。

2. 查询表数据。
   - 查询mysql.web.clicks表的数据。

     ```
     select * from mysql.web.clicks
     ```

   - 查询mysql.web.clicks表的列信息。

     ```
     show columns from mysql.web.clicks;
     ```

     或者使用以下命令查询表的列信息。

     ```
     describe mysql.web.clicks;
     ```

下推（Pushdown）

Pushdown详细信息，请参见Presto官网文档Pushdown。

MySQL连接器支持下推的算子和函数情况如下：

- 算子
  - Join
  - Limit
  - Top-N
- 函数
  - avg()
  - count()
  - max()
  - min()
  - sum()
  - stddev()
  - stddev_pop()
  - stddev_samp()
  - variance()
  - var_pop()
  - var_samp()

6.2.30.2.5.3. Kudu连接器

使用Kudu连接器可以查询、插入和删除存储在Kudu里的数据。

背景信息

本文为您介绍Kudu连接器相关的内容和操作，具体如下：

- 修改Kudu连接器配置
- 数据查询
- 数据类型映射
- 支持的Presto SQL语法
- 创建表
- 增加列

前提条件

已创建集群，详情请参见创建集群。

使用限制

- Kudu版本需要为1.10及以上。
- Presto集群和Kudu集群网络互通。
- Kudu表名称和列名称仅支持小写字母。

修改Kudu连接器配置

修改Kudu连接器配置，详情请参见配置连接器。

进入EMR控制台的Presto服务的**配置**页面，在**服务配置**区域，单击**kudu.properties**页签。您可以看到以下参数，参数值请根据您实际情况修改。

参数	描述
kudu.client.master-addresses	Kudu主地址列表，多个地址时使用逗号（,）分隔。 支持以下格式：example.com、example.com:7051、192.0.2.1、192.0.2.1:7051、[2001:db8::1]、[2001:db8::1]:7051和2001:db8::1。 默认值为localhost。

参数	描述
kudu.schema-emulation.enabled	是否开启Schema模拟功能。 取值如下： • false（默认值）：不开启Schema模拟功能。 • true：开启Schema模拟功能。 注意 您可以在kudu.properties页签，单击右上角的自定义配置添加该配置项，添加配置项详情请参见添加组件参数。
kudu.schema-emulation.prefix	Schema模拟功能的前缀。 注意 当 kudu.schema emulation.enabled=true 时，需要设置此参数。 标准前缀为 'presto:` ，也支持空前缀。
kudu.client.default-admin-operation-timeout	管理操作（例如，Create Table、Delete Table等）的默认超时。 默认值为30s。
kudu.client.default-operation-timeout	用户操作的默认超时。 默认值为30s。
kudu.client.default-socket-read-timeout	等待来自Socket的数据时使用的默认超时。 默认值为10s。
kudu.client.disable-statistics	是否启用Kudu客户端的统计信息收集功能。取值如下： • false（默认值）：禁用。 • true：启用。

数据查询

Apache Kudu不支持Schema，但是通过配置Kudu Connector可以支持Schema功能。

Schema模拟默认是关闭的，此时kudu的表都在 `default` 的Schema下。

例如，您可以通过执行 `SELECT * FROM kudu.default.orders` 来查询表 `orders` ，如果Catalog和 Schema分别指定为 `kudu` 和 `default` ，则查询语句可以简化为 `SELECT * FROM orders` 。

Kudu的表名称可以包含任意字符，因此需要用双引号（"）引用表名称。例如，查询表 `special.table!` ，执行语句为 `SELECT * FROM kudu.default."special.table!"` 。

示例如下：

1. 在 `default` 的Schema下创建表 `users` 。

```
CREATE TABLE kudu.default.users (
  user_id int WITH (primary_key = true),
  first_name varchar,
  last_name varchar
) WITH (
  partition_by_hash_columns = ARRAY['user_id'],
  partition_by_hash_buckets = 2
);
```

   说明 创建表时必须指定必要的表信息，例如，主键、列的编码格式或压缩格式、Hash分区或Range分区等。

2. 查看表信息。

```
DESCRIBE kudu.default.users;
```

   返回如下类似信息。

```
   Column   |  Type   |                     Extra                      | Comment
------------+---------+------------------------------------------------+---------
 user_id    | integer | primary_key, encoding=auto, compression=default |
 first_name | varchar | nullable, encoding=auto, compression=default    |
 last_name  | varchar | nullable, encoding=auto, compression=default    |
(3 rows)
```

3. 插入数据。

```
INSERT INTO kudu.default.users VALUES (1, 'Donald', 'Duck'), (2, 'Mickey', 'Mouse');
```

4. 查询数据。

```
SELECT * FROM kudu.default.users;
```

如果在连接器的配置文件*etc/catalog/kudu.properties*里设置了开启Schema模拟功能，则表会根据命名约定被映射到对应的Schema里。

- 如果设置了 `kudu.schema-emulation.enabled=true` 和 `kudu.schema-emulation.prefix=` ，则映射关系如下表。

Kudu表名	Presto表名
`orders`	`kudu.default.orders`
`part1.part2`	`kudu.part1.part2`
`x.y.z`	`kudu.x."y.z"`

> **说明** 由于Kudu不能直接支持Schema，Presto会创建一个特殊表 `$schemas` 用来管理Schema。

- 如果设置了 `kudu.schema-emulation.enabled=true` 和 `kudu.schema-emulation.prefix=presto::` ，则映射关系如下表。

Kudu表名	Presto表名
`orders`	`kudu.default.orders`
`part1.part2`	`kudu.default."part1.part2"`
`x.y.z`	`kudu.default."x.y.z"`
`presto::part1.part2`	`kudu.part1.part2`
`presto::x.y.z`	`kudu.x."y.z"`

> **说明** 因为Kudu不能直接支持Schema，所以Presto会创建一个特殊表 `presto::$schemas` 用来管理Schema。

不开启Schema模拟功能（默认行为）

开启Schema模拟功能

数据类型映射

下表为您介绍Presto数据类型和Kudu数据类型的对应情况。

Presto数据类型	Kudu数据类型	备注
BOOLEAN	BOOL	无
TINYINT	INT8	
SMALLINT	INT16	
INTEGER	INT32	
BIGINT	INT64	
REAL	FLOAT	
DOUBLE	DOUBLE	

Presto数据类型	Kudu数据类型	备注
VARCHAR	STRING	从Presto表创建Kudu表执行 `CREATE TABLE ... AS ...` 时，会损失可选的VARCHAR最大长度。
VARBINARY	BINARY	
TIMESTAMP	UNIXTIME_MICROS	kudu列的μs精度降低到ms精度。
DECIMAL	DECIMAL	仅支持Kudu server的1.7.0及后续版本。
DATE	无	不支持 从Presto表创建Kudu表执行 `CREATE TABLE ... AS ...` 时，列的DATE类型会转为STRING类型。
CHAR	无	不支持
TIME		
JSON		
TIME WITH TIMEZONE		
TIMESTAMP WITH TIME ZONE		
INTERVAL YEAR TO MO NTH		
INTERVAL DAY TO SEC OND		
ARRAY		
MAP		
IPADDRESS		

支持的Presto SQL语法

> **说明** 不支持 `ALTER SCHEMA ... RENAME TO ...` 。

SQL语法	备注
`SELECT`	无
`INSERT INTO ... VALUES`	无
`INSERT INTO ... SELECT ...`	无
`DELETE`	无
`DROP SCHEMA`	仅在Schema启用时可用。
`CREATE SCHEMA`	仅在Schema启用时可用。
`CREATE TABLE`	创建表，详情请参见创建表。
`CREATE TABLE ... AS`	无
`DROP TABLE`	无
`ALTER TABLE ... RENAME TO ...`	无
`ALTER TABLE ... ADD COLUMN ...`	增加列，详情请参见增加列。
`ALTER TABLE ... RENAME COLUMN ...`	仅对非主键可用。
`ALTER TABLE ... DROP COLUMN ...`	
`SHOW SCHEMAS`	无
`SHOW TABLES`	无

SQL语法	备注
`SHOW CREATE TABLE`	无
`SHOW COLUMNS FROM`	无
`DESCRIBE`	作用同 `SHOW COLUMNS FROM` 。
`CALL kudu.system.add_range_partition`	增加Range分区，详情请参见Range分区。
`CALL kudu.system.drop_range_partition`	删除Range分区，详情请参见Range分区。

创建表

创建表需要指定列、数据类型和分区信息，也可以根据您实际的情况指定列编码格式或压缩格式。创建表示例如下。

```
CREATE TABLE user_events (
  user_id int WITH (primary_key = true),
  event_name varchar WITH (primary_key = true),
  message varchar,
  details varchar WITH (nullable = true, encoding = 'plain')
) WITH (
  partition_by_hash_columns = ARRAY['user_id'],
  partition_by_hash_buckets = 5,
  number_of_replicas = 3
);
```

该示例中主键为 `user_id` 和 `event_name` ，通过 `user_id` 列的HASH值划分为5个分区， `number_of_replicas` 设置为3。

创建表时相关参数描述如下：

- 主键列必须放在前面，分区列必须选取自主键列。
- `number_of_replicas` ：可选项，该值指定了tablet的副本数，且必须为奇数。如果没有指定该配置，则使用Kudu Master默认配置的副本数。
- Kudu支持Hash和Range两种类型的分区。Hash分区根据Hash值分发数据行到多个桶中的一个桶。Range分区使用有序的Range分区键分发数据行，具体的Range分区必须被显式创建。Kudu支持多级分区，一个表至少要有一个Hash或Range分区，但最多有一个Range分区，可以有多个Hash分区。

除了指定列名称和类型，还可以指定其他列属性。

列属性名	类型	描述
primary_key	BOOLEAN	设置为true，则表示使用该列作为主键。 Kudu主键需要满足唯一性约束。当待插入数据行的主键已经存在，再插入与已有相同主键值的行，则会导致更新已有的数据行，详情请参见Primary Key Design。
nullable	BOOLEAN	设置为true，则表示该列可以取null。 注意 主键列不可为null。
encoding	VARCHAR	指定列编码格式以节省存储空间和提高查询性能。 如果没有指定该属性，则Kudu根据列数据类型自动编码。取值为auto、plain、bitshuffle、runlength、prefix、dictionary和group_varint，详情信息请参见Column Encoding。
compression	VARCHAR	指定列压缩格式。 如果没有指定该属性，Kudu会使用默认压缩格式。取值为default、no、lz4、snappy和zlib，详情信息请参见Column compression。

示例如下。

```
CREATE TABLE mytable (
  name varchar WITH (primary_key = true, encoding = 'dictionary', compression = 'snappy'),
  index bigint WITH (nullable = true, encoding = 'runlength', compression = 'lz4'),
  comment varchar WITH (nullable = true, encoding = 'plain', compression = 'default'),
   ...
) WITH (...);
```

一个表至少要有一个Hash或Range分区，但最多只能有一个Range分区，可以有多个Hash分区。分区信息如下：

- Hash分区
  - 定义一组分区

    您可以使用表属性**partition_by_hash_columns**配置分区列，表属性 `partition_by_hash_buckets` 配置分区个数，所有的分区列必须是主键列的子集。示例如下。

    ```
    CREATE TABLE mytable (
      col1 varchar WITH (primary_key=true),
      col2 varchar WITH (primary_key=true),
      ...
    ) WITH (
      partition_by_hash_columns = ARRAY['col1', 'col2'],
      partition_by_hash_buckets = 4
    )
    ```

    > **说明** 该示例定义了（col1，col2）的Hash分区，数据分布到4个分区。

  - 定义两组分区

    如果需要定义两个独立的Hash分区组，可以再设置表属性 `partition_by_second_hash_columns` 和 `partition_by_second_hash_buckets` 。示例如下。

    ```
    CREATE TABLE mytable (
      col1 varchar WITH (primary_key=true),
      col2 varchar WITH (primary_key=true),
      ...
    ) WITH (
      partition_by_hash_columns = ARRAY['col1'],
      partition_by_hash_buckets = 2,
      partition_by_second_hash_columns = ARRAY['col2'],
      partition_by_second_hash_buckets = 3
    )
    ```

    > **说明** 该示例定义了两组Hash分区，第一组Hash分区按照列col1对数据行分布到2个分区，第二组Hash分区按照列col2对数据行分布到3个分区，因此该表会有共计2*3=6个分区。

- Range分区

  kudu表最多可以有一个Range分区，可以使用表属性 `partition_by_range_columns` 定义。在创建表的时候，可以使用表属性 `range_partitions` 定义分区范围，表属性 `kudu.system.add_range_partition` 和 `kudu.system.drop_range_partition` 可以对已有的表进行Range分区管理。示例如下。

  ```
  CREATE TABLE events (
    rack varchar WITH (primary_key=true),
    machine varchar WITH (primary_key=true),
    event_time timestamp WITH (primary_key=true),
    ...
  ) WITH (
    partition_by_hash_columns = ARRAY['rack'],
    partition_by_hash_buckets = 2,
    partition_by_second_hash_columns = ARRAY['machine'],
    partition_by_second_hash_buckets = 3,
    partition_by_range_columns = ARRAY['event_time'],
    range_partitions = '[{"lower": null, "upper": "2018-01-01T00:00:00"},
                         {"lower": "2018-01-01T00:00:00", "upper": null}]'
  )
  ```

  > **说明** 该示例定义了二组Hash分区和一个对 `event_time` 列进行Range分区的表，Range分区范围由 `2018-01-01T00:00:00` 进行分割。

- 管理Range分区

  对于已经存在的表，有2个存储过程可以增加和删除Range分区。

  分区操作示例如下：

  - 增加一个Range分区

    ```
    CALL kudu.system.add_range_partition(<your_schema_name>, <your_table_name>, <range_partition_as_json_string>)
    ```

- 删除一个Range分区

```
CALL kudu.system.drop_range_partition(<your_schema_name>, <your_table_name>, <range_partition_as_json_string>)
```

参数	描述
`<your_schema_name>`	表所在的Schema。
`<your_table_name>`	表名称。
`<range_partition_as_json_string>`	JSON格式表示的Range分区的上下边界，格式为 `'{"lower": <value>, "upper": <value>}'` ，如果分区有多个列，则格式为 `'{"lower": [<value_col1>,...], "upper": [<value_col1>,...]}'` ，上下边界具体的取值形式由列数据类型决定。数据类型和JSON字符串类型对应关系如下： - BIGINT：`'{"lower": 0, "upper": 1000000}'` - SMALLINT：`'{"lower": 10, "upper": null}'` - VARCHAR：`'{"lower": "A", "upper": "M"}'` - TIMESTAMP：`'{"lower": "2018-02-01T00:00:00.000", "upper": "2018-02-01T12:00:00.000"}'` - BOOLEAN：`'{"lower": false, "upper": true}'` - VARBINARY：Base64编码的字符串值 **说明** 设置为null时，则表示无限边界。

示例如下。

```
CALL kudu.system.add_range_partition('myschema', 'events', '{"lower": "2018-01-01", "upper": "2018-06-01"}')
```

说明 该示例 `myschema` 里的表 `events` 增加了一个Range分区，该分区的下界是 `2018-01-01` ，即精确值是 `2018-01-01T00:00:00.000` ，分区的上界是 `2018-06-01` 。

您可以使用SQL语句 `SHOW CREATE TABLE` 查询已经存在的Range分区，分区信息通过表属性 `range_partitions` 展示。

列属性

表属性

分区设计

增加列

您可以使用SQL语句 `ALTER TABLE ... ADD COLUMN ...` 为已经存在的表增加数据列，还可以使用列属性来增加数据列。列属性的详细信息，可以在创建表模块中查看。

```
ALTER TABLE mytable ADD COLUMN extraInfo varchar WITH (nullable = true, encoding = 'plain')
```

6.2.30.2.5.4. Hive连接器

使用Hive连接器可以查询和分析存储在Hive数据仓库中的数据。

背景信息

Hive数仓系统由以下三部分内容组成：

- 不同格式的数据文件，通常存储在Hadoop分布式文件系统（HDFS）或对象存储系统（例如，阿里云OSS）中。
- 存储着数据文件到Schema和Table映射的元数据。该元数据存储在数据库（例如，MySQL）中，并通过Hive Metastore Service（HMS）访问。
- 一种称为HiveQL的查询语言。该查询语言在分布式计算框架（例如，MapReduce或Tez）上执行。

本文为您介绍Hive连接器相关的内容和操作，具体如下：

- 修改Hive连接器配置
- 支持的文件类型
- 支持的表类型
- Hive视图
- 配置属性

前提条件

已创建集群，详情请参见创建集群。

使用限制

- 配置Hive连接器需要配置Hive Metastore Service。
- Hive连接器支持多种分布式存储系统，包括HDFS、阿里云OSS或OSS的兼容系统，都可以使用Hive连接器查询。Coordinator节点和所有worker节点必须能够通过网络访问Hive Metastore以及存储系统。通过Thrift协议访问Hive Metastore的默认端口是9083。

修改Hive连接器配置

修改Hive连接器配置，详情请参见配置连接器。

进入EMR控制台的Presto服务的**配置**页面，在**服务配置**区域，单击**hive.properties**页签。您可以看到以下参数，参数值请根据您实际情况修改。

参数	描述
hive.recursive-directories	允许从表或分区所在位置的子目录读取数据，类似Hive的 `hive.mapred.supports.subdirectories` 属性。 默认值为false。
hive.metastore.uri	Hive Metastore使用Thrift协议连接的URI。 默认值格式 `thrift://emr-header-1.cluster-24****:9083` 。
hive.config.resources	HDFS配置文件的列表，多个配置文件时以逗号（,）分隔。这些配置文件必须存在于Presto运行的所有主机上。 注意 仅在必须访问HDFS的情况下配置此项。 默认值为 `/etc/ecm/hadoop-conf/core-site.xml, /etc/ecm/hadoop-conf/hdfs-site.xml` 。
hive.delta-table-enabled	是否支持Presto读取Delta Lake表。取值如下： • true（默认值）：Presto可以读取Delta Lake表。 • false：Presto不可以读取Delta Lake表。
hive.delta-compatible-mode-enabled	Delta Lake表是否启用兼容模式。取值如下： • true（默认值）：Delta Lake表启用兼容模式。 • false：Delta Lake表不启用兼容模式。
hive.hdfs.impersonation.enabled	是否启用用户代理。取值如下： • true（默认值）：启用用户代理。 • false：不启用用户代理。
hive.metastore.use-dlf-catalog	是否将Datalake Formation Catalog用作Hive Metastore。 默认值为default。

如果您有多个Hive集群，可以在*etc/catalog*路径下增加相应数量的Catalog文件，确保文件后缀名为*.properties*。

例如，如果属性文件名为*sales.properties*，则Presto将使用其中配置的连接器创建一个叫*sales*的Catalog。

通常情况下，Presto会自动配置HDFS客户端，不需要任何配置文件。在某些情况下，例如启用联合HDFS或NameNode高可用时，需要额外指定HDFS客户端选项才能访问HDFS集群，此时需要添加 `hive.config.resources` 属性以引用所需的HDFS配置文件。

注意
- 仅在需要配置时，才需要额外指定配置文件。建议减少配置文件以包含所需的最少属性集，防止属性间不兼容。
- 配置文件必须存在于Presto运行的所有主机上。如果要引用现有的Hadoop配置文件，请确保将其拷贝到任何未运行Hadoop的Presto节点上。

在Presto中为Hive表运行任何 `CREATE TABLE` 或 `CREATE TABLE AS` 语句之前，都需要检查Presto用于访问HDFS的用户是否有权访问Hive的仓库目录。 Hive仓库目录由*hive-site.xml*中的配置变量 `hive.metastore.warehouse.dir` 指定，默认值为 `/user/hive/warehouse` 。

连接器默认配置

多个Hive集群配置

HDFS配置

HDFS用户名与权限

支持的文件类型

Hive连接器支持下列文件类型。

文件类型	备注
ORC	无
Parquet	无
Avro	无
RCText	使用 `ColumnarSerDe` 的RCFile。
RCBinary	使用 `LazyBinaryColumnarSerDe` 的RCFile。
SequenceFile	无
JSON	使用 `org.apache.hive.hcatalog.data.JsonSerDe` 。
CSV	使用 `org.apache.hadoop.hive.serde2.OpenCSVSerde` 。
TextFile	无

支持的表类型

Hive连接器支持下列表类型。

表类型	描述
ACID表	在连接3.x版本的Hive Metastore时，Hive连接器支持insert-only和ACID表的读写，且完全支持分区和分桶。 对ACID表支持行级的DELETE与UPDATE，不支持分区键列和桶列的UPDATE，不支持使用Hive Streaming Ingest创建的ACID表，详情Streaming Data Ingest。
物化视图	Hive连接器支持从Hive的物化视图中读取数据。在Presto中，这些视图将以常规的、只读表的形式展示。

Hive视图

Hive视图由HiveQL定义，存储在Hive Metastore Service中。

Hive连接器包含以下三种不同模式的Hive视图。

模式	描述
Disabled	视图中编码的业务逻辑和数据在Presto中是不可见的。 默认行为是忽略Hive视图。
Legacy	Hive视图的简单实现，可以读取Presto中的数据。 可以通过配置 `hive.translate-hive-views=true` 和 `hive.legacy-hive-view-translation=true` 启用此模式。 如果想为特定的Catalog临时启用此传统访问方式，可以将Catalog Session属性 `legacy_hive_view_translation` 设置为true。 由于HiveQL与SQL非常相似，因此Legacy方式可以通过SQL语言解释任何定义了视图的HiveQL查询，不做任何转换。 此方式适用于简单的Hive视图，但可能会导致复杂的查询出现问题。例如，如果HiveQL函数具有与SQL相同的签名，但具有不同行为，则返回的结果可能会有所不同。在更极端的情况下，查询可能会失败，甚至无法解析和执行。

模式	描述
Experimental	可以分析、处理与重写Hive视图，包括其包含的表达式和语句。 可以通过配置 `hive.translate-hive-views=true` 启用此模式。 使用此模式时，暂不支持以下功能： • HiveQL的 `current_date` 、 `current_timestamp` ，及其它若干类似的语句。 • `translate()` 、窗口函数以及其他若干类似的Hive函数调用。 • 公用表表达式和简单的Case表达式。 • 设置时间戳精度。 • 将全部Hive数据类型正确映射到Presto类型。 • 处理自定义UDF的能力。

配置属性

Hive连接器支持使用Jindo Table加速。EMR集群中内置了两个Hive连接器，分别为 `hive.properties` 和 `hive-acc.properties` 。 `hive-acc.properties` 内置了JindoTable Native Engine，对ORC或Parquet格式的文件进行加速优化，请根据您SmartData的版本查看相应的文档，详情请参见开启native查询加速。

下表列出了Hive连接器的各项配置属性。

属性名	描述
hive.config.resources	HDFS配置文件的列表，多个文件时以逗号（,）分隔。这些文件必须存在于Presto运行的所有主机上。例如，*/etc/hdfs-site.xml*。 **说明** 仅在必须访问HDFS的情况下配置该属性。
hive.recursive-directories	允许从表或分区所在位置的子目录读取数据，类似Hive的 `hive.mapred.supports.subdirectories` 属性。 默认值为false。
hive.ignore-absent-partitions	当文件系统位置不存在时，忽略该分区而不是报查询失败，但也有可能会跳过原本可能属于表的一部分数据。 默认值为false。
hive.storage-format	建表时的默认文件格式。 默认值为ORC。
hive.compression-codec	写文件时使用的文件编码方式。取值可以为NONE、SNAPPY、LZ4、ZSTD或GZIP。 默认值为GZIP。
hive.force-local-scheduling	强制将分片规划到与处理该分片数据的Hadoop DataNode服务相同的节点上。此配置方式对于Presto与每个DataNode并置的安装很有用，可以提升并置安装的效率。 默认值为false。
hive.respect-table-format	新分区应使用现有的表格式还是Presto的格式。取值如下： • true（默认值）：使用现有的表格式。 • false：使用Presto的格式。
hive.immutable-partitions	新数据能否插入到现存的分区中。 设置为true时， `hive.insert-existing-partitions-behavior` 将不允许设置为 `APPEND` 。 默认值为false。
hive.insert-existing-partitions-behavior	数据插入现有分区时的行为。取值如下： • APPEND（默认值）：在现有分区追加数据。 • OVERWRITE：覆盖现有分区。 • ERROR：不允许修改现有分区。

属性名	描述
hive.create-empty-bucket-files	是否应为没有数据存储的桶创建空文件。取值如下： • true：创建空文件。 • false（默认值）：不创建空文件。
hive.max-partitions-per-writers	每个writer的最大分区数。 默认值为100。
hive.max-partitions-per-scan	一次表扫描的最大分区数。 默认值为100,000。
hive.hdfs.authentication.type	HDFS身份验证类型。取值如下： • NONE（默认值）：表示普通模式，不进行Kerberos认证。 • KERBEROS：表示使用安全模式，进行Kerberos认证。
hive.hdfs.impersonation.enabled	是否启用HDFS端用户模拟。取值如下： • true：启用HDFS端用户模拟。 • false（默认值）：不启用HDFS端用户模拟。
hive.hdfs.trino.principal	Presto连接HDFS时使用的Kerberos主体。
hive.hdfs.trino.keytab	HDFS客户端密钥文件的位置。
hive.dfs.replication	HDFS副本因子。
hive.security	默认值为legacy。详情请参见Hive connector security configuration。
security.config-file	当设置 `hive.security=file` 时使用的配置文件路径。
hive.non-managed-table-writes-enabled	启用对非托管（外部）Hive表的写入。 默认值为false。
hive.non-managed-table-creates-enabled	启用对非托管（外部）Hive表的创建。 默认值为true。
hive.collect-column-statistics-on-write	启用在写入时以列为单位自动收集统计信息。详情请参见配置属性。 默认值为true。
hive.file-status-cache-tables	特定表的缓存路径列表。 例如，`fruit.apple,fruit.orange` 表示仅缓存Schema fruit中的apple和orange表。`fruit.*,vegetable.*` 表示缓存Schema fruit和vegetable中的所有表。`*` 表示缓存所有Schema中的所有表。
hive.file-status-cache-size	缓存文件状态条目的最大总数。 默认值为1000000。
hive.file-status-cache-expire-time	缓存路径列表的有效时间。 默认值为1m。
hive.rcfile.time-zone	将时间戳的二进制编码值调整到指定时区。 默认值为JVM default。 **说明** Hive 3.1及后续版本，需要将此值设为UTC。

属性名	描述
hive.timestamp-precision	指定Timestamp类型Hive列的精度。取值如下： • MILLISECONDS：毫秒。 • MICROSECONDS：微秒。 • NANOSECONDS：纳秒。 默认值为MILLISECONDS。 **说明** 精度高于配置的值时将会四舍五入。
hive.temporary-staging-directory-enabled	控制是否将在 `hive.temporary-staging-directory-path` 中配置的临时模拟目录用于写入操作。 临时模拟目录不会用于写入OSS、加密HDFS或外部位置的无序表。写入排序表时会在排序操作期间使用此路径暂存临时文件。设置为禁用时，目标存储将用于在写入排序表时进行暂存，此方式在写入对象存储时效率低下。 默认值为true。
hive.temporary-staging-directory-path	控制用于写操作的临时暂存目录的位置。 默认值为 `/tmp/presto-${USER}` 。 **说明** *${USER}*占位符可用于让每个用户使用不同的位置。
hive.translate-hive-views	启用Hive视图的变换操作。 默认值为false。
hive.legacy-hive-view-translation	使用传统算法转换Hive视图。可以将 `legacy_hive_view_translation catalog session` 属性用于特定的Catalog。 默认值为false。
hive.parallel-partitioned-bucketed-writes	提高分区表和分桶表写入的并行性。 默认值为true。 **说明** 禁用时，写入线程的数量将仅限于存储桶的数量。

以下属性用于配置由Hive连接器执行ORC文件的读写操作。

属性名	描述
hive.orc.time-zone	为未声明时区的旧版ORC文件设置默认时区。 默认值为JVM default。
hive.orc.use-columns-names	按名称访问ORC列。 默认情况下，ORC文件中的列按它们在Hive表定义中的顺序位置进行访问。 `catalog session` 属性和 `orc_use_column_names` 功能一样。 默认值为false。

以下属性用于配置由Hive连接器执行Parquet文件的读写操作。

属性名	描述
hive.parquet.time-zone	将Timestamp的值调整到指定时区。 默认值为JVM default。 **说明** Hive 3.1及后续版本，需要将此值设为UTC。

属性名	描述
hive.parquet.use-columns-names	访问Parquet列的方式。取值如下： • true（默认值）：按名称访问Parquet列，无需保持列名顺序与文件一致。 • false：按列在Hive表定义中的顺序位置访问Parquet列。 `catalog session` 属性和 `parquet_use_column_names` 功能一样。

Hive metastore可以使用下列属性进行配置，使用特定的属性可以进一步配置Thrift，详情请参见Thrift Metastore配置属性。

属性名	描述
hive.metastore	Hive Metastore使用的类型。Presto支持默认的Hive Thrift metastore（thrift）及其衍生产品。 默认值为thrift。
hive.metastore-cache-ttl	Hive Metastore缓存的Metastore数据被视为可用的持续时间。 默认值为0s。
hive.metastore-cache-maximum-size	Hive Metastore缓存的Metastore数据对象的最大个数。 默认值为10000。
hive.metastore-refresh-interval	访问后异步刷新缓存的Metastore数据。如果缓存的数据是尚未过期的旧数据，则允许后续访问查看新数据。
hive.metastore-refresh-max-threads	刷新Metastore数据缓存的最大线程数。 默认值为10。
hive.metastore-timeout	Hive Metastore请求的超时时间。 默认值为10s。

下表介绍了Hive连接器的Thrift Metastore配置属性。

属性名	描述
hive.metastore.uri	Hive Metastore使用Thrift协议连接的URI。 如果提供了多个URI，则会默认使用第一个，其余的当作备用Metastore。此属性是必填项。示例：`thrift://192.0.**.**:9083` 或 `thrift://192.0.**.**:9083,thrift://192.0.**.**:9083` 。
hive.metastore.username	Presto用于访问Hive Metastore的用户名。
hive.metastore.authentication.type	Hive Metastore身份验证的方式类型。取值如下： • NONE（默认值）：表示普通模式，不进行Kerberos认证。 • KERBEROS：表示使用安全模式，进行Kerberos认证。
hive.metastore.thrift.impersonation.enabled	启用Hive Metastore端用户模拟。
hive.metastore.thrift.delegation-token.cache-ttl	Metastore Delegation Token缓存的有效期限。 默认值为1h。
hive.metastore.thrift.delegation-token.cache-maximum-size	Delegation Token缓存的最大值。 默认值为1000。
hive.metastore.thrift.client.ssl.enabled	连接到Metastore时是否启用SSL。取值如下： • true：连接到Metastore时使用SSL。 • false（默认值）：连接到Metastore时不启用SSL。
hive.metastore.thrift.client.ssl.key	私钥和客户端证书（keyStore）的路径。
hive.metastore.thrift.client.ssl.key-password	私钥的密码。

属性名	描述
hive.metastore.thrift.client.ssl.trust-certificate	服务器证书链（trustStore）的路径。 说明 启用SSL时必填。
hive.metastore.thrift.client.ssl.trust-certificate-password	服务器证书链的密码。
hive.metastore.service.principal	Hive Metastore服务的Kerberos规则。
hive.metastore.client.principal	Presto连接Hive Metastore服务时使用的Kerberos规则。
hive.metastore.client.keytab	Hive Metastore客户端keytab文件的位置。

下表介绍了Hive连接器的性能调优配置属性。

注意 更改下表属性的默认值可能会导致不稳定和性能下降，请谨慎操作。

属性名	描述
hive.max-outstanding-splits	在Scheduler尝试暂停之前，一次查询中每个表扫描的缓存split的目标数量。 默认值为1000。
hive.max-splits-per-second	每次表扫描每秒生成的最大split数，可用于减少存储系统的负载。默认情况下没有限制，即Presto将最大化数据访问的并行度。
hive.max-initial-splits	对于每次表扫描，coordinator首先分配大小不超过max-initial-split-size的文件片段。在分配了max-initial-splits个片段之后，剩余split的最大值由max-split-size决定。 默认值为200。
hive.max-initial-split-size	在已分配的片段数不超过max-initial-splits时，分配给worker节点的单个文件片段的大小。较小的split会导致更高的并行度，从而加速小查询。 默认值为32 MB。
hive.max-split-size	分配给worker节点的单个文件片段的最大值。较小的split会导致更高的并行度，从而可以减少延迟，但也会产生更大的开销并增加系统负载。 默认值为64 MB。

Hive连接器支持收集和管理表统计数据以改进查询过程的性能。

写数据时，Hive连接器默认会收集基础信息，例如，文件数、行数、原始数据大小和总大小，及下表的列级统计数据。

列类型	可收集信息
TINYINT	空值数量、不同的值数量、最大值或最小值
SMALLINT	空值数量、不同的值数量、最大值或最小值
INTEGER	空值数量、不同的值数量、最大值或最小值
BIGINT	空值数量、不同的值数量、最大值或最小值
DOUBLE	空值数量、不同的值数量、最大值或最小值
REAL	空值数量、不同的值数量、最大值或最小值
DECIMAL	空值数量、不同的值数量、最大值或最小值
DATE	空值数量、不同的值数量、最大值或最小值
TIMESTAMP	空值数量、不同的值数量、最大值或最小值
VARCHAR	空值数量、不同的值数量
CHAR	空值数量、不同的值数量

列类型	可收集信息
VARBINARY	空值数量
BOOLEAN	空值数量、true或false值数量

Hive配置属性

ORC格式配置属性

Parquet格式配置属性

Metastore配置属性

Thrift Metastore配置属性

性能调优配置属性

表统计数据

6.2.30.2.5.5. Iceberg连接器

Iceberg是一种开放的数据湖表格式，使用Iceberg连接器可以用来查询Iceberg格式的数据文件。

背景信息

Iceberg的详细信息，请参见Iceberg概述。

阿里云EMR Iceberg连接器兼容社区Iceberg连接器的功能，并在此基础上进行了深度优化，新增以下功能：

- 支持Dynamic Filter。
- 支持Iceberg V2读取。
- 支持Iceberg Filter下推。

本文为您介绍Iceberg连接器相关的内容和操作，具体如下：

- 配置Iceberg连接器
- 示例：查询Iceberg表数据
- SQL语法
- 分区表
- 按分区删除
- 回滚
- 系统表和列
- Iceberg表属性
- 物化视图

前提条件

已创建Hadoop集群，并选择了Presto服务，或者创建单独的Presto集群，详情请参见创建集群。

使用限制

EMR-3.38.0及后续版本的Hadoop集群或Presto集群，支持配置Iceberg连接器。

配置Iceberg连接器

修改Iceberg连接器配置，详情请参见配置连接器。

您可以进入EMR控制台的Presto服务的**配置**页面，在**服务配置**区域，单击**iceberg.properties**页签。您可以看到参数hive.metastore.uri，该参数表示Hive Metastore使用Thrift协议连接的URI。参数值请根据您实际情况修改。默认格式为*thrift://emr-header-1.cluster-24****:9083*。

您可以进入EMR控制台的Presto服务的**配置**页面，在**服务配置**区域，单击**iceberg.properties**页签，单击右侧的**自定义配置**。您可以新增以下配置。

参数	描述
iceberg.file-format	Iceberg表的数据存储文件格式。支持以下格式： • ORC（默认值） • PARQUET

参数	描述
iceberg.compression-codec	写入文件时使用的压缩格式。支持以下格式： • GZIP（默认值） • ZSTD • LZ4 • SNAPPY • NONE
iceberg.max-partitions-per-writer	每个writer最多可处理的分区数。默认值为100。

连接器默认配置

Iceberg配置列表

示例：查询Iceberg表数据

使用Presto的基本语法即可查询Iceberg表。

1. 通过SSH方式连接集群，详情请参见登录集群。
2. 执行以下命令，连接Presto客户端。

   ```
   presto --server emr-header-1:9090 --catalog iceberg --schema default
   ```

   返回如下信息，表示Presto连接成功。

   ```
   presto:default>
   ```

3. 执行以下命令，创建表iceberg_test。

   ```
   create table iceberg_test(id int);
   ```

4. 执行以下命令，向表iceberg_test中插入数据。

   ```
   insert into iceberg_test values(1),(2);
   ```

5. 执行以下命令，查询表数据。

   ```
   select * from iceberg_test;
   ```

   返回如下信息。

   ```
    id
   ----
    1
    2
   ```

SQL语法

Iceberg连接器支持读写Iceberg表数据和元信息，除了支持基础的SQL语法，还支持下表语法。

SQL语法	描述
INSERT	Presto官网文档，请参见INSERT。
DELETE	可以参见本文的按分区删除。 Presto官网文档，请参见DELETE。
Schema and table management	可以参见本文的分区表。 Presto官网文档，请参见Schema and table management。
Materialized views management	可以参见本文的物化视图。 Presto官网文档，请参见Materialized views management。
Views management	Presto官网文档，请参见Views management。

分区表

Iceberg可以基于如下函数对表进行分区。

函数	描述
year(ts)	按年创建分区，分区值是从ts到1970年1月1日之间的年份差。
month(ts)	按月创建分区，分区值是从ts到1970年1月1日之间的月份差。
day(ts)	按天创建分区，分区值是从ts到1970年1月1日之间的天数差。
hour(ts)	按小时创建分区，分区值是ts忽略分钟和秒的时间戳值。
bucket(x, nbuckets)	数据被Hash到指定数量的桶，分区值是x的整数Hash值，范围是[0, nbuckets - 1)。
truncate(s, nchars)	分区值是s的前nchars个字符。

例如，customer_orders表按order_date的月份值、account_number的哈希值 （桶数量为10）和country进行分区。

```
CREATE TABLE iceberg.testdb.customer_orders (
    order_id BIGINT,
    order_date DATE,
    account_number BIGINT,
    customer VARCHAR,
    country VARCHAR)
WITH (partitioning = ARRAY['month(order_date)', 'bucket(account_number, 10)', 'country'])
```

按分区删除

对于分区表，如果WHERE子句对整个分区进行过滤，则Iceberg连接器支持删除整个分区。例如，下面代码将删除country=US的所有分区。

```
DELETE FROM iceberg.testdb.customer_orders
WHERE country = 'US'
```

目前，Iceberg连接器仅支持按分区删除。例如，下面代码选择分区中的一些行进行删除，运行则会报错。

```
DELETE FROM iceberg.testdb.customer_orders
WHERE country = 'US' AND customer = 'Freds Foods'
```

回滚

Iceberg支持数据的Snapshot模型，其中表快照由Snapshot ID标识。

Iceberg连接器为每个Iceberg表提供了一个系统快照表，快照由BIGINT类型的Snapshot ID标识，您可以通过运行以下命令查看customer_orders表的最新Snapshot ID。

```
SELECT snapshot_id FROM iceberg.testdb."customer_orders$snapshots" ORDER BY committed_at DESC LIMIT 1
```

使用system.rollback_to_snapshot可以将表的状态回滚到之前的快照ID。

```
CALL iceberg.system.rollback_to_snapshot('testdb', 'customer_orders', 895459706749342****)
```

系统表和列

Iceberg连接器支持查询系统表分区。例如，Iceberg表customer_orders，执行以下语句可以显示表分区，包括每个分区列的最大值和最小值。

```
SELECT * FROM iceberg.testdb."customer_orders$partitions"
```

Iceberg表属性

下表列出了Iceberg表的属性。

属性名	描述
format	指定表的数据文件存储格式。支持以下格式： • ORC（默认值） • PARQUET
partitioning	指定表的分区。 例如，表的分区列有c1和c2，该属性便为partitioning = ARRAY['c1', 'c2']。

属性名	描述
location	指定表所在的文件系统地址URI。

例如，下表定义了PARQUET格式的文件，由c1和c2列分区，文件系统地址为*/var/my_tables/test_table*。

```
CREATE TABLE test_table (
    c1 integer,
    c2 date,
    c3 double)
WITH (
    format = 'PARQUET',
    partitioning = ARRAY['c1', 'c2'],
    location = '/var/my_tables/test_table')
```

物化视图

Iceberg连接器支持物化视图，每个物化视图包含一个视图定义和Iceberg表，表名称存储在物化视图属性，数据存储在Iceberg表里。

物化视图支持操作如下表。

操作语句	描述
CREATE MATERIALIZED VIEW	创建并查询物化视图的数据。 您可以使用Iceberg表属性控制表存储格式。例如，使用ORC存储数据文件，使用_date列按天进行分区。 `WITH ( format = 'ORC', partitioning = ARRAY['event_date'] )`
REFRESH MATERIALIZED VIEW	更新物化视图的数据。 该操作会先删除Iceberg表数据，再插入物化视图Query定义的执行结果。 **注意** 删除和插入之间有一个小的时间窗口，当物化视图数据为空时，如果插入操作失败了，物化视图会保持空数据。 您也可以使用该语句，删除物化视图的定义和Iceberg表。

6.2.30.2.5.6. Hudi连接器

Hudi是一种数据湖的存储格式，在Hadoop文件系统之上提供了更新数据和删除数据的能力，以及消费变化数据的能力。EMR Presto已经将相关JAR包集成至独立的Hudi Plugin里面，EMR Hudi连接器目前支持查询COW和MOR表。

背景信息

EMR Hudi的详细信息，请参见Hudi概述。

前提条件

已创建Hadoop集群，并选择了Presto服务，或者创建单独的Presto集群，详情请参见创建集群。

使用限制

- EMR-3.38.0及后续版本的Hadoop集群或Presto集群，支持配置Hudi连接器。
- 只支持Hudi COW表的快照查询。
- 只支持Hudi MOR表的快照查询和读优化查询。
- 不支持增量查询。

配置连接器

修改Hudi连接器配置，详情请参见修改内置连接器。

Hudi连接器默认配置，您可以进入EMR控制台的Presto服务的**配置**页面，在**服务配置**区域，单击**hudi.properties**页签。您可以看到以下参数，参数值请根据您实际情况修改。

参数	描述

参数	描述
hive.recursive-directories	允许从表或分区所在位置的子目录读取数据，类似Hive的 `hive.mapred.supports.subdirectories` 属性。 默认值为false。
hive.metastore.uri	Hive Metastore使用Thrift协议连接的URI。 默认值格式 `thrift://emr-header-1.cluster-24****:9083` 。
hive.config.resources	HDFS配置文件的列表，多个配置文件时以逗号（,）分隔。这些配置文件必须存在于Presto运行的所有主机上。 注意 仅在必须访问HDFS的情况下配置此项。 默认值为 `/etc/ecm/hadoop-conf/core-site.xml, /etc/ecm/hadoop-conf/hdfs-site.xml` 。
hive.delta-table-enabled	是否支持Presto读取Delta Lake表。取值如下： • true（默认值）：Presto可以读取Delta Lake表。 • false：Presto不可以读取Delta Lake表。
hive.delta-compatible-mode-enabled	Delta Lake表是否启用兼容模式。取值如下： • true（默认值）：Delta Lake表启用兼容模式。 • false：Delta Lake表不启用兼容模式。
hive.hdfs.impersonation.enabled	是否启用用户代理。取值如下： • true（默认值）：启用用户代理。 • false：不启用用户代理。
hive.metastore.use-dlf-catalog	是否将Datalake Formation Catalog用作Hive Metastore。 默认值为default。

示例

Hudi表作为Hive的外表存储，可以通过连接Hive连接器来访问Hudi表进行数据查询。Hudi表的生成以及同步到Hive表中的步骤，请参见Hudi与Spark SQL集成和Spark写Hudi。

生成数据和查询数据示例如下所示：

1. 使用SSH方式登录EMR集群，详情请参见登录集群。
2. 执行以下命令，进入spark-sql命令行。

   ```
   spark-sql --conf 'spark.serializer=org.apache.spark.serializer.KryoSerializer' \
   --conf 'spark.sql.extensions=org.apache.spark.sql.hudi.HoodieSparkSessionExtension'
   ```

   当返回信息中包含如下信息时，表示已进入spark-sql命令行。

   ```
   spark-sql>
   ```

3. 执行以下命令，创建测试表。

   ```
   create table if not exists emr_test(
     id bigint,
     name string,
     price double
   ) using hudi
   options (
     type = 'mor',
     primaryKey = 'id,name'
   );
   ```

4. 执行以下命令，插入测试数据。

   ```
   insert into emr_test select 1, 'a2', 10;
   insert into emr_test select 1, 'a1', 10;
   insert into emr_test select 2, 'a1', 20;
   ```

> **说明** EMR的Spark SQL会自动同步Hudi数据到DLF或Hive MetaStore。

5. 在Presto客户端中查询数据。
   i. 执行以下命令，进入Presto客户端。

   ```
   presto --server emr-header-1:9090 --catalog hudi --schema default --user hadoop
   ```

   ii. 执行以下命令，查询表信息。

   ```
   select * from emr_test;
   ```

   返回信息如下。

   ```
    _hoodie_commit_time | _hoodie_commit_seqno | _hoodie_record_key | _hoodie_partition_path |
   _hoodie_file_name                             | id | name | price
   --------------------+----------------------+--------------------+------------------------+------------------------
   ------------------------------------------------+----+------+-------
    20211025145616      | 20211025145616_0_1   | id:1,name:a2       |                        | ac4ec1e6-528d-4189-bde6
   -d09e137f63f6-0_0-20-1604_20211025145616.parquet |  1 | a2   |  10.0
    20211025145629      | 20211025145629_0_1   | id:1,name:a1       |                        | ac4ec1e6-528d-4189-bde6
   -d09e137f63f6-0_0-48-3211_20211025145629.parquet |  1 | a1   |  10.0
    20211025145640      | 20211025145640_0_2   | id:2,name:a1       |                        | ac4ec1e6-528d-4189-bde6
   -d09e137f63f6-0_0-76-4818_20211025145640.parquet |  2 | a1   |  20.0
   (3 rows)
   ```

6. 在spark-sql中更新数据。
   i. 执行以下命令，进入spark-sql命令行。

   ```
   spark-sql --conf 'spark.serializer=org.apache.spark.serializer.KryoSerializer' \
   --conf 'spark.sql.extensions=org.apache.spark.sql.hudi.HoodieSparkSessionExtension'
   ```

   当返回信息中包含如下信息时，表示已进入spark-sql命令行。

   ```
   spark-sql>
   ```

   ii. 执行以下命令，更新id为2的price。

   ```
   update emr_test set price = price + 20 where id = 2;
   ```

7. 更新数据后，在Presto客户端中查询数据。
   i. 执行以下命令，进入Presto客户端。

   ```
   presto --server emr-header-1:9090 --catalog hudi --schema default --user hadoop
   ```

   ii. 执行以下命令，查询表信息。

   ```
   select * from emr_test;
   ```

   返回信息如下。

   ```
    _hoodie_commit_time | _hoodie_commit_seqno | _hoodie_record_key | _hoodie_partition_path |
   _hoodie_file_name                             | id | name | price
   --------------------+----------------------+--------------------+------------------------+------------------------
   ------------------------------------------------+----+------+-------
    20211025145616      | 20211025145616_0_1   | id:1,name:a2       |                        | ac4ec1e6-528d-4189-bde6
   -d09e137f63f6-0_0-20-1604_20211025145616.parquet |  1 | a2   |  10.0
    20211025145629      | 20211025145629_0_1   | id:1,name:a1       |                        | ac4ec1e6-528d-4189-bde6
   -d09e137f63f6-0_0-48-3211_20211025145629.parquet |  1 | a1   |  10.0
    20211025145640      | 20211025145640_0_2   | id:2,name:a1       |                        | ac4ec1e6-528d-4189-bde6
   -d09e137f63f6-0_0-76-4818_20211025145640.parquet |  2 | a1   |  40.0
   (3 rows)
   ```

6.2.30.2.5.7. Delta连接器

EMR Presto提供了独立的Delta连接器，在E-MapReduce集群上支持了较为完整的数据湖特性并进行了特性扩展。

背景信息

Delta Lake是DataBricks公司推出的一种数据湖方案，以数据为中心，围绕数据流走向推出了一系列功能特性，详情请参见Delta Lake概述。

前提条件

已创建Hadoop集群，并选择了Presto服务，或者创建单独的Presto集群，详情请参见创建集群。

使用限制

EMR-3.39.1及后续版本、EMR-5.5.0及后续版本的Hadoop集群或Presto集群，支持配置Delta连接器。

基础使用

修改Delta连接器配置，详情请参见修改内置连接器。

进入EMR控制台的Presto服务的配置页面，在服务配置区域，单击**delta.properties**页签，您可以看到以下参数，参数值请根据您实际情况修改。

参数	描述
hive.metastore.uri	Hive Metastore使用Thrift协议连接的URI。参数值您可以根据实际情况修改，默认格式为*thrift://emr-header-1.cluster-24****:9083*。
hive.config.resources	Hive Metastore使用的资源文件位置。

Presto无法新建或修改Delta Lake表，可以使用Spark-sql来创建，详情请参见基础使用。

1. 生成数据。
   i. 执行以下命令，进入Spark-sql命令行。
   ```
   spark-sql
   ```
   ii. 执行以下命令，创建Delta表。
   ```
   CREATE TABLE delta_table (id INT) USING delta;
   ```
   iii. 执行以下命令，写入数据。
   ```
   INSERT INTO delta_table VALUES 0,1,2,3,4;
   ```
2. 查询数据。
   i. 执行下列语句，进入Presto客户端。
   ```
   presto --server emr-header-1:9090 --catalog delta --schema default
   ```
   ii. 执行以下命令，查询表信息。
   ```
   SELECT * FROM delta_table;
   ```
   返回信息如下。
   ```
    id
   ----
     0
     1
     2
     3
     4
   (5 rows)
   ```

修改Delta连接器配置

连接器默认配置

示例

高阶使用

Time Travel允许查询表的历史数据。

EMR Presto支持Delta表的Time Travel特性，语法为 `for xxx as of` ，其中 `xxx` 的值可以为VERSION或TIMESTAMP，分别对应版本号和时间戳两种Time travel模式。

> 注意 Presto支持的Time Travel语法和Delta Lake在Spark SQL上的语法相比，多了一个 `FOR` 关键字。

示例如下：

1. 执行以下命令，进入Spark-sql命令行。
   ```
   spark-sql
   ```
2. 执行以下命令，覆盖数据。

```
INSERT OVERWRITE TABLE delta_table VALUES 5,6,7,8,9;
```

3. 查询数据。
   i. 执行下列语句，进入Presto客户端。

   ```
   presto --server emr-header-1:9090 --catalog delta --schema default
   ```

   ii. 执行以下命令，查询表信息。

   ```
   SELECT * FROM delta_table;
   ```

   返回信息如下。

   ```
    id
   ----
     5
     6
     7
     8
     9
   (5 rows)
   ```

4. 使用Time Travel查询历史数据。

   执行以下命令，按版本号查询数据，直接填写版本号即可。 版本号是一个单调递增的整数。默认第一次INSERT之后版本号为1，之后每修改一次版本号加1。

   ```
   SELECT * FROM delta_table FOR VERSION AS OF 1;
   ```

   返回信息如下。

   ```
    id
   ----
     2
     1
     3
     4
     0
   (5 rows)
   ```

   按时间戳查询数据，共支持DATE、TIMESTAMP和TIMESTAMP WITH TIME ZONE三种类型的时间戳。

   - DATE类型：查询日期所对应的UTC时间00:00:00的数据。
   - TIMESTAMP类型：查询指定时间戳对应的UTC的数据。

     例如，使用TIMESTAMP类型查询北京时间（+08:00）2022年2月15日20点整的数据，则代码如下。

     ```
     SELECT * FROM delta_table FOR TIMESTAMP AS OF TIMESTAMP '2022-02-15 12:00:00';
     ```

     > **说明** 其中，第一个TIMESTAMP说明使用的是时间戳进行Time Travel查询（非版本号），第二个TIMESTAMP则说明时间戳是TIMESTAMP类型（非DATE类型）。

     返回信息如下。

     ```
      id
     ----
       2
       0
       3
       4
       1
     (5 rows)
     ```

   - TIMESTAMP WITH TIME ZONE类型：无法直接读取数据，需要进行格式转换。

     例如，查询北京时间（+08:00）2022年2月15日20点的数据。代码示例如下。

     ```
     SELECT * FROM delta_table FOR TIMESTAMP AS OF CAST('2022-02-15 20:00:00 +0800' AS TIMESTAMP WITH TIME ZONE);
     ```

Presto基于Z-Order优化了Delta表查询。目前支持Parquet自身的优化和Data Skipping的优化。执行优化后，Delta会按文件粒度统计各个字段的最大和最小值，该统计信息用于直接过滤数据文件。Presto的Delta连接器可以读取到这些统计信息。

对于使用OPTIMIZE和ZORDER BY命令优化过的Delta表，在Z-Order列设置合适时，Presto的查询速度最大能够提升数十倍。具体优化方法请参见通过文件管理优化性能。

Presto支持Z-order的数据类型有Int、Long、Double、Float、Binary、Boolean、String和Array。

Presto支持Z-Order Data Skipping的谓词有 `=` 、 `<` 、 `<=` 、 `>` 和 `>=` 。

> **说明** Presto暂不支持like和in等谓词，但由于Z-order的局部排序能力，这些谓词在Z-order优化后同样可以提升查询速度。

例如，表conn_zorder，共含有src_ip、src_port、dst_ip和dst_port四列。

先在Spark中执行优化，命令如下所示。

```
OPTIMIZE conn_zorder ZORDER BY (src_ip, src_port, dst_ip, dst_port);
```

> **注意** 括号中的顺序即为Z-Order的顺序。

OPTIMIZE操作会根据数据量大小耗费一定时间。优化完成后，执行符合条件的查询均会提升性能。

- 查询一部分Z-Order优化的列能提升性能，命令如下所示。

  ```
  SELECT COUNT(*) FROM conn_zorder WHERE src_ip > '64.';
  ```

- 按Z-Order的优化顺序执行查询，速度提升非常大，命令如下所示。

  ```
  SELECT COUNT(*) FROM conn_zorder WHERE src_ip >= '64.' AND dst_ip < '192.' AND src_port < 1000 AND dst_port > 50000;
  ```

Time Travel

Z-Order

6.2.30.3. 高阶使用

6.2.30.3.1. 管理LDAP认证

服务开启LDAP认证功能后，访问服务需要提供LDAP身份认证（LDAP用户名和密码），可以提升服务的安全性。开启LDAP认证的功能可以方便您使用LDAP认证，避免了复杂的配置过程。

前提条件

已创建EMR-3.34.0及后续版本或EMR-4.8.0及后续版本的Hadoop集群，请参见创建集群。

开启LDAP认证

> **注意** Presto开启LDAP认证后，Hue访问Presto需要进行额外的配置，请参见Hue连接开启LDAP认证的引擎。

1. 进入Presto页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务 > Presto**。
2. 开启LDAP认证。
   i. 在Presto服务页面，选择右上角的**操作 > 开启LDAP认证**。
   ii. 在**执行集群操作**对话中，单击**确认**。
3. 单击上方的**查看操作历史**。

   直至操作状态显示**成功**。
4. 重启PrestoMaster。
   i. 在Presto服务页面，选择右上角的**操作 > 重启PrestoMaster**。
   ii. 在**执行集群操作**对话中，输入**执行原因**，单击**确定**。
   iii. 在**确认**对话中，单击**确定**。

访问Presto

开启LDAP认证后，当您访问Presto时需要提供LDAP认证凭据。

1. 通过SSH方式连接集群，请参见登录集群。
2. 您可以执行以下命令访问Presto。

因为Presto开启LDAP认证需要关闭HTTP端口（9090），所以只能使用HTTPS端口（7778）进行访问Presto。

```
presto --server https://emr-header-1.cluster-xxxx:7778 --keystore-path /etc/ecm/presto-conf/keystore --keystore-passwor
d <keystore_password> --catalog hive --user <user> --password
```

- emr-header-1.cluster-xxxx为集群PrestoMaster所在节点的长域名，获取方式为在该节点执行 `hostname` 命令。
- keystore_password为keystore的密码。需要在EMR控制台Presto的配置中查看，其中配置项keystore_password中的值即为keystore的密码。
- user为LDAP的用户名。

执行上述命令后，需要输入password，输入的password为LDAP的密码，获取方式请参见管理用户。

关闭LDAP认证

1. 进入Presto页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务 > Presto**。
2. 关闭LDAP认证。
   i. 在Presto服务页面，选择右上角的**操作 > 关闭LDAP认证**。
   ii. 在**执行集群操作**对话中，单击**确认**。
3. 单击上方的**查看操作历史**。
   直至操作状态显示**成功**。
4. 重启PrestoMaster。
   i. 在Presto服务页面，选择右上角的**操作 > 重启PrestoMaster**。
   ii. 在**执行集群操作**对话中，输入**执行原因**，单击**确定**。
   iii. 在**确认**对话中，单击**确定**。

6.2.30.3.2. 动态加载UDF

本文为您介绍阿里云EMR Presto如何动态加载和更新UDF。

背景信息

之前版本Presto新增UDF需要把JAR包上传到所有节点，然后重启服务器生效，此方式不是很方便。在on ACK场景下，如果需要使用新的UDF，还需重新打镜像，此方式不太友好。因此EMR-3.39.1和EMR-5.5.0及之后版本的阿里云EMR Presto实现了UDF的动态加载和更新。

EMR-3.39.1和EMR-5.5.0及之后版本的EMR集群，已在Presto的*config.properties*文件中添加udf.hadoop.config.resources配置，该参数值为UDF所在文件系统的资源配置文件的绝对路径（即*core-site.xml*位置）。该配置的目的是读取文件系统的配置。

- Hadoop集群：参数值为/etc/ecm/hadoop-conf/core-site.xml。
- Presto集群：参数值为/usr/lib/presto-current/etc/core-site.xml。

前提条件

已创建Hadoop集群，并选择了Presto服务，或者创建单独的Presto集群，详情请参见创建集群。

使用限制

EMR-3.39.1及后续版本、EMR-5.5.0及后续版本的Hadoop集群或Presto集群，支持UDF的动态加载和更新。

注意事项

- 如果本文示例中的 `xxxxxx` 的内容为现有连接器，例如*hive*、*mysql*等，执行DROP命令会直接删除该连接器目录下的所有内容，且无法恢复，因此请谨慎操作。
- 集群扩容时，扩容新增的节点不含上传到原有节点的UDF包，因此在扩容前先执行DROP命令，扩容完成后再重新执行ADD命令。

操作流程

1. 步骤一：环境准备
2. 步骤二：增加UDF
3. 步骤三：删除UDF

步骤一：环境准备

1. （可选）在Presto的*config.properties*配置文件中新增文件系统相关的配置。支持HDFS、集群本地以及OSS三种文件系统。

   > **注意** 您可以在后续执行ADD命令报无法识别HDFS、File或OSS问题时进行相应的配置。

文件系统	key	value
HDFS	fs.hdfs.impl	org.apache.hadoop.hdfs.DistributedFileSystem
集群本地	fs.file.impl	org.apache.hadoop.fs.LocalFileSystem
OSS	fs.oss.impl	com.aliyun.emr.fs.oss.JindoOssFileSystem

2. 将UDF对应的JAR包文件放在对应的文件系统中。

   建议使用OSS方式。OSS需与集群处于同一个区域，且集群需要有读取文件所在OSS的权限。处于同一个账号的EMR集群默认已具有读权限，可以直接访问。

   目前UDF文件支持以下两种方式：

   - 方式一：将UDF的全部内容打成一个JAR包（例如，*udfjar.jar*），上传到文件系统中。如果OSS属于其他账号，或使用的是ACK集群，则需保证该JAR包有公开的读权限。

     > **注意** JAR包名不可与现有连接器或UDF重名。

   - 方式二：如果UDF依赖多个JAR包，且不想打成一个包，可以将UDF的所有JAR包上传到同一个目录下（例如，*udfdir*），再将该目录完整上传到文件系统中。

     > **注意**
     > - 目录名不可与现有连接器或UDF重名。
     > - 目录下最好不要有无关内容。
     > - 目录需要配置权限。

步骤二：增加UDF

1. 启动客户端，连接Presto，并输入以下指令增加UDF包。
   i. 通过SSH方式登录集群，详情请参见登录集群。
   ii. 执行如下命令，进入Presto控制台。

   ```
   presto --server emr-header-1:9090
   ```

2. 执行如下命令，增加UDF。

   此操作会在Presto安装路径的*plugin*目录下新建一个文件夹，然后将路径所对应的文件上传到新建的文件夹下，并刷新当前函数列表。语法为 `add jar "xxxxxx"` 。

   - 如果是步骤一：环境准备中的方式一，则对应命令如下。

     ```
     add jar "oss://路径/udfjar.jar"
     ```

   - 如果是步骤一：环境准备中的方式二，则对应命令如下。

     ```
     add jar "oss://路径/udfdir"
     ```

     此时Presto能够识别出要上传的是一个目录，并将目录下的所有内容下载到集群上。

   - 如果是将UDF文件保存在HDFS上，则对应命令如下。

     ```
     add jar "hdfs://xxxxxx";
     ```

   - 如果是将UDF文件保存在本地文件系统上，则对应命令如下。

     ```
     add jar "file:///xxxxxx";
     ```

     > **注意** 在使用本地文件系统时，需要将UDF文件上传到Presto运行的所有节点的对应路径下，`file` 后面需要三个正斜线（/）。

步骤三：删除UDF

直接删除*plugin*目录下对应名称的整个目录，指定UDF包名即可删除UDF，并重新加载函数列表。

语法为 `DROP JAR xxxxxx` 。

> 说明 `xxxxxx`为上传的文件名，同时也是Presto读取UDF plugin的路径名。

删除语句不需要加引号，且无论通过方式一还是方式二上传，删除时均不需要加`jar`后缀。示例如下：

- `drop jar udfjar;`
- `drop jar udfdir;`

6.2.31. Flume

6.2.31.1. 概述

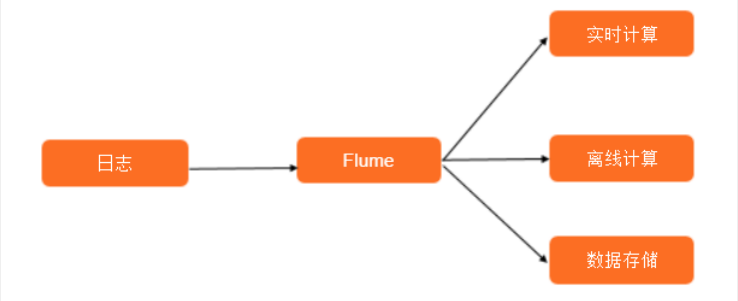
Apache Flume是一个分布式、可靠和高可用的系统，可以从大量不同的数据源有效地收集、聚合和移动日志数据，从而集中式的存储数据。

EMR-3.19.0及后续版本的集群，您可以在E-MapReduce控制台配置和管理Flume Agent。

使用场景

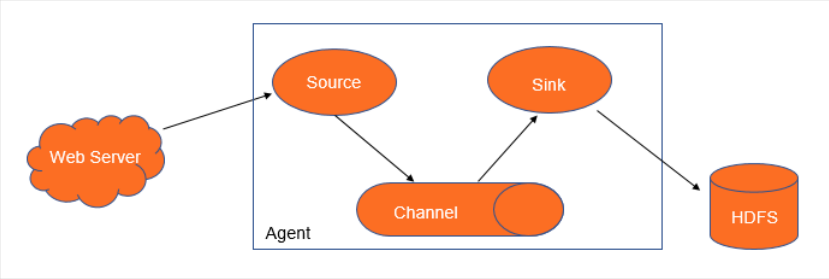
Flume使用最多的场景是日志收集，也可以通过定制Source来传输其他不同类型的数据。

Flume最终会将数据落地到实时计算平台（例如Flink、Spark Streaming和Storm）、离线计算平台上（例如MR、Hive和Presto），也可仅落地到数据存储系统中（例如HDFS、OSS、Kafka和Elasticsearch），为后续分析数据和清洗数据做准备。



架构

Flume Agent是一个Flume的实例，本质是一个JVM进程，控制Event数据流从生产者传输到消费者。一个Flume Agent由Source、Channel、Sink组成。其中，Source和Channel可以是一对多的关系，Channel和Sink也可以是一对多的关系。



基本概念

名称	描述
Event	是数据流通过Flume Agent的基本单位。Event由一个可选的Header字典和一个装载数据的字节数组组成。 示例如下。 `---------- | Header (Map) | Body (byte[]) | ---------- Flume Event`

名称	描述
Source	是数据源收集器，从外部数据源收集数据，并批量发送到一个或多个Channel中。 常见Source如下： • Avro Source：通过监听Avro端口获取Avro Client发送的事件。Avro是Hadoop提供的一种协议，用于数据序列化。 • Exec Source：通过监听命令行输出获取数据，例如 `tail -f /var/log/messages` 。 • NetCat TCP Source：监听指定TCP端口获取数据，与Netcat UDP Source类似。 • Taildir Source：监控目录下的多个文件，记录偏移量，并且不会丢失数据，较为常用。
Channel	是Source和Sink之间的缓冲队列。 常见Channel如下： • Memory Channel：缓存到内存中，性能高，较为常用。 • File Channel：缓存到文件中，会记录Checkpoint和DATA文件，可靠性高，但性能较差。 • JDBC Channel：缓存到关系型数据库中。 • Kafka Channel：通过Kafka来缓存数据。
Sink	从Channel中获取Event，并将以事务的形式Commit到外部存储中。一旦事务Commit成功，该Event会从Channel中移除。 常见Sink如下： • Logger Sink：用于测试。 • Avro Sink：转换成Avro Event，主要用于连接多个Flume Agent。 • HDFS Sink：写入HDFS，较为常用。 • Hive Sink：写入Hive表或分区，使用Hive事务写Events。 • Kafka Sink：写入Kafka。

6.2.31.2. 高阶使用

本文通过示例为您介绍E-MapReduce中的Flume组件，如何配置拦截器（Interceptor）、Channel选择器（Channel Selector）和Sink组逻辑处理器（Sink Processor）。

拦截器

拦截器的位置在Source和Channel之间，用于修改或丢弃Event。拦截图示意图如下。

拦截器的主要类型如下表。

类型	描述
时间戳拦截器	在Event Header中添加Unix时间戳属性。
Host拦截器	在Event Header中添加Host属性。
静态拦截器	在Event Header中添加一个固定键值对属性。
Header拦截器	在Event Header中删除一个或多个属性。
UUID拦截器	在Event中设置一个UUID。如果应用层没有UUID，则可以使用该拦截器来默认添加。
Morphline拦截器	通过Morphline配置文件过滤Event或修改插入Event Header。
查找拦截器	使用Java正则表达式查找Event Body。
替换拦截器	使用Java正则表达式替换Event Body。
正则过滤拦截器	过滤配置匹配或者没有匹配上正则表达式的Event Body。

相关示例如下：

- 示例1：Event Body包含 `1:2:3.4foobar5` ，如果想配置正则过滤器，则配置如下。

```
a1.sources.r1.interceptors.i1.regex = (\\d):(\\d):(\\d)
a1.sources.r1.interceptors.i1.serializers = s1 s2 s3
a1.sources.r1.interceptors.i1.serializers.s1.name = one
a1.sources.r1.interceptors.i1.serializers.s2.name = two
a1.sources.r1.interceptors.i1.serializers.s3.name = three
```

修改后的Event Body不变，后续的Header中增加了 `one=>1, two=>2, three=>3` 。

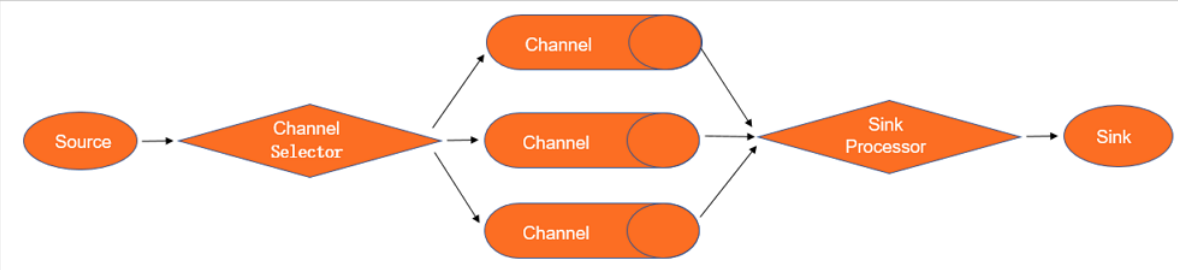
- 示例2：Event Body包含 `2012-10-18 18:47:57,614 some log line` ，如果想配置时间过滤器，则配置如下。

```
a1.sources.r1.interceptors.i1.regex = ^(?:\\n)?(\\d\\d\\d\\d-\\d\\d-\\d\\d\\s\\d\\d:\\d\\d)
a1.sources.r1.interceptors.i1.serializers = s1
a1.sources.r1.interceptors.i1.serializers.s1.type = org.apache.flume.interceptor.RegexExtractorInterceptorMillisSerialize
r
a1.sources.r1.interceptors.i1.serializers.s1.name = timestamp
a1.sources.r1.interceptors.i1.serializers.s1.pattern = yyyy-MM-dd HH:mm
```

修改后的Event Body不变，后续的Header中增加了 `timestamp=>1350611220000` 。

Channel选择器

Channel选择器用于在Source与Channel一对多场景下选择Channel。Channel选择器示意图如下。



Flume内置复制选择器（Replicating）和多路复用选择器（Multiplexing）两种选择器，默认为复制选择器。复制选择器会把所有Event发送到每个Channel，而多路复用选择器，则会按照一定的规则发送。多路复用选择器的示例如下。

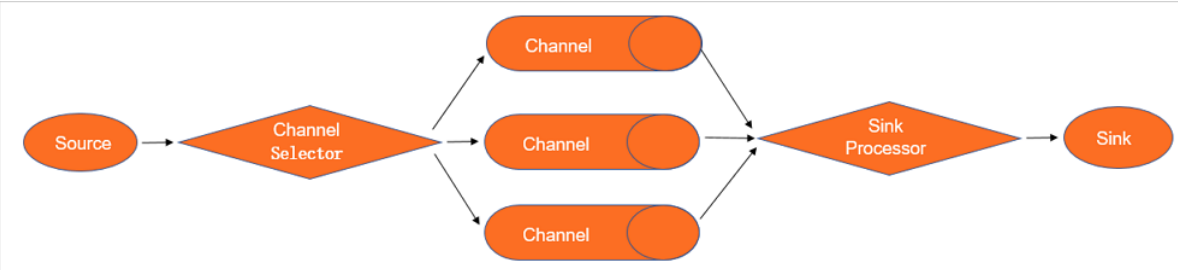
```
a1.sources = r1
a1.channels = c1 c2 c3 c4
a1.sources.r1.selector.type = multiplexing
a1.sources.r1.selector.header = state
a1.sources.r1.selector.mapping.CZ = c1
a1.sources.r1.selector.mapping.US = c2 c3
a1.sources.r1.selector.default = c4
```

上述示例中，r1会选择性地将Event发送给c1、c2、c3和c4四个Channel。如果Header中的state属性值为CZ，则发送给c1，如果属性值为US，则发送给c2和c3，其他情况默认发送给c4。

Sink组逻辑处理器

Sink组逻辑处理器示意图请参见Channel选择器的示意图。

Sink组逻辑处理器用于多个Sink一同消费Channel队列中的数据，并把这些Sink配置为负载均衡或故障转移的工作方式。默认Sink与Channel是一对一的。配置为负载均衡方式，则根据配置的负载均衡机制，将Event分发到Sink中。配置为故障转移方式，则表示多个Sink是一主多备的工作方式，当工作的Sink中止后，Event会被转移到备用的Sink上。



相关示例如下：

- 示例1：故障转移方式

```
a1.sinkgroups = g1
a1.sinkgroups.g1.sinks = k1 k2
a1.sinkgroups.g1.processor.type = failover
a1.sinkgroups.g1.processor.priority.k1 = 5
a1.sinkgroups.g1.processor.priority.k2 = 10
a1.sinkgroups.g1.processor.maxpenalty = 10000
```

上述示例中有k1和k2两个Sink，权重分别是5和10，最大故障转移时间是10000毫秒。

- 示例2：负载均衡方式

```
a1.sinkgroups = g1
a1.sinkgroups.g1.sinks = k1 k2
a1.sinkgroups.g1.processor.type = load_balance
a1.sinkgroups.g1.processor.backoff = true
a1.sinkgroups.g1.processor.selector = random
```

上述示例中有k1和k2两个Sink，通过随机选择（random）方法进行负载分配，您也可以使用轮询（round_robin）方法。示例中的`a1.sinkgroups.g1.processor.backoff`参数表示是否以指数的形式退避失败的Sinks，设置为true，则Sink Processor会屏蔽故障的Sink。

6.2.31.3. 开发指南

6.2.31.3.1. 安装第三方插件

安装第三方插件，可以在不修改Flume源码的情况下，以插件的方式定制您需要的功能。例如，添加数据源和存储组件等。通过添加JindoFS SDK的插件，可以支持通过Flume直接将数据写入OSS。本文为您介绍如何安装第三方插件。

前提条件

- 已获取第三方插件的JAR包。
- 已创建集群，并且选择了Flume服务，详情请参见创建集群。

操作步骤

支持以下方式：

- 方式一：修改*flume-env.sh*文件，添加指定的JAR包。

  > **说明** 本文以EMR集群为例介绍，Flume的配置文件目录在*/etc/ecm/flume-conf/*下，非EMR集群请修改为您实际的目录。

  i. 通过SSH方式登录集群，详情请参见登录集群。

  ii. 执行以下命令，进入Flume的配置文件目录。

  ```
  cd /etc/ecm/flume-conf/
  ```

  iii. 编辑*flume-env.sh.template*文件。

  a. 执行以下命令，打开文件*flume-env.sh.template*。

  ```
  vim flume-env.sh.template
  ```

  b. 按下`i`键进入编辑模式。

  c. 在*flume-env.sh.template*文件中，指定参数`FLUME_CLASSPATH`的值为待添加的JAR包。

  d. 按下`Esc`键退出编辑模式，输入`:wq`保存并关闭文件。

- 方式二：上传JAR包至Flume的软件安装目录*$FLUME_HOME/lib/*。

  > **说明** 本文示例中的`$FLUME_HOME`表示Flume的安装路径，配置第三方插件时，请根据实际情况指定路径。EMR集群中Flume的软件安装目录为*/usr/lib/flume-current/lib*。

- 方式三：上传JAR包至目录*$FLUME_HOME/plugins.d/*。

  相关的目录：

  - *$FLUME_HOME/plugins.d/lib*：插件本身的JAR包。
  - *$FLUME_HOME/plugins.d/libext*：插件依赖的JAR包。
  - *$FLUME_HOME/plugins.d/native*：依赖的本地库文件。例如，SO文件。

> **说明** EMR集群中，通常普通的插件都直接放在*/usr/lib/flume-current/lib*目录中，如果需要上传有复杂依赖的插件，特别是有native依赖的插件，请创建*/usr/lib/flume-current/plugins.d*目录，并按照方式三部署。

6.2.31.3.2. 自定义Source

通过自定义Source，您可以自行扩展更多的数据源，例如，加密的数据流、自建的服务端口和专有的数据存储中心等。本文通过示例为您介绍如何自定义Source。

前提条件

- 已创建集群，并且选择了Flume服务，详情请参见创建集群。
- 本地安装了文件传输工具（SSH Secure File Transfer Client）。

操作步骤

1. 创建自定义Source。
   i. 添加pom依赖。

```
<dependencies>
    <dependency>
        <groupId>org.apache.flume</groupId>
        <artifactId>flume-ng-core</artifactId>
        <version>1.9.0</version>
    </dependency>
</dependencies>
```

> **说明** `1.9.0` 为Flume的版本信息，需要根据您Flume的版本信息进行替换。

ii. 编写自定义的Source类。

`org.example.MySource` 实现了一个按照特定格式打印日志的Source。

```
package org.example;
import java.text.SimpleDateFormat;
import java.util.Date;
import org.apache.flume.Context;
import org.apache.flume.Event;
import org.apache.flume.EventDeliveryException;
import org.apache.flume.PollableSource;
import org.apache.flume.conf.Configurable;
import org.apache.flume.event.SimpleEvent;
import org.apache.flume.source.AbstractSource;
public class MySource extends AbstractSource implements Configurable, PollableSource {
    private String myDateFormat;
    private int myIntervalMS;
    @Override
    public void configure(Context context) {
        String myFormat = context.getString("dateFormat", "HH:mm:ss.SSS");
        int myInterval = context.getInteger("intervalMS", 1000);
        // Process the myProp value (e.g. validation, convert to another type, ...)
        // Store myProp for later retrieval by process() method
        this.myDateFormat = myFormat;
        this.myIntervalMS = myInterval;
    }
    @Override
    public void start() {
        // Initialize the connection to the external client
    }
    @Override
    public void stop () {
        // Disconnect from external client and do any additional cleanup
        // (e.g. releasing resources or nulling-out field values) ..
    }
    @Override
    public Status process() throws EventDeliveryException {
        Status status = null;
        try {
            // This try clause includes whatever Channel/Event operations you want to do
            // Receive new data
            Event e = new SimpleEvent();
            Date date = new Date();
            SimpleDateFormat sdf = new SimpleDateFormat(myDateFormat);
            e.setBody((sdf.format(date)).getBytes());
            // Store the Event into this Source's associated Channel(s)
            getChannelProcessor().processEvent(e);
            status = Status.READY;
        } catch (Exception e) {
            // Log exception, handle individual exceptions as needed
            status = Status.BACKOFF;
            e.printStackTrace();
        }
        try {
            Thread.sleep(myIntervalMS);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return status;
    }
    @Override
    public long getBackOffSleepIncrement() {
        return 0;
    }
    @Override
    public long getMaxBackOffSleepInterval() {
        return 0;
    }
}
```

2. 将自定义的代码打成JAR包。

   在*pom.xml*所在目录，执行如下命令制作JAR包。

```
mvn clean package -DskipTests
```

3. 使用文件传输工具，上传生成的JAR包至Flume的*/usr/lib/flume-current/lib*目录。

   > **说明** 非EMR集群时，请上传到您实际Flume的安装目录。

4. 新增配置。
   i. 通过SSH方式登录集群，详情请参见登录集群。
   ii. 执行以下命令，进入*/conf*目录。

   ```
   cd /usr/lib/flume-current/conf
   ```

   iii. 执行以下命令，新增配置文件。

   ```
   vim mysource.conf
   ```

   > **说明** 本文示例中配置文件为*mysource.conf*，您可以自定义文件名称。

   iv. 添加如下内容至配置文件*mysource.conf*中。

   ```
   a1.sources = r1
   a1.sinks = k1
   a1.channels = c1
   a1.sources.r1.type = org.example.MySource
   a1.sources.r1.dateFormat = HH:mm:ss.SSS
   a1.sources.r1.intervalMS = 2000
   a1.sinks.k1.type = logger
   a1.channels.c1.type = memory
   a1.channels.c1.capacity = 1000
   a1.channels.c1.transactionCapacity = 100
   a1.sources.r1.channels = c1
   a1.sinks.k1.channel = c1
   ```

   > **说明** 代码中的`dateFormat`表示日期格式，`intervalMS`表示间隔时间，单位ms。

5. 启动Flume。
   i. 执行以下命令，进入*/flume-current*目录。

   ```
   cd /usr/lib/flume-current
   ```

ii. 执行以下命令，启动Flume。

```
bin/flume-ng agent --name a1 -c conf -f conf/mysource.conf -Dflume.root.logger=INFO,console
```

返回如下信息。

```
2021-07-16 14:44:27,620 (conf-file-poller-0) [INFO - org.apache.flume.node.Application.startAllComponents(Applicati
on.java:169)] Starting Channel c1
2021-07-16 14:44:27,700 (lifecycleSupervisor-1-0) [INFO - org.apache.flume.instrumentation.MonitoredCounterGroup.re
gister(MonitoredCounterGroup.java:119)] Monitored counter group for type: CHANNEL, name: c1: Successfully registere
d new MBean.
2021-07-16 14:44:27,700 (lifecycleSupervisor-1-0) [INFO - org.apache.flume.instrumentation.MonitoredCounterGroup.st
art(MonitoredCounterGroup.java:95)] Component type: CHANNEL, name: c1 started
2021-07-16 14:44:27,701 (conf-file-poller-0) [INFO - org.apache.flume.node.Application.startAllComponents(Applicati
on.java:196)] Starting Sink k1
2021-07-16 14:44:27,701 (conf-file-poller-0) [INFO - org.apache.flume.node.Application.startAllComponents(Applicati
on.java:207)] Starting Source r1
2021-07-16 14:44:27,709 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.pr
ocess(LoggerSink.java:95)] Event: { headers:{} body: 31 34 3A 34 34 3A 32 37 2E 37 30 35             14:44:27.705 }
2021-07-16 14:44:29,709 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.pr
ocess(LoggerSink.java:95)] Event: { headers:{} body: 31 34 3A 34 34 3A 32 39 2E 37 30 39             14:44:29.709 }
2021-07-16 14:44:31,709 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.pr
ocess(LoggerSink.java:95)] Event: { headers:{} body: 31 34 3A 34 34 3A 33 31 2E 37 30 39             14:44:31.709 }
2021-07-16 14:44:33,710 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.pr
ocess(LoggerSink.java:95)] Event: { headers:{} body: 31 34 3A 34 34 3A 33 33 2E 37 31 30             14:44:33.710 }
2021-07-16 14:44:35,710 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.pr
ocess(LoggerSink.java:95)] Event: { headers:{} body: 31 34 3A 34 34 3A 33 35 2E 37 31 30             14:44:35.710 }
2021-07-16 14:44:37,710 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.pr
ocess(LoggerSink.java:95)] Event: { headers:{} body: 31 34 3A 34 34 3A 33 37 2E 37 31 30             14:44:37.710 }
2021-07-16 14:44:39,711 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.apache.flume.sink.LoggerSink.pr
ocess(LoggerSink.java:95)] Event: { headers:{} body: 31 34 3A 34 34 3A 33 39 2E 37 31 30             14:44:39.710 }
```

6.2.31.3.3. 自定义Sink

通过自定义Sink，您可以自行扩展更多的数据存储组件，或者根据需求裁剪和优化现有Sink的功能。本文通过示例为您介绍如何自定义Sink。

前提条件

- 已创建集群，并且选择了Flume服务，详情请参见创建集群。
- 本地安装了文件传输工具（SSH Secure File Transfer Client）。

操作步骤

1. 创建自定义Sink。
   i. 添加pom依赖。

```
<dependencies>
    <dependency>
        <groupId>org.apache.flume</groupId>
        <artifactId>flume-ng-core</artifactId>
        <version>1.9.0</version>
    </dependency>
</dependencies>
```

> **说明** `1.9.0` 为Flume的版本信息，需要根据您创建集群的信息替换。

ii. 编写自定义的Sink类。

`org.example.MySink` 仿照LoggerSink实现了一个默认Buffer更大的Sink。

```
package org.example;
import org.apache.flume.Channel;
import org.apache.flume.Context;
import org.apache.flume.Event;
import org.apache.flume.EventDeliveryException;
import org.apache.flume.Transaction;
import org.apache.flume.conf.Configurable;
import org.apache.flume.event.EventHelper;
import org.apache.flume.sink.AbstractSink;
import org.apache.flume.sink.LoggerSink;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
public class MySink extends AbstractSink implements Configurable {
    private static final Logger logger = LoggerFactory.getLogger(MySink.class);
    // Default Max bytes to dump
    public static final int DEFAULT_MAX_BYTE_DUMP = 32;
    // Max number of bytes to be dumped
    private int maxBytesToLog = DEFAULT_MAX_BYTE_DUMP;
    public static final String MAX_BYTES_DUMP_KEY = "maxBytesToLog";
    private String myProp;
    @Override
    public void configure(Context context) {
        this.maxBytesToLog = context.getInteger(MAX_BYTES_DUMP_KEY, DEFAULT_MAX_BYTE_DUMP);
    }
    @Override
    public void start() {
        // Initialize the connection to the external repository (e.g. HDFS) that
        // this Sink will forward Events to ..
    }
    @Override
    public void stop () {
        // Disconnect from the external respository and do any
        // additional cleanup (e.g. releasing resources or nulling-out
        // field values) ..
    }
    @Override
    public Status process() throws EventDeliveryException {
        Status status = Status.READY;
        // Start transaction
        Channel ch = getChannel();
        Transaction txn = ch.getTransaction();
        Event event = null;
        try {
            txn.begin();
            // This try clause includes whatever Channel operations you want to do
            event = ch.take();
            // Send the Event to the external repository.
            // storeSomeData(e);
            if (event != null) {
                if (logger.isInfoEnabled()) {
                    logger.info("Event: " + EventHelper.dumpEvent(event, maxBytesToLog));
                }
            } else {
                // No event found, request back-off semantics from the sink runner
                status = Status.BACKOFF;
            }
            txn.commit();
        } catch (Exception e) {
            txn.rollback();
            throw new EventDeliveryException("Failed to log event: " + event, e);
        } finally {
            txn.close();
        }
        return status;
    }
}
```

2. 将自定义的代码打成JAR包。

在*pom.xml*所在目录，执行如下命令制作JAR包。

```
mvn clean package -DskipTests
```

3. 使用文件传输工具，上传生成的JAR包至Flume的*/usr/lib/flume-current/lib*目录。

   > **说明** 非EMR集群时，请上传到您实际Flume的安装目录。

4. 新增配置。
   i. 通过SSH方式登录集群，详情请参见登录集群。
   ii. 执行以下命令，进入*/conf*目录。

   ```
   cd /usr/lib/flume-current/conf
   ```

   iii. 执行以下命令，新增配置文件。

   ```
   vim custom_sink.conf
   ```

   > **说明** 本文示例中配置文件为*custom_sink.conf*，您可以自定义文件名称。

   iv. 添加如下内容至配置文件*custom_sink.conf*中。

   ```
   a1.sources = r1
   a1.sinks = k1
   a1.channels = c1
   a1.sources.r1.type = org.apache.flume.source.StressSource
   a1.sources.r1.maxEventsPerSecond = 1
   a1.sources.r1.batchSize = 1
   a1.sources.r1.maxTotalEvents = 100
   a1.sinks.k1.type = org.example.MySink
   a1.sinks.k1.maxBytesToLog = 64
   a1.channels.c1.type = memory
   a1.channels.c1.capacity = 1000
   a1.channels.c1.transactionCapacity = 100
   a1.sources.r1.channels = c1
   a1.sinks.k1.channel = c1
   ```

   > **说明** 代码中的`maxBytesToLog`表示Buffer最大字节数。

5. 启动Flume。
   i. 执行以下命令，进入*/flume-current*目录。

   ```
   cd /usr/lib/flume-current
   ```

ii. 执行以下命令，启动Flume。

```
bin/flume-ng agent --name a1 -c conf -f conf/custom_sink.conf  -Dflume.root.logger=INFO,console
```

返回如下信息。

```
2021-07-16 14:49:29,024 (conf-file-poller-0) [INFO - org.apache.flume.node.Application.startAllComponents(Applicati
on.java:169)] Starting Channel c1
2021-07-16 14:49:29,024 (conf-file-poller-0) [INFO - org.apache.flume.node.Application.startAllComponents(Applicati
on.java:184)] Waiting for channel: c1 to start. Sleeping for 500 ms
2021-07-16 14:49:29,118 (lifecycleSupervisor-1-2) [INFO - org.apache.flume.instrumentation.MonitoredCounterGroup.re
gister(MonitoredCounterGroup.java:119)] Monitored counter group for type: CHANNEL, name: c1: Successfully registere
d new MBean.
2021-07-16 14:49:29,118 (lifecycleSupervisor-1-2) [INFO - org.apache.flume.instrumentation.MonitoredCounterGroup.st
art(MonitoredCounterGroup.java:95)] Component type: CHANNEL, name: c1 started
2021-07-16 14:49:29,525 (conf-file-poller-0) [INFO - org.apache.flume.node.Application.startAllComponents(Applicati
on.java:196)] Starting Sink k1
2021-07-16 14:49:29,525 (conf-file-poller-0) [INFO - org.apache.flume.node.Application.startAllComponents(Applicati
on.java:207)] Starting Source r1
2021-07-16 14:49:29,526 (lifecycleSupervisor-1-2) [INFO - org.apache.flume.source.StressSource.doStart(StressSource
.java:169)] Stress source doStart finished
2021-07-16 14:49:29,529 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.example.MySink.process(MySink.j
ava:66)] Event: { headers:{} body: 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................
00000010 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................
00000020 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................
00000030 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................ }
2021-07-16 14:49:30,006 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.example.MySink.process(MySink.j
ava:66)] Event: { headers:{} body: 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................
00000010 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................
00000020 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................
00000030 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................ }
2021-07-16 14:49:31,007 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.example.MySink.process(MySink.j
ava:66)] Event: { headers:{} body: 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................
00000010 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................
00000020 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................
00000030 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................ }
2021-07-16 14:49:32,007 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.example.MySink.process(MySink.j
ava:66)] Event: { headers:{} body: 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................
00000010 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................
00000020 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................
00000030 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................ }
2021-07-16 14:49:33,006 (SinkRunner-PollingRunner-DefaultSinkProcessor) [INFO - org.example.MySink.process(MySink.j
ava:66)] Event: { headers:{} body: 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................
00000010 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................
00000020 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................
00000030 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................ }
```

6.2.31.3.4. 常用参数调优

本文为您介绍Taildir Source、File Channel和HDFS Sink配置中的一些常用参数调优。

Taildir Source

参数	描述
filegroups	将一个目录拆分为多个，增加Source读取并发。
batchSize	默认值为100。一次性读取数据行数，适当调大该参数值可以提高吞吐量。

File Channel

参数	描述
checkpointInterval	默认值为30，单位为秒（s）。适当调小该参数值可以缩短写CheckPoint间隔。
useDualCheckpoints	默认值为false。修改该参数值为true，可以防止重启时因为CheckPoint丢失，从头读取所有Event。
maxFileSize	默认值为1.6，单位为GB。表示单个DATA文件的大小。 适当调小该参数值以便FileChannel可以尽快移除过期文件，减少占用磁盘。

参数	描述
capacity	默认值为1000000。表示Channel最多保留的Events数量。 调大该参数可以优化吞吐量，也可以使用该参数乘以单个Event大小来估算磁盘占用情况。
transactionCapacity	默认值为10000。Channel支持的单个事务的最大容量。

HDFS Sink

参数	描述
hdfs.batchSize	默认值为100。向HDFS写入内容时每次批量操作的Event数量。 适当调大该参数可提高吞吐量。 **说明** 建议此参数值和Source配置中batchSize的参数值保持一致，且均不应该超过Channel配置中transactionCapacity的参数值。
hdfs.threadsPoolSize	默认值为10，HDFS IO线程数，根据机器配置调整。
hdfs.useLocalTimeStamp	默认值为false。表示是否使用本地时间戳。 如果需要在Event的Head中添加时间戳，设置该参数值为true。
hdfs.rollInterval	默认值为30，单位为秒（s）。表示间隔多久临时文件滚动为目标文件。 设置为0时，表示不基于时间滚动。
hdfs.rollSize	默认值为1024，单位为字节（Byte）。表示文件大小到达该该参数值时，滚动为目标文件。 设置为0时，表示不基于大小滚动。
hdfs.rollCount	默认10个事件。表示事件数量达到该数量时滚动为目标文件。 设置为0时，表示不基于事件数量滚动。
hdfs.minBlockReplicas	默认为HDFS副本数，表示HDFS文件块的最小副本数。 通常配置为1，才能正确滚动文件。

6.2.31.4. 最佳实践

6.2.31.4.1. 同步HDFS Audit日志至HDFS

EMR Flume支持多种服务启动方式，本文介绍通过EMR控制台修改Flume配置并启动Flume Agent，继而实时同步HDFS Audit日志至HDFS。

前提条件

已创建E-MapReduce的Hadoop集群，并且选择了Flume服务。详情请参见创建集群。

启动Flume Agent

1. 进入Flume页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务 > FLUME**。
2. 配置emr-worker-1节点的Flume Agent。
   以下配置项请遵循开源Flume内容，详情请参见Flume。

i. 单击**配置**页签，修改如下参数。

配置项	参数值
default-agent.sinks.default-sink.type	hdfs
default-agent.channels.default-channel.type	file
default-agent.sources.default-source.type	avro
deploy_node_hostname	emr-worker-1

ii. 在**服务配置**区域，单击**flume-conf**页签。

iii. 单击**自定义配置**，添加如下配置项。

配置项	描述
default-agent.sinks.default-sink.hdfs.path	HDFS路径。 ■ 高可用集群：例如*hdfs://emr-cluster/path*。 ■ 非高可用集群：例如*hdfs://emr-header-1:9000/path*。
default-agent.sinks.default-sink.hdfs.fileType	DataStream
default-agent.sinks.default-sink.hdfs.rollSize	0
default-agent.sinks.default-sink.hdfs.rollCount	0
default-agent.sinks.default-sink.hdfs.rollInterval	86400
default-agent.sinks.default-sink.hdfs.batchSize	51200
default-agent.sources.default-source.bind	0.0.0.0
default-agent.sources.default-source.port	根据实际情况设置。
default-agent.channels.default-channel.transactionCapacity	可选参数，默认值为10000。
default-agent.channels.default-channel.dataDirs	Channel存储Event数据的路径。 可选参数，默认路径*~/.flume/file-channel/data*。
default-agent.channels.default-channel.checkpointDir	存储Checkpoint的路径。 可选参数，默认路径*~/.flume/file-channel/checkpoint*。
default-agent.channels.default-channel.capacity	根据HDFS Roll设置。 可选参数，默认值为1000000。

iv. 单击**确定**。

3. 单击**部署拓扑**页签。

   您可以查看Flume Agent组件emr-worker-1节点的状态信息，组件状态为**STARTED**。

4. 请参见上面步骤2配置emr-worker-2节点。

5. 配置Master实例的Flume Agent。

   以下配置项请遵循开源Flume内容，详情请参见Flume。

   i. 单击**配置**页签，修改如下参数。

配置项	参数值
additional_sinks	k1
deploy_node_hostname	emr-header-1
default-agent.sources.default-source.type	taildir
default-agent.sinks.default-sink.type	avro
default-agent.channels.default-channel.type	file

ii. 在**服务配置**区域，单击**flume-conf**页签。

iii. 单击**自定义配置**，添加如下配置项。

配置项	参数值
default-agent.sources.default-source.filegroups	f1
default-agent.sources.default-source.filegroups.f1	/mnt/disk1/log/hadoop-hdfs/hdfs-audit.log.*
default-agent.sources.default-source.positionFile	存储Position File的路径。 可选参数，默认路径为*~/.flume/taildir_position.json*。
default-agent.channels.default-channel.checkpointDir	存储Checkpoint的路径。
default-agent.channels.default-channel.dataDirs	存储Event数据的路径。
default-agent.channels.default-channel.capacity	根据HDFS Roll设置。
default-agent.sources.default-source.batchSize	2000
default-agent.channels.default-channel.transactionCapacity	2000
default-agent.sources.default-source.ignoreRenameWhenMultiMatching	true
default-agent.sinkgroups	g1
default-agent.sinkgroups.g1.sinks	default-sink k1
default-agent.sinkgroups.g1.processor.type	failover
default-agent.sinkgroups.g1.processor.priority.default-sink	10
default-agent.sinkgroups.g1.processor.priority.k1	5
default-agent.sinks.default-sink.hostname	emr-worker-1节点的IP地址。
default-agent.sinks.default-sink.port	emr-worker-1节点Flume Agent的Port。
default-agent.sinks.k1.hostname	emr-worker-2节点的IP地址。
default-agent.sinks.k1.port	emr-worker-2节点Flume Agent的Port。
default-agent.sinks.default-sink.batch-size	2000
default-agent.sinks.k1.batch-size	2000
default-agent.sinks.k1.type	avro
default-agent.sinks.k1.channel	default-channel

iv. 单击**确定**。

6. 启动Flume Agent。

i. 单击右上角的**保存**。

ii. 在**确认修改**对话中，输入**执行原因**，开启**自动更新配置**开关，单击**确定**。

iii. 在右上角选择**操作 > 重启All Components**。

iv. 在**执行集群操作**对话框中，输入**执行原因**，单击**确定**。

v. 在**确认**对话框，单击**确定**。
执行完成后，日志就从HDFS Audit同步至HDFS了。

查看日志

Flume Agent日志的存放路径为*/mnt/disk1/log/flume/default-agent/flume.log*。

查看监控信息

在Flume服务的**状态**页面，您可以查看Flume Agent的监控信息。

6.2.31.4.2. 同步EMR Kafka数据至HDFS

E-MapReduce（简称EMR）从EMR-3.16.0版本开始支持Apache Flume。本文介绍如何通过命令行方式，同步EMR Kafka集群的数据至EMR Hadoop集群的HDFS。

前提条件

- 已创建Hadoop集群，并且选择了Flume服务，详情请参见创建集群。

  > **说明** Flume软件安装目录在*/usr/lib/flume-current*下，其他常用文件路径获取方式请参见常用文件路径。

- 已创建Kafka集群，详情请参见创建集群。

  > **说明**
  > - 如果创建的是Hadoop高安全集群，消费标准Kafka集群的数据，则需要在Hadoop集群配置Kerberos认证，详情请参见兼容MIT Kerberos认证。
  > - 如果创建的是Kafka高安全集群，通过Flume将Kafka数据写入Hadoop集群，详情请参见 消费Kerberos Kafka Source。
  > - 如果创建的Hadoop集群和Kafka集群都是高安全集群，需配置跨域互信，详情请参见跨域互信，其它配置详情请参见跨域互信使用Flume。

同步Kafka数据至HDFS

1. 通过SSH方式连接Hadoop集群。

   详情请参见登录集群。

2. 配置Flume。

   i. 进入目录*/etc/ecm/flume-conf*。

   ```
   cd /etc/ecm/flume-conf
   ```

   ii. 创建配置文件*flume.properties*。

   ```
   vim flume.properties
   ```

iii. 添加如下内容。

```
a1.sources = source1
a1.sinks = k1
a1.channels = c1
a1.sources.source1.type = org.apache.flume.source.kafka.KafkaSource
a1.sources.source1.channels = c1
a1.sources.source1.kafka.bootstrap.servers = <kafka-host1:port1,kafka-host2:port2...>
a1.sources.source1.kafka.topics = flume-test
a1.sources.source1.kafka.consumer.group.id = flume-test-group
# Describe the sink
a1.sinks.k1.type = hdfs
a1.sinks.k1.hdfs.path = hdfs://emr-cluster/tmp/flume/test-data
a1.sinks.k1.hdfs.fileType=DataStream
# Use a channel which buffers events in memory
a1.channels.c1.type = memory
a1.channels.c1.capacity = <100>
a1.channels.c1.transactionCapacity = <100>
# Bind the source and sink to the channel
a1.sources.source1.channels = c1
a1.sinks.k1.channel = c1
```

- a1.sources.source1.kafka.bootstrap.servers：Kafka集群Broker的Host和端口号。
- a1.sources.source1.kafka.topics：Flume消费Kafka数据的Topic。
- a1.channels.c1.capacity：通道中存储的最大事件数。请根据实际环境修改该参数值。
- a1.channels.c1.transactionCapacity：每个事务通道将从源接收或提供给接收器的最大事件数。请根据实际环境修改该参数值。
- a1.sinks.k1.hdfs.path：Flume向HDFS写入数据的路径。

  针对普通集群和高可用集群代码示例如下：

  - 高可用集群

    ```
    a1.sinks.k1.hdfs.path = hdfs://emr-cluster/tmp/flume/test-data
    ```

  - 普通集群

    ```
    a1.sinks.k1.hdfs.path = hdfs://emr-header-1:9000/tmp/flume/test-data
    ```

3. 执行如下命令，启动服务。

```
flume-ng agent --name a1 --conf /etc/ecm/flume-conf --conf-file flume.properties
```

4. 测试数据同步情况。

i. 通过SSH方式连接Kafka集群，详情请参见登录集群。

ii. 创建名称为*flume-test*的Topic。

```
/usr/lib/kafka-current/bin/kafka-topics.sh --partitions 10 --replication-factor 2 --zookeeper emr-header-1:2181 /ka
fka-1.0.0 --topic flume-test --create
```

iii. 生成测试数据。

```
kafka-console-producer.sh --topic flume-test --broker-list emr-header-1:9092
```

例如输入 `abc` 并回车。

iv. 通过SSH方式连接Hadoop集群，查看生成的文件。

Flume会在HDFS中以当前时间的时间戳（毫秒）生成文件*FlumeData.xxxx*。文件内容是在Kafka中输入的数据 `abc` 。

```
hdfs dfs -cat /tmp/flume/test-data/<FlumeData.xxxx>
```

消费Kerberos Kafka source

消费高安全Kafka集群的数据时，需要完成额外的配置：

- 在Kafka集群配置Kerberos认证，将生成的*test.keytab*文件拷贝至Hadoop集群的*/etc/ecm/flume-conf*路径下，详情请参见兼容MIT Kerberos认证；将Kafka集群的*/etc/ecm/has-conf/krb5.conf*文件拷贝至Hadoop集群的*/etc/ecm/flume-conf*路径下。
- 配置*flume.properties*。

  在*flume.properties*中添加如下配置。

```
a1.sources.source1.kafka.consumer.security.protocol = SASL_PLAINTEXT
a1.sources.source1.kafka.consumer.sasl.mechanism = GSSAPI
a1.sources.source1.kafka.consumer.sasl.kerberos.service.name = kafka
```

- 配置Kafka client。
  - 在*/etc/ecm/flume-conf*下创建文件*flume_jaas.conf*。

```
KafkaClient {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  storeKey=true
  keyTab="/etc/ecm/flume-conf/test.keytab"
  serviceName="kafka"
  principal="test@EMR.${realm}.COM";
};
```

  ${realm} 需要替换为Kafka集群的Kerberos realm。

  ${realm}获取方式：在Kafka集群执行命令 `hostname` ，得到形式为 `emr-header-1.cluster-xxx` 的主机名，例如 `emr-header-1.cluster-123456` ，其中数字串123456即为realm。

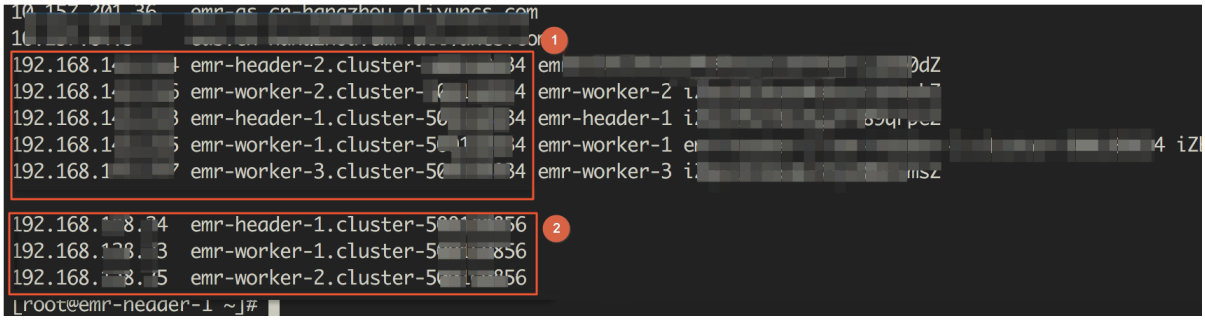
  - 修改*/etc/ecm/flume-conf/flume-env.sh*。

    初始情况下，*/etc/ecm/flume-conf/*下没有*flume-env.sh*文件，需要拷贝*flume-env.sh.template*并重命名为*flume-env.sh*。在*flume-env.sh*文件末尾添加如下内容。

```
export JAVA_OPTS="$JAVA_OPTS -Djava.security.krb5.conf=/etc/ecm/flume-conf/krb5.conf"
export JAVA_OPTS="$JAVA_OPTS -Djava.security.auth.login.config=/etc/ecm/flume-conf/flume_jaas.conf"
```

- 设置域名。

  将Kafka集群各节点的长域名和IP的绑定信息添加到Hadoop集群的*/etc/hosts*文件末尾。长域名的形式为*emr-header-1.cluster-xxxx*。



> **说明** 图中标注①表示的是Hadoop集群的域名；图中标注②表示新增加的Kafka集群域名。

跨域互信使用Flume

在配置了跨域互信后，其他配置如下：

- 在Kafka集群配置Kerberos认证，将生成的keytab文件*test.keytab*拷贝至Hadoop集群的*/etc/ecm/flume-conf*路径下，详情请参见兼容MIT Kerberos认证。
- 配置*flume.properties*。

  在*flume.properties*中添加如下配置。

```
a1.sources.source1.kafka.consumer.security.protocol = SASL_PLAINTEXT
a1.sources.source1.kafka.consumer.sasl.mechanism = GSSAPI
a1.sources.source1.kafka.consumer.sasl.kerberos.service.name = kafka
```

- 配置Kafka client。
  - 在*/etc/ecm/flume-conf*下创建文件*flume_jaas.conf*，内容如下。

```
KafkaClient {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  storeKey=true
  keyTab="/etc/ecm/flume-conf/test.keytab"
  serviceName="kafka"
  principal="test@EMR.${realm}.COM";
};
```

  ${realm}替换为Kafka集群的Kerberos realm。

  ${realm}获取方式：在Kafka集群执行命令 `hostname` ，得到形式为 `emr-header-1.cluster-xxx` 的主机名，例如 `emr-header-1.cluster-123456` ，其中数字串123456即为realm。

- 修改*/etc/ecm/flume-conf/flume-env.sh*。

  初始情况下，*/etc/ecm/flume-conf/*下没有*flume-env.sh*文件，需要拷贝*flume-env.sh.template*并重命名为*flume-env.sh*。在*flume-env.sh*文件末尾添加如下内容。

  ```
  export JAVA_OPTS="$JAVA_OPTS -Djava.security.auth.login.config=/etc/ecm/flume-conf/flume_jaas.conf"
  ```

6.2.31.4.3. 同步EMR Kafka数据至Hive

E-MapReduce（简称EMR）从EMR-3.16.0版本开始支持Apache Flume。本文介绍如何通过命令方式，使用Flume同步EMR Kafka集群的数据至EMR Hadoop集群的Hive。

前提条件

- 已创建Hadoop集群，并且选择了Flume服务，详情请参见创建集群。

  > **说明** Flume软件安装目录在*/usr/lib/flume-current*下，其他常用文件路径获取方式请参见常用文件路径。

- 已创建Kafka集群，详情请参见创建集群。

  > **说明**
  > - 如果创建的是Hadoop高安全集群，消费标准Kafka集群的数据，需在Hadoop集群配置Kerberos认证，详情请参见兼容MIT Kerberos认证。
  > - 如果创建的是Kafka高安全集群，通过Flume将数据写入标准Hadoop集群，请参见 Kerberos Kafka Source。
  > - 如果创建的Hadoop集群和Kafka集群都是高安全集群，需配置跨域互信，详情请参见跨域互信，其它配置请参见跨域互信使用Flume。

同步Kafka数据至Hive

1. 通过SSH方式连接Hadoop集群。

   详情请参见登录集群。

2. 创建Hive表。

   Flume使用事务操作将数据写入Hive，需要在创建Hive表（*flume_test*）时设置transactional属性。

   ```
   create table flume_test (id int, content string)
   clustered by (id) into 2 buckets stored as orc  TBLPROPERTIES('transactional'='true');
   ```

3. 配置Flume。

   i. 进入目录*/etc/ecm/flume-conf*。

   ```
   cd /etc/ecm/flume-conf
   ```

   ii. 创建配置文件*flume.properties*。

   ```
   vim flume.properties
   ```

iii. 添加如下内容。

```
a1.sources = source1
a1.sinks = k1
a1.channels = c1
a1.sources.source1.type = org.apache.flume.source.kafka.KafkaSource
a1.sources.source1.channels = c1
a1.sources.source1.kafka.bootstrap.servers = <kafka-host1:port1,kafka-host2:port2...>
a1.sources.source1.kafka.topics = flume-test
a1.sources.source1.kafka.consumer.group.id = flume-test-group
# Describe the sink
a1.sinks.k1.type = hive
a1.sinks.k1.hive.metastore = thrift://xxxx:9083
a1.sinks.k1.hive.database = default
a1.sinks.k1.hive.table = flume_test
a1.sinks.k1.serializer = DELIMITED
a1.sinks.k1.serializer.delimiter = ","
a1.sinks.k1.serializer.serdeSeparator = ','
a1.sinks.k1.serializer.fieldnames =id,content
a1.channels.c1.type = memory
a1.channels.c1.capacity = <100>
a1.channels.c1.transactionCapacity = <100>
a1.sources.source1.channels = c1
a1.sinks.k1.channel = c1
```

- a1.sources.source1.kafka.bootstrap.servers：Kafka集群Broker的Host和端口号。
- a1.channels.c1.capacity：通道中存储的最大事件数。请根据实际环境修改该参数值。
- a1.channels.c1.transactionCapacity：每个事务通道将从源接收或提供给接收器的最大事件数。请根据实际环境修改该参数值。
- a1.sinks.k1.hive.metastore：Hive metastore的URI，格式为*thrift://emr-header-1.cluster-xxx:9083*。其中*emr-header-1.cluster-xxx*您可以通过 `hostname` 获取。

4. 执行如下命令，启动服务。

```
flume-ng agent --name a1 --conf /etc/ecm/flume-conf --conf-file flume.properties
```

5. 测试数据同步情况。

i. 通过SSH方式连接Kafka集群，详情请参见登录集群。

ii. 创建名称为*flume-test*的Topic。

```
/usr/lib/kafka-current/bin/kafka-topics.sh --partitions 10 --replication-factor 2 --zookeeper emr-header-1:2181 /ka
fka-1.0.0 --topic flume-test --create
```

iii. 生成测试数据。

```
kafka-console-producer.sh --topic flume-test --broker-list emr-header-1:9092
```

例如输入 `1,a` 并回车。

iv. 通过SSH方式连接Hadoop集群，在客户端配置Hive参数并查询表中的数据。

```
set hive.support.concurrency=true;
set hive.exec.dynamic.partition.mode=nonstrict;
set hive.txn.manager=org.apache.hadoop.hive.ql.lockmgr.DbTxnManager;
```

配置好后查询*flume_test*表中的数据。

```
select * from flume_test;
```

返回信息如下：

```
OK
1    a
```

消费Kerberos Kafka source

消费高安全Kafka集群的数据时，需要完成额外的配置：

- 在Kafka集群配置Kerberos认证，将生成的*test.keytab*文件拷贝至Hadoop集群的*/etc/ecm/flume-conf*路径下，详情请参见兼容MIT Kerberos认证；将Kafka集群的*/etc/ecm/has-conf/krb5.conf*文件拷贝至Hadoop集群的*/etc/ecm/flume-conf*路径下。
- 配置*flume.properties*。

  在*flume.properties*中添加如下配置。

```
a1.sources.source1.kafka.consumer.security.protocol = SASL_PLAINTEXT
a1.sources.source1.kafka.consumer.sasl.mechanism = GSSAPI
a1.sources.source1.kafka.consumer.sasl.kerberos.service.name = kafka
```

- 配置Kafka client。
  - 在*/etc/ecm/flume-conf*下创建文件*flume_jaas.conf*。

    ```
    KafkaClient {
      com.sun.security.auth.module.Krb5LoginModule required
      useKeyTab=true
      storeKey=true
      keyTab="/etc/ecm/flume-conf/test.keytab"
      serviceName="kafka"
      principal="test@EMR.${realm}.COM";
    };
    ```

    ${realm} 需要替换为Kafka集群的Kerberos realm。

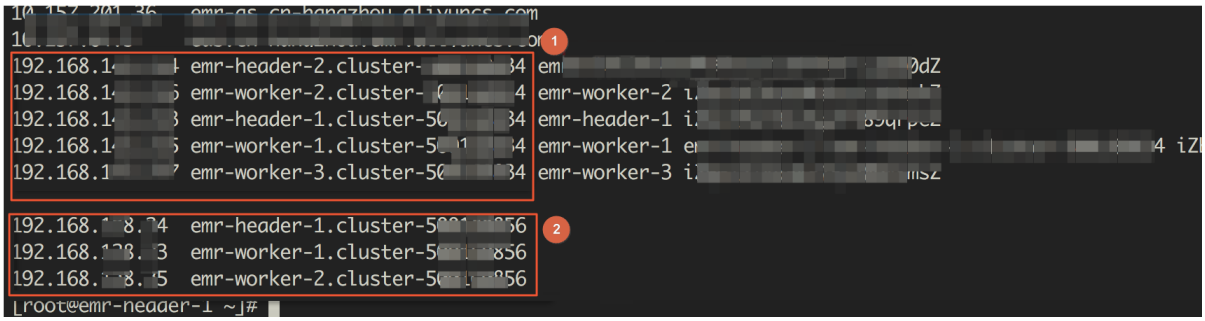
    ${realm}获取方式：在Kafka集群执行命令 `hostname` ，得到形式为 `emr-header-1.cluster-xxx` 的主机名，例如 `emr-header-1.cluster-123456` ，其中数字串123456即为realm。
  - 修改*/etc/ecm/flume-conf/flume-env.sh*。

    初始情况下，*/etc/ecm/flume-conf/*下没有*flume-env.sh*文件，需要拷贝*flume-env.sh.template*并重命名为*flume-env.sh*。在*flume-env.sh*文件末尾添加如下内容。

    ```
    export JAVA_OPTS="$JAVA_OPTS -Djava.security.krb5.conf=/etc/ecm/flume-conf/krb5.conf"
    export JAVA_OPTS="$JAVA_OPTS -Djava.security.auth.login.config=/etc/ecm/flume-conf/flume_jaas.conf"
    ```

- 设置域名。

  将Kafka集群各节点的长域名和IP的绑定信息添加到Hadoop集群的*/etc/hosts*文件末尾。长域名的形式为*emr-header-1.cluster-xxxx*。

  > **说明** 图中标注①表示的是Hadoop集群的域名；图中标注②表示新增加的Kafka集群域名。

跨域互信使用Flume

在配置了跨域互信后，其他配置如下：

- 在Kafka集群配置Kerberos认证，将生成的keytab文件*test.keytab*拷贝至Hadoop集群的*/etc/ecm/flume-conf*路径下，详情请参见兼容MIT Kerberos认证。
- 配置*flume.properties*。

  在*flume.properties*中添加如下配置。

  ```
  a1.sources.source1.kafka.consumer.security.protocol = SASL_PLAINTEXT
  a1.sources.source1.kafka.consumer.sasl.mechanism = GSSAPI
  a1.sources.source1.kafka.consumer.sasl.kerberos.service.name = kafka
  ```

- 配置Kafka client。

- 在*/etc/ecm/flume-conf*下创建文件*flume_jaas.conf*，内容如下。

```
KafkaClient {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  storeKey=true
  keyTab="/etc/ecm/flume-conf/test.keytab"
  serviceName="kafka"
  principal="test@EMR.${realm}.COM";
};
```

${realm}替换为Kafka集群的Kerberos realm。

${realm}获取方式：在Kafka集群执行命令`hostname`，得到形式为`emr-header-1.cluster-xxx`的主机名，例如`emr-header-1.cluster-123456`，其中数字串123456即为realm。

- 修改*/etc/ecm/flume-conf/flume-env.sh*。

初始情况下，*/etc/ecm/flume-conf/*下没有*flume-env.sh*文件，需要拷贝*flume-env.sh.template*并重命名为*flume-env.sh*。在*flume-env.sh*文件末尾添加如下内容。

```
export JAVA_OPTS="$JAVA_OPTS -Djava.security.auth.login.config=/etc/ecm/flume-conf/flume_jaas.conf"
```

6.2.31.4.4. 同步EMR Kafka数据至HBase

E-MapReduce（简称EMR）从EMR-3.16.0版本开始支持Apache Flume。本文介绍如何通过命令行方式，使用Flume同步EMR Kafka集群的数据至EMR Hadoop集群的HBase。

前提条件

- 已创建Hadoop集群，并且选择了Flume和HBase服务，详情请参见创建集群。

> **说明** Flume软件安装目录在*/usr/lib/flume-current*下，其他常用文件路径获取方式请参见常用文件路径。

- 已创建Kafka集群，详情请参见创建集群。

> **说明**
> - 如果创建的是Hadoop高安全集群，消费标准Kafka集群的数据，则需要在Hadoop集群配置Kerberos认证，详情请参见兼容MIT Kerberos认证。
> - 如果创建的是Kafka高安全集群，通过Flume将Kafka数据写入Hadoop集群，详情请参见消费Kerberos Kafka source。
> - 如果创建的Hadoop集群和Kafka集群都是高安全集群，需配置跨域互信，详情请参见跨域互信，其它配置详情请参见跨域互信使用Flume。

同步Kafka数据至HBase

1. 通过SSH方式连接Hadoop集群。

   详情请参见登录集群。

2. 创建HBase表*flume_test*及列簇column。

```
create 'flume_test','column'
```

3. 配置Flume。
   i. 进入目录*/etc/ecm/flume-conf*。

```
cd /etc/ecm/flume-conf
```

   ii. 执行以下命令，创建配置文件*flume.properties*。

```
vim flume.properties
```

iii. 添加如下内容。

```
a1.sources = source1
a1.sinks = k1
a1.channels = c1
a1.sources.source1.type = org.apache.flume.source.kafka.KafkaSource
a1.sources.source1.channels = c1
a1.sources.source1.kafka.bootstrap.servers = <kafka-host1:port1,kafka-host2:port2...>
a1.sources.source1.kafka.topics = flume-test
a1.sources.source1.kafka.consumer.group.id = flume-test-group
a1.sinks.k1.type = hbase
a1.sinks.k1.table = flume_test
a1.sinks.k1.columnFamily = column
# Use a channel which buffers events in memory
a1.channels.c1.type = memory
a1.channels.c1.capacity = <100>
a1.channels.c1.transactionCapacity = <100>
# Bind the source and sink to the channel
a1.sources.source1.channels = c1
a1.sinks.k1.channel = c1
```

- a1.sources.source1.kafka.bootstrap.servers：Kafka集群Broker的Host和端口号。
- a1.sinks.k1.table：HBase表名。
- a1.sinks.k1.columnFamily：列簇名。
- a1.channels.c1.capacity：通道中存储的最大事件数。请根据实际环境修改该参数值。
- a1.channels.c1.transactionCapacity：每个事务通道将从源接收或提供给接收器的最大事件数。请根据实际环境修改该参数值。

4. 执行以下命令，启动服务。

```
flume-ng agent --name a1 --conf /etc/ecm/flume-conf --conf-file flume.properties
```

5. 测试数据写入情况。

在Kafka集群使用*kafka-console-producer.sh*生成数据后，在HBase查到数据。

```
=> ["flume_test"]
hbase(main):003:0> scan 'flume_test'
ROW                    COLUMN+CELL
 defaultf2add0ee-5040-4 column=column:pCol, timestamp=1543493834351, value=data
 7dc-b002-f269b679977b
 incRow                column=column:iCol, timestamp=1543493834373, value=\x00\x00\x00\
                       x00\x00\x00\x00\x01
2 row(s) in 0.0310 seconds
```

消费Kerberos Kafka source

消费高安全Kafka集群的数据时，需要完成额外的配置：

- 在Kafka集群配置Kerberos认证，将生成的*test.keytab*文件拷贝至Hadoop集群的*/etc/ecm/flume-conf*路径下，详情请参见兼容MIT Kerberos认证；将Kafka集群的*/etc/ecm/has-conf/krb5.conf*文件拷贝至Hadoop集群的*/etc/ecm/flume-conf*路径下。
- 配置*flume.properties*。

  在*flume.properties*中添加如下配置。

  ```
  a1.sources.source1.kafka.consumer.security.protocol = SASL_PLAINTEXT
  a1.sources.source1.kafka.consumer.sasl.mechanism = GSSAPI
  a1.sources.source1.kafka.consumer.sasl.kerberos.service.name = kafka
  ```

- 配置Kafka client。

- 在*/etc/ecm/flume-conf*下创建文件*flume_jaas.conf*。

```
KafkaClient {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  storeKey=true
  keyTab="/etc/ecm/flume-conf/test.keytab"
  serviceName="kafka"
  principal="test@EMR.${realm}.COM";
};
```

${realm} 需要替换为Kafka集群的Kerberos realm。

${realm}获取方式：在Kafka集群执行命令 `hostname` ，得到形式为 `emr-header-1.cluster-xxx` 的主机名，例如 `emr-header-1.cluster-123456` ，其中数字串123456即为realm。

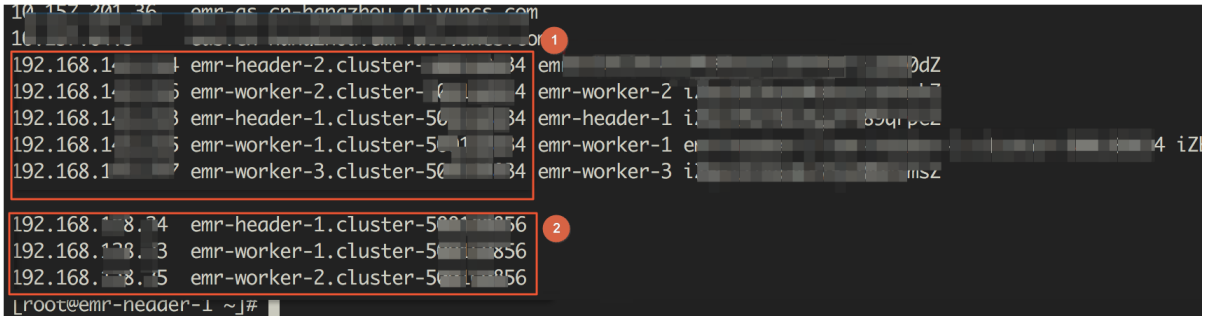
- 修改*/etc/ecm/flume-conf/flume-env.sh*。

初始情况下，*/etc/ecm/flume-conf/*下没有*flume-env.sh*文件，需要拷贝*flume-env.sh.template*并重命名为*flume-env.sh*。在*flume-env.sh*文件末尾添加如下内容。

```
export JAVA_OPTS="$JAVA_OPTS -Djava.security.krb5.conf=/etc/ecm/flume-conf/krb5.conf"
export JAVA_OPTS="$JAVA_OPTS -Djava.security.auth.login.config=/etc/ecm/flume-conf/flume_jaas.conf"
```

- 设置域名。

将Kafka集群各节点的长域名和IP的绑定信息添加到Hadoop集群的*/etc/hosts*文件末尾。长域名的形式为*emr-header-1.cluster-xxxx*。



> **说明** 图中标注①表示的是Hadoop集群的域名；图中标注②表示新增加的Kafka集群域名。

跨域互信使用Flume

在配置了跨域互信后，其他配置如下：

- 在Kafka集群配置Kerberos认证，将生成的keytab文件*test.keytab*拷贝至Hadoop集群的*/etc/ecm/flume-conf*路径下，详情请参见兼容MIT Kerberos认证。
- 配置*flume.properties*。

在*flume.properties*中添加如下配置。

```
a1.sources.source1.kafka.consumer.security.protocol = SASL_PLAINTEXT
a1.sources.source1.kafka.consumer.sasl.mechanism = GSSAPI
a1.sources.source1.kafka.consumer.sasl.kerberos.service.name = kafka
```

- 配置Kafka client。
  - 在*/etc/ecm/flume-conf*下创建文件*flume_jaas.conf*，内容如下。

```
KafkaClient {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  storeKey=true
  keyTab="/etc/ecm/flume-conf/test.keytab"
  serviceName="kafka"
  principal="test@EMR.${realm}.COM";
};
```

${realm}替换为Kafka集群的Kerberos realm。

${realm}获取方式：在Kafka集群执行命令 `hostname` ，得到形式为 `emr-header-1.cluster-xxx` 的主机名，例如 `emr-header-1.cluster-123456` ，其中数字串123456即为realm。

- 修改*/etc/ecm/flume-conf/flume-env.sh*。

  初始情况下，*/etc/ecm/flume-conf/*下没有*flume-env.sh*文件，需要拷贝*flume-env.sh.template*并重命名为*flume-env.sh*。在*flume-env.sh*文件末尾添加如下内容。

  ```
  export JAVA_OPTS="$JAVA_OPTS -Djava.security.auth.login.config=/etc/ecm/flume-conf/flume_jaas.conf"
  ```

6.2.31.4.5. 同步EMR Kafka数据至OSS

E-MapReduce（简称EMR）从EMR-3.16.0版本开始支持Apache Flume。本文介绍如何使用Flume同步EMR Kafka集群的数据至阿里云OSS。

前提条件

- 已开通OSS服务，详情请参见开通OSS服务。
- 已创建Kafka集群，详情请参见创建集群。

> **说明**
> - 如果创建的是Hadoop高安全集群，消费标准Kafka集群的数据，则需要在Hadoop集群配置Kerberos认证，详情请参见兼容MIT Kerberos认证。
> - 如果创建的是Kafka高安全集群，通过Flume将Kafka数据写入Hadoop集群，详情请参见消费Kerberos Kafka source。
> - 如果创建的Hadoop集群和Kafka集群都是高安全集群，需配置跨域互信，详情请参见跨域互信，其它配置详情请参见跨域互信使用Flume。

操作流程

1. 创建OSS路径，详情请参见创建存储空间。

   本文OSS路径为*oss://flume-test/result*。

2. 配置Flume。

   i. 通过SSH方式连接Kafka集群。

   详情请参见登录集群。

   ii. 修改OSS缓存大小或设置JVM最大可用内存（Xmx）。

   Flume向OSS写入数据时，因为需要占用较大的JVM内存，所以可以减小OSS缓存或者增大Flume Agent的Xmx。

   - 修改OSS缓存大小。

     将*hdfs-site.xml*配置文件从*/etc/ecm/hadoop-conf*拷贝至*/etc/ecm/flume-conf*，改小配置项smartdata.cache.buffer.size的值，例如修改为1048576。

   - 修改Xmx。

     在Flume的配置路径*/etc/ecm/flume-conf*下，复制配置文件*flume-env.sh.template*并重命名为*flume-env.sh*，设置Xmx的值，例如设置为1g。

     ```
     export JAVA_OPTS="-Xmx1g"
     ```

iii. 创建配置文件*flume.properties*。

```
a1.sources = source1
a1.sinks = k1
a1.channels = c1
a1.sources.source1.type = org.apache.flume.source.kafka.KafkaSource
a1.sources.source1.channels = c1
a1.sources.source1.kafka.bootstrap.servers = <kafka-host1:port1,kafka-host2:port2...>
a1.sources.source1.kafka.topics = flume-test
a1.sources.source1.kafka.consumer.group.id = flume-test-group
a1.sinks.k1.type = hdfs
a1.sinks.k1.hdfs.path = oss://flume-test/result
a1.sinks.k1.hdfs.fileType=DataStream
# Use a channel which buffers events in memory
a1.channels.c1.type = memory
a1.channels.c1.capacity = <100>
a1.channels.c1.transactionCapacity = <100>
# Bind the source and sink to the channel
a1.sources.source1.channels = c1
a1.sinks.k1.channel = c1
```

- a1.sources.source1.kafka.bootstrap.servers：Kafka集群Broker的Host和端口号。
- a1.sinks.k1.hdfs.path：OSS路径。
- a1.channels.c1.capacity：通道中存储的最大事件数。请根据实际环境修改该参数值。
- a1.channels.c1.transactionCapacity：每个事务通道将从源接收或提供给接收器的最大事件数。请根据实际环境修改该参数值。

3. 启动Flume。
   - 如果配置Flume时修改了OSS缓存大小，需要使用--classpath参数传入OSS相关依赖和配置。

     ```
     flume-ng agent --name a1 --conf /etc/ecm/flume-conf --conf-file flume.properties --classpath "/opt/apps/extra-jars/*:/etc/ecm/flume-conf/hdfs-site.xml"
     ```

   - 如果修改了Flume Agent的Xmx，只需要传入OSS相关依赖。

     ```
     flume-ng agent --name a1 --conf /etc/ecm/flume-conf --conf-file flume.properties --classpath "/opt/apps/extra-jars/*"
     ```

4. 测试数据同步情况。
   i. 通过SSH方式连接Kafka集群，详情请参见登录集群。
   ii. 创建名称为*flume-test*的Topic。

   ```
   /usr/lib/kafka-current/bin/kafka-topics.sh --partitions 10 --replication-factor 2 --zookeeper emr-header-1:2181 /kafka-1.0.0 --topic flume-test --create
   ```

   iii. 生成测试数据。

   ```
   kafka-console-producer.sh --topic flume-test --broker-list emr-header-1:9092
   ```

   例如输入 `abc` 并回车。

   iv. 在OSS的*oss://flume-test/result*路径下会以当前时间的时间戳（毫秒）为后缀生成文件*FlumeData.xxxx*。

6.2.31.4.6. 同步LogHub数据至HDFS

本文介绍如何使用E-MapReduce（简称EMR）的Flume实时同步日志服务（LogHub）的数据至E-MapReduce集群的HDFS，并根据数据记录的时间戳将数据存入HDFS相应的分区中。

背景信息

EMR-3.20.0及后续版本的集群，支持通过Flume同步日志服务数据至E-MapReduce集群。您可以借助日志服务的Logtail工具，将需要同步的数据实时采集并上传到LogHub，再使用E-MapReduce的Flume将LogHub的数据同步至EMR集群的HDFS。

采集数据到日志服务的LogHub的详细步骤参见数据采集概述。

前提条件

创建EMR-3.20.0及后续版本的Hadoop集群，并在可选服务中选择Flume，详情请参见创建集群。

配置Flume

- 配置Source

  以下配置项请遵循开源Flume内容，详情请参见Avro Source和Taildir Source。

参数	说明
type	设置为org.apache.flume.source.loghub.LogHubSource。
endpoint	LogHub的Endpoint。 **说明** 如果使用VPC或经典网络的Endpoint，需要保证与EMR集群在同一个地区；如果使用公网Endpoint，需要保证运行Flume agent的节点有公网IP。
project	LogHub的项目名。
logstore	LogStore名称。
accessKeyId	阿里云的AccessKey ID。
accessKey	阿里云的AccessKey Secret。
useRecordTime	设置为true。 默认值为false。如果Header中没有Timestamp属性，接收Event的时间戳会被加入到Header中。但是在Flume Agent启停或者同步滞后等情况下，会将数据放入错误的时间分区中。为避免这种情况，可以将该值设置为true，使用数据收集到LogHub的时间作为Timestamp。
consumerGroup	消费组名称，默认值为consumer_1。

其他参数说明如下。

参数	说明
consumerPosition	消费组在第一次消费LogHub数据时的位置，默认值为end，即从最近的数据开始消费。 ○ begin：表示从最早的数据开始消费。 ○ special：表示从指定的时间点开始消费。 在配置为special时，需要配置startTime为开始消费的时间点，单位为秒。 首次运行后LogHub服务端会记录消费组的消费点，此时如果想更改 consumerPosition，可以清除LogHub的消费组状态，或者更改配置consumerGroup为新的消费组。
heartbeatInterval	消费组与服务端维持心跳的间隔，单位是毫秒，默认为30000毫秒。
fetchInOrder	相同Key的数据是否按序消费，默认值为false。
batchSize	通用的source batch配置，在一个批处理中写入通道的最大消息数。
batchDurationMillis	通用的source batch配置，在将批处理写入通道之前的最大时间。
backoffSleepIncrement	通用的source sleep配置，表示LogHub没有数据时触发Sleep的初始和增量等待时间。
maxBackoffSleep	通用的source sleep配置，表示LogHub没有数据时触发Sleep的最大等待时间。

- 配置Channel和Sink

  此处使用Memory Channel和HDFS Sink。以下配置项请遵循开源Flume内容，详情请参见Sink和Channel。

- HDFS Sink配置如下。

参数	值
hdfs.path	/tmp/flume-data/loghub/datetime=%y%m%d/hour=%H
hdfs.fileType	DataStream
hdfs.rollInterval	3600
hdfs.round	true
hdfs.roundValue	60
hdfs.roundUnit	minute
hdfs.rollSize	0
hdfs.rollCount	0

- Memory Channel配置如下。

参数	值
capacity	2000
transactionCapacity	2000

运行Flume agent

在阿里云E-Mapreduce控制台页面启动Flume agent，详情请参见同步HDFS Audit日志至HDFS。启动成功后，您可以看到配置的HDFS路径下按照Record Timestamp存储的日志数据。

```
[root@emr-worker-3 ~]# hdfs dfs -ls /tmp/flume-data/loghub/datetime=190430/hour=11
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/apps/ecm/service/hadoop/2.7.2-1.2.8/package/hadoop-2.7.2-1.2.8/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/apps/ecm/service/tez/0.8.4/package/tez-0.8.4/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 1 items
-rw-r--r--   3 root hadoop       1344 2019-04-30 11:18 /tmp/flume-data/loghub/datetime=190430/hour=11/FlumeData.1556594288836.tmp
[root@emr-worker-3 ~]#
```

6.2.31.5. 常见问题

本文汇总了Flume使用时的常见问题。

- Hive日志数量少于实际日志数量？
- 终止Flume进程时出现DeadLock报错，如何处理？
- 如何处理强制退出后，FileChannel偶发性异常？

Hive日志数量少于实际日志数量？

- 问题现象：通过Flume将日志写入Hive，发现Hive中的日志数量少于实际产生的日志数量。
- 解决方法：需要新增配置hdfs.batchSize，添加组件参数详情，请参见管理组件参数。HDFS Sink通过hdfs.batchSize配置来控制每满多少Event，刷新一次到HDFS。如果不配置该参数，默认每100条刷新一次到HDFS，会导致数据刷新不及时。

终止Flume进程时出现DeadLock报错，如何处理？

- 问题现象：脚本调用exit方法终止Flume进程时，偶现DeadLock报错。
- 解决方法：您可以使用命令 `kill -9` 强制退出。

如何处理强制退出后，FileChannel偶发性异常？

- 问题1
  - 问题现象：使用FileChannel作为Channel类型，并在执行 `kill -9` 强制退出后，因获取目录锁失败，导致Flume无法成功重启，具体报错如下。

    ```
    Due to java.io.IOException: Cannot lock data/checkpoints/xxx. The directory is already locked.
    ```

  - 解决方法：强制退出后，重启时需要清理相关目录下的*in_use.lock*文件，否则会出现异常。因此请尽量避免 `kill -9` 操作。
- 问题2

- 问题现象：使用FileChannel作为Channel类型，并在执行 `kill -9` 强制退出后，因DATA目录解析失败，导致Flume无法成功重启，具体报错如下。

  ```
  org.apache.flume.channel.file.CorruptEventException: Could not parse event from data file.
  ```

- 解决方法：强制退出后，重启时需要清理相关目录下的CheckPoint和DATA目录，否则会出现异常。因此请尽量避免 `kill -9` 操作。

6.2.32. Hue

6.2.32.1. 使用说明

本文介绍如何在E-MapReduce上配置及访问Hue，通过使用Hue可以在浏览器端与Hadoop集群进行交互来分析处理数据。

前提条件

- 已设置安全组访问，详情请参见管理安全组。

  > **注意** 设置安全组规则时要针对有限的IP范围。禁止在配置的时候对0.0.0.0/0开放规则。

- 已打开8888端口，详情请参见访问链接与端口。

注意事项

- 当您需要使用Hue的Workflow作业时，请在Hue配置页签，删除**app_blacklist**参数值中的*jobbrowser*。

- 当您需要使用Hue通过界面浏览或者操作HDFS系统的目录时，请在Hue配置页签，删除**app_blacklist**参数值中的*filebrowser*，并启动HDFS服务的HttpFS组件。

访问Hue WebUI

1. 进入集群详情页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
2. 在左侧导航栏中，单击**访问链接与端口**。
3. 在**访问链接与端口**页面，单击Hue服务所在行的链接。

   输入Hue的账户和密码，即可正常的访问Web UI页面。

   > **说明** 第一次登录Hue WebUI时，请参见查看初始密码获取admin的初始密码。

查看初始密码

Hue服务默认在第一次运行时，如果未设置管理员则将第一个登录用户设置为管理员。因此出于安全考虑，E-MapReduce将默认为Hue服务创建一个名为admin的管理员账号，并为其设置一个随机的初始密码。您可以通过以下方式查看该管理员账号的初始密码：

1.
2. 在顶部菜单栏处，根据实际情况选择地域和资源组。
3. 单击上方的**集群管理**页签。
4. 在**集群管理**页面，单击相应集群所在行的**详情**。
5. 单击左侧导航栏中的**集群服务**，在集群服务列表中，选择**Hue**。
6. 单击**配置**页签，找到**admin_pwd**参数，该参数对应的值就是随机密码。

> **注意** admin_pwd仅为admin账号的初始密码，在E-MapReduce控制台上更改该密码不会同步到Hue中。如果需要更改admin账号在Hue中的登录密码，您可以使用该初始密码登录Hue，然后在Hue的用户管理模块中进行修改，或者重置账号密码。

6.2.32.2. 基础使用

6.2.32.2.1. 管理用户

E-MapReduce（简称EMR）的Hue支持创建用户账号和重置账号的密码。本文为您介绍如何创建用户账号和重置账号密码。

前提条件

已创建E-MapReduce的Hadoop集群，详情请参见创建集群。

创建用户

1. 使用SSH方式登录集群，详情请参见登录集群。
2. 执行以下命令，创建新账号。

```
/opt/apps/hue/build/env/bin/hue createsuperuser
```

3. 输入新用户名、电子邮件，然后输入密码，再次输入密码后，按Enter键。

```
[root@emr-header-1 ~]# /opt/apps/hue/build/env/bin/hue createsuperuser
Username (leave blank to use 'root'): emr
Email address: 60[redacted]
Password:
Password (again):
Superuser created successfully.
[root@emr-header-1 ~]#
```

当提示**Superuser created successfully**时，则说明新账号创建成功，即可使用新账号登录Hue。

重置账号密码

1. 使用SSH方式登录集群，详情请参见登录集群。
2. 执行以下命令，查看Hue的路径。

```
ps aux | grep hue
```

回显如下类似信息。

```
[root@emr-header-1 ~]# ps aux | grep hue
root      3831  1.8  0.4 401888 73696 pts/2    S+   16:25   0:01 /opt/apps/hue/build/env/bin/python2.7 hue/build/env/bin/hue shell
root      5949  0.0  0.0 112716   960 pts/1    S+   16:26   0:00 grep --color=auto hue
root      7832  0.0  0.1 369196 19820 ?        Sl   15:13   0:01 /opt/apps/hue/build/env/bin/python2.7 /usr/lib/hue-current/build/env/bin/supervisor
hue       8019  0.1  0.8 2954004 135800 ?      Sl   15:13   0:06 /opt/apps/hue/build/env/bin/python2.7 /opt/apps/hue/build/env/bin/hue runcherrypyserver
root     30881  0.3  0.4 401888 73804 pts/0    S+   16:22   0:00 /opt/apps/hue/build/env/bin/python2.7 /opt/apps/hue/build/env/bin/hue shell
```

> **说明** 本示例中获取到Hue的路径为*/opt/apps/hue/build/env/bin/hue*。

3. 执行以下命令，启动Hue的Shell。

```
/opt/apps/hue/build/env/bin/hue shell
```

> **说明** 代码中的*/opt/apps/hue/build/env/bin/hue*为步骤2中获取的Hue路径。

4. 执行以下命令，重置用户密码。

```
from django.contrib.auth.models import User
user = User.objects.get(username='<your_username>')
user.set_password('<your_new_password>')
user.save()
```

> **说明** 代码中的 `<your_username>` 和 `<your_new_password>` 需要替换为待重置密码的用户名和新密码。您可以按下键盘的*ctrl+D*组合键退出Shell。

代码示例如下。

```
[root@emr-header-1 ~]#  /opt/apps/hue/build/env/bin/hue shell
Python 2.7.5 (default, Aug  7 2019, 00:51:29)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from django.contrib.auth.models import User
>>> user = User.objects.get(username='[redacted]')
>>> user.set_password('[redacted]')
>>> user.save()
>>>
[root@emr-header-1 apps]#
```

重置密码后，即可使用新密码登录Hue 。

6.2.32.2.2. 添加配置

本文为您介绍如何为全局配置文件*hue.ini*添加配置信息。

前提条件

已创建E-MapReduce的Hadoop集群，详情请参见创建集群。

操作步骤

1. 进入详情页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
2. 进入Hue页面。
   i. 在页面左侧导航栏中，选择**集群服务 > Hue**。
   ii. 在Hue页面，单击**配置**页签。
   iii. 在**服务配置**区域，单击**hue**。
3. 在Hue页面，单击右上角的**自定义配置**。
4. 在**新增配置项**对话框中，添加配置项。

   添加的参数形式如下所示。

   ```
   $section_path.$real_key
   ```

   参数信息如下：

   - `$real_key`：需要添加的实际的Key。例如 `hive_server_host` 。
   - `$section_path`：Key的层级，可以通过查看*hue.ini*文件获取，详情请参见hue.ini。

     例如：本文示例中添加的参数为 `hive_server_host` ，通过查看*hue.ini*文件， `hive_server_host` 是属于 `[beeswax]` 下的，则 `$section_path` 应该为 `beeswax` ，所以您应该添加参数为 `beeswax.hive_server_host` ，参数值为 `localhost` 的配置信息。

     ```
     [beeswax]

       # Host where HiveServer2 is running.
       # If Kerberos security is enabled, use fully-qualified domain name (FQDN).
       ## hive_server_host=localhost

     ```

     例如：如果您需要修改*hue.ini*文件中的多级Section下的值，具体如下图所示，则需要配置的参数为 `desktop.ldap.users.user_name_attr` 。

     ```
     [desktop]
       [[ldap]]
         [[[users]]]
           user_name_attr
     ```

   添加完成后，单击**确定**。

5. 保存配置。
   i. 单击右上角的**保存**。
   ii. 在**确认修改**对话框中，输入**执行原因**，单击**确定**。
6. 重启Hue。
   i. 单击右上角的**操作 > 重启Hue**。
   ii. 在**执行集群操作**对话框中，输入**执行原因**，单击**确定**。
   iii. 在**确认**对话框，单击**确定**。

6.2.32.2.3. 调整YARN队列

Hue进行SQL交互查询时，需要向YARN申请资源进行计算，如果需要对计算资源进行管理和隔离，则需要配置HiveSQL和SparkSQL的队列。本文为您介绍如何调整YARN队列。

前提条件

已创建E-MapReduce的Hadoop集群，详情请参见创建集群。

操作步骤

1. 进入详情页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
2. 修改或添加自定义配置。

   > **说明** 如果需要修改配置，您可以**配置**页面，搜索待修改的配置项，直接修改配置项的值。

   - HiveSQL需要根据不同引擎设置HiveServer2：
     a. 在左侧导航栏中，选择**集群服务 > Hive**。
     b. 在Hive页面，单击上方的**配置**页签。
     c. 在**服务配置**区域，单击**hiveserver2-site**页签。
     d. 在**hiveserver2-site**页签，单击右上角的**自定义配置**，添加配置信息。

引擎	参数	描述
Hive on MR	mapreduce.job.queuename	队列名称，您可以自定义。 本文示例参数值为QUEUENAME。
Hive on Tez	tez.queue.name	
Hive on Spark	spark.yarn.queue	

     e. 添加完配置信息后，单击**确定**。
     f. 保存配置。
        a. 单击右上角的**保存**。
        b. 在**确认修改**对话框中，输入执行原因，单击**确定**。
   - SparkSQL使用SparkThriftServer，在Spark组件上修改spark-thriftServer配置或添加自定义配置：
     a. 在左侧导航栏中，选择**集群服务 > Spark**。
     b. 单击**配置**页签。
     c. 单击**服务配置**区域的**spark-thriftServer**页签。
     d. 在**spark-thriftServer**页签，单击右上角的**自定义配置**。
     e. 在**新增配置项**对话框中，添加参数为spark.yarn.queue，参数值为QUEUENAME的信息，单击**确定**。
     f. 保存配置。
        a. 单击右上角的**保存**。
        b. 在**确认修改**对话框中，输入执行原因，单击**确定**。
3. 重启服务。
   - 重启Hive的HiveServer2组件。
     a. 在左侧导航栏中，选择**集群服务 > Hive**。

b. 在**组件列表**区域，单击**HiveServer2**所在行的**重启**。
c. 在**执行集群操作**对话框，输入**执行原因**，单击**确定**。
d. 在**确认**对话框中，单击**确定**。

- 重启Spark的ThriftServer组件。
  a. 在左侧导航栏中，选择**集群服务 > Spark**。
  b. 在**组件列表**区域，单击**ThriftServer**所在行的**重启**。
  c. 在**执行集群操作**对话框，输入**执行原因**，单击**确定**。
  d. 在**确认**对话框中，单击**确定**。

6.2.32.3. 高阶使用

6.2.32.3.1. Hue对接LDAP

当您使用LDAP管理用户账号，访问Hue时需进行LDAP相关的配置。本文以Hue对接E-MapReduce自带的OpenLDAP为例，介绍如何配置Hue后端对接LDAP，并通过LDAP进行身份验证。自建的LDAP请您根据实际情况修改参数。

操作步骤

1. 进入服务配置。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏中，单击**集群服务 > Hue**。
   vi. 单击**配置**页签。
   vii. 在**服务配置**区域，单击**hue**。

2. 修改**backend**的值为**desktop.auth.backend.LdapBackend**。
3. 增加自定义配置。

i. 单击右上角的**自定义配置**，添加如下配置项。

> **注意** 增加下表的自定义配置时，只有**desktop.ldap.bind_password**参数需要在E-MapReduce控制台获取，其余参数请按表格中给出的参数值填写。

配置项	描述	示例值
desktop.ldap.ldap_url	LDAP服务器的URL。	ldap://emr-header-1:10389
desktop.ldap.bind_dn	绑定的管理员用户dn，该dn用于连接到LDAP或AD以搜索用户和用户组信息。如果LDAP服务器支持匿名绑定，则此项可不设置。	uid=admin,o=emr
desktop.ldap.bind_password	绑定的管理员用户dn的密码。 **说明** 需要在E-MapReduce控制台上，通过OpenLDAP的服务配置获取**manager_password**的值，即为密码。	无
desktop.ldap.ldap_username_pattern	LDAP用户名dn匹配模式，描述了username如何对应到LDAP中的dn。必须包含**<username>**字符串才能在身份验证期间用于替换。	uid=<username>,ou=people,o=emr
desktop.ldap.base_dn	用于搜索LDAP用户名及用户组的base dn。	ou=people,o=emr
desktop.ldap.search_bind_authentication	是否使用**desktop.ldap.bind_dn**和**desktop.ldap.bind_password**配置中提供的凭据，搜索绑定身份验证连接到LDAP服务器。	false
desktop.ldap.use_start_tls	是否尝试与用*ldap://*指定的LDAP服务器建立TLS连接。	false
desktop.ldap.create_users_on_login	用户尝试以LDAP凭据登录后，是否在Hue中创建用户。	true

ii. 单击**确定**。

4. 保存配置。
   i. 单击右上角的**保存**。
   ii. 开启**自动更新配置**并设置相关信息。
   iii. 单击**确定**。
5. 部署配置。
   i. 单击右上角的**部署客户端配置**。
   ii. 设置相关信息。
   iii. 单击**确定**。
6. 重启Hue。
   i. 在右上角选择**操作 > 重启Hue**。
   ii. 在**执行集群操作**对话框，填写执行原因，单击**确定**。
   iii. 在**确认**对话框，单击**确定**。

   重启Hue成功后，您就可以在Hue中访问开启LDAP认证的引擎了。

后续步骤

> **注意** 对接LDAP之后，原有的管理员账号admin已经不能登录，新的管理员用户为对接LDAP之后第一个登录的用户。

访问Hue，请参见使用说明。

6.2.32.3.2. Hue连接开启LDAP认证的引擎

Hue作为一个数据开发平台，支持连接各种执行引擎，例如Hive、Spark、Impala和Presto。当这些执行引擎开启LDAP认证后，Hue需要进行相应的配置后才能成功连接。本文介绍如何配置Hue连接开启LDAP认证的执行引擎。

前提条件

已创建E-MapReduce的Hadoop集群，详情请参见创建集群。

步骤一：创建LDAP代理认证用户

当执行引擎开启LDAP认证后，Hue访问执行引擎时将会被LDAP认证拦截。您需要创建一个LDAP代理认证用户，Hue使用该用户通过引擎的LDAP认证，并代理登录Hue的用户执行作业。创建Hue代理认证用户的流程如下。

1. 使用SSH方式登录到集群，具体步骤请参见登录集群。
2. 创建*hue.ldif*文件，文件内容如下。

```
dn: uid=hue,ou=people,o=emr
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
userPassword: <password>
sn: hue
cn: hue
uid: hue
```

> **说明** password自定义填写，在步骤二：配置连接开启LDAP认证的引擎需要使用。

3. 创建Hue代理用户。

```
ldapadd -x -H ldap://emr-header-1:10389 -D uid=admin,o=emr -w <admin_pwd> -f hue.ldif
```

<admin_pwd>：LDAP Admin用户的密码。可以在EMR控制台左侧导航栏，选择**集群服务 > OpenLDAP**，在OpenLDAP服务页面，单击**配置**页签，搜索manager_password配置，获取到对应的值。

步骤二：配置连接开启LDAP认证的引擎

1. 进入Hue配置页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在左侧导航栏，选择**集群服务 > Hue**。
   v. 在Hue服务页面，单击**配置**页签。
   vi. 在**服务配置**区域，单击**hue**。
2. 添加Hue自定义配置。
   i. 单击右上角的**自定义配置**。

ii. 在**新增配置项**区域，添加配置信息。请根据具体的执行引擎添加相应的配置信息。

- 连接开启LDAP认证的Hive、Spark配置信息。

参数	描述
beeswax.auth_username	固定值为**hue**。
beeswax.auth_password	为创建代理用户Hue的密码，详细信息，请参见步骤一：创建LDAP代理认证用户。

- 连接开启LDAP认证的Impala配置信息。

参数	描述
impala.auth_username	固定值为**hue**。
impala.auth_password	为创建代理用户Hue的密码，详细信息，请参见步骤一：创建LDAP代理认证用户。

- 连接开启LDAP认证的Presto配置信息。

  添加参数**notebook.interpreters.presto.options**，参数值如下。

```
{"url": "jdbc:presto://<hostname>:7778/hive/default?SSL=true&SSLKeyStorePath=/etc/ecm/presto-conf/keystore&SSLKeyStorePassword=<keystore_pwd>", "driver": "com.facebook.presto.jdbc.PrestoDriver", "has_impersonation": true}
```

  参数值中涉及的需要替换的参数如下表。

参数	描述
hostname	在PrestoMaster所在节点上执行 `hostname` 命令查看。 hostname的形式为emr-header-1.cluster-xxx。
keystore_pwd	在E-MapReduce控制台集群管理页面，选择**集群服务 > Presto**，单击**配置**页签，搜索keystore_password获取到对应的配置信息。 **说明** 如果您在Presto配置页面无法搜索到keystore_password参数，您可以通过SSH登录集群的Header节点，执行 `sed -n 's/http-server.https.keystore.key=\([^;]*\)/\1/p' /etc/ecm/presto-conf/config.properties` 命令获取密码，登录集群信息请参见登录集群。

  在Hue中使用Presto时，需要填写正确的LDAP的用户名和密码，才能成功执行SQL语句。

iii. 配置信息添加完成后，单击**确定**。

3. 保存配置。
   i. 单击右上角的**保存**。
   ii. 在**确认修改**区域，填写执行原因，并开启**自动更新配置**。
   iii. 单击**确定**。
4. 重启Hue。
   i. 单击**查看操作历史**，等待Configure HUE任务执行完成。
   ii. 配置任务完成后，选择**操作 > 重启Hue**。
   iii. 在**执行集群操作**对话框，填写执行原因，单击**确定**。

iv. 在**确认**对话框，单击**确定**。

重启Hue成功后，您就可以在Hue中访问开启LDAP认证的引擎了。

6.2.32.3.3. 实现Hue多实例负载均衡

阿里云E-MapReduce默认在每个Master节点上部署一个Hue实例，当Hue访问压力过大时，其加载速度会变慢。本文介绍如何通过Gateway集群增加Hue实例数量，并通过阿里云负载均衡（Server Load Balancer）访问Hue，实现Hue多实例负载均衡。

前提条件

已创建E-MapReduce的Hadoop集群，详情请参见创建集群。

> **说明** 如果您创建的集群是EMR-3.28.0和EMR-4.3.0之前版本，请提交工单处理，使其支持多实例部署。

步骤一：创建Gateway

1. 登录阿里云E-MapReduce控制台。
2. 在顶部菜单栏处，根据实际情况选择地域和资源组。
3. 单击上方的**集群管理**页签。
4. 单击右上角的**创建Gateway**。
5. 在**创建Gateway**页面，配置如下参数，其余参数使用默认值。
   i. 设置**集群名称**。
   ii. 开启**挂载公网**开关。
   iii. 设置**登录密码**。
   iv. 在**关联集群**列表中，选择已创建的集群名称。
   v. 勾选**E-MapReduce服务条款**。

   > **说明** 如果该Gateway只用来Hue负载均衡，则**系统盘大小**可以设置为120 GB、**数据盘大小**设置为200 GB，**数量**设置为2。
6. 单击**创建**。

步骤二：添加Hue服务至Gateway集群

1. 在**集群管理**页面，单击已创建的Gateway集群所在行的**详情**。
2. （可选）在**集群基础信息**的**软件信息**区域，查看您Gateway集群的版本。

   如果您创建的集群是EMR-3.28.0和EMR-4.3.0之前版本，请在添加Hue服务前，登录Gateway集群任意节点，执行如下命令。

```
mysql -u root -pEMRroot1234 -hemr-header-1 << EOF
GRANT ALL PRIVILEGES on *.* to 'hue'@'%' WITH GRANT OPTION;
GRANT ALL on hue.* to 'hue'@'%' IDENTIFIED BY 'EMRhue1234';
FLUSH PRIVILEGES;
CREATE DATABASE IF NOT EXISTS hue;
EOF
```

3. 在左侧导航栏，单击**集群管理**。
4. 在**集群与服务管理**页面，单击**添加服务**。
5. 在**添加服务**对话框，勾选**Hue**。
6. 单击**确定**。

步骤三：配置Hue

1. 在**集群管理**页面，单击已创建的Gateway集群所在行的**详情**。
2. 在左侧导航栏，单击**集群服务 > Hue**。
3. 在**配置**页签，设置**more_hue_ports**以添加端口，多个端口时用英文逗号（,）分隔。

> **说明** 每一个端口代表在所有节点上的一个Hue实例，默认端口为8888。例如，Gateway上有m个节点，**more_hue_ports**设置了n个端口，则Gateway上一共有m*（n+1）个Hue实例。

4. 单击右上角的**保存**。
5. 在**确认修改**对话框，配置各项参数。

i. 输入**执行原因**。
ii. 打开**自动更新配置**开关。
iii. 单击**确定**。

6. 重启Hue。
   i. 单击右上角的**操作 > 重启Hue**。
   ii. 在**执行集群操作**对话框，配置各项参数。
   iii. 单击**确定**。
   iv. 在**确认**对话框，单击**确定**。

步骤四：配置负载均衡

1. 创建负载均衡实例，详情请参见创建实例。
2. 在**实例管理**页面，单击已创建实例所在行的**监听配置向导**。
3. 在**协议&监听**页面，完成以下配置。
   i. 选择**HTTP**协议。
   ii. 设置**监听端口**为80。
   iii. 在**高级配置**区域，单击**修改**。
   iv. 打开**开启会话保持**开关，设置**会话保持超时时间**为36000。
   v. 设置**连接空闲超时时间**为60。
   vi. 单击**下一步**。
4. 在**后端服务器**页面，完成以下配置。
   i. 在**选择服务器组**列表中，选择**新建虚拟服务器组**。
   ii. 设置**虚拟服务器组名称**。例如HUE_SLB。
   iii. 单击**添加**。
   iv. 搜索并勾选Gateway节点的ECS ID，单击**下一步**。
   v. 单击**继续添加**，添加步骤三：配置Hue步骤中参数**more_hue_ports**配置的所有端口以及默认的8888端口。
   vi. 单击**添加**。
   vii. 单击**下一步**。
5. 在**健康检查**页签，单击**下一步**。
6. 在**配置审核**页签，单击**提交**。
7. 单击**知道了**。

   返回**实例管理**页面，单击 图标。

   当后端ECS的健康检查状态为**正常**时，表示后端ECS可以正常处理负载均衡转发的请求。

步骤五：访问Hue

在浏览器中输入*<负载均衡服务地址>:<监听端口>*，使用默认账号admin和初始密码登录Hue。

- 服务地址：在负载均衡的**实例管理**页面查看。
- 监听端口：您在步骤四：配置负载均衡中配置的**监听端口**。例如80。

您可以参照以下步骤，查看初始密码：

1. 在E-MapReduce的**集群管理**页面，单击相应集群所在行的**详情**。
2. 在左侧导航栏中，单击**集群服务 > Hue**。
3. 在**配置**页签的**服务配置**区域，查看**admin_pwd**参数，该参数对应的值就是随机密码。

6.2.32.3.4. 管理LDAP认证

服务开启LDAP认证功能后，访问服务时需要提供LDAP身份认证（LDAP用户名和密码），以便于提升服务的安全性。开启LDAP功能对接的LDAP为E-MapReduce自带的OpenLDAP。开启LDAP认证的功能可以方便您使用LDAP认证，避免了复杂的配置过程。本文为您介绍如何一键开启和关闭LDAP认证。

前提条件

已创建Hadoop集群，详情请参见创建集群。

使用限制

EMR-3.34.0及后续版本或EMR-4.8.0及后续版本的Hadoop集群，支持一键开启LDAP认证。

开启LDAP认证

1. 进入Hue页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务 > Hue**。
2. 开启LDAP认证。
   i. 在Hue服务页面，选择右上角的**操作 > 开启LDAP认证**。
   ii. 在**执行集群操作**对话中，单击**确认**。
3. 单击上方的**查看操作历史**。
   直至操作状态显示**成功**。
4. 重启Hue。
   i. 在Hue服务页面，选择右上角的**操作 > 重启Hue**。
   ii. 在**执行集群操作**对话中，输入**执行原因**，单击**确定**。
   iii. 在**确认**对话中，单击**确定**。

访问Hue的Web UI

1. 进入详情页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
2. 在左侧导航栏，单击**访问链接与端口**。
3. 在**公网访问链接**页面，单击**Hue**所在行的链接。
   使用LDAP用户和密码进行登录。

   > 注意 开启LDAP后第一次登录Hue Web UI的用户将成为管理员用户。

关闭LDAP认证

1. 进入Hue页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务 > Hue**。
2. 关闭LDAP认证。
   i. 在Hue服务页面，选择右上角的**操作 > 关闭LDAP认证**。
   ii. 在**执行集群操作**对话中，单击**确认**。
3. 单击上方的**查看操作历史**。
   直至操作状态显示**成功**。
4. 重启Hue。
   i. 在Hue服务页面，选择右上角的**操作 > 重启Hue**。
   ii. 在**执行集群操作**对话中，输入**执行原因**，单击**确定**。
   iii. 在**确认**对话中，单击**确定**。

6.2.32.4. 最佳实践

6.2.32.4.1. 配置Hue访问Presto服务

EMR-3.33.0及后续版本或EMR-4.6.0及后续版本的集群已经默认支持Hue访问E-MapReduce（EMR）集群的Presto服务。本文为您介绍EMR-3.33.0以前版本或EMR-4.6.0以前版本的集群如何通过配置Hue访问Presto服务。

前提条件

- 已创建EMR-3.33.0以前版本或EMR-4.6.0以前版本的集群，并选择了Presto服务，详细信息请参见创建集群。
- 已打开8888端口，详细信息请参见访问链接与端口。

> 注意 设置安全组规则时要针对有限的IP范围。禁止在配置的时候对0.0.0.0/0开放规则。

步骤一：下载Presto的JDBC驱动

1. 使用SSH方式登录到集群，详细信息请参见登录集群。
2. 创建*presto-jdbc*的目录。

```
mkdir -p /opt/apps/presto-jdbc/
```

3. 下载相应版本的JDBC JAR包，并上传至*/opt/apps/presto-jdbc/*目录。
   i. 下载presto-jdbc。

   例如，本文示例创建的是EMR-3.32.0版本的集群，因为Presto版本是338，所以您可以下载338版本的JDBC JAR包。

   下载下图中的JAR包。

   ii. 上传JAR包至*/opt/apps/presto-jdbc/*目录。
4. 执行以下命令，修改文件权限。

```
chmod 644 /opt/apps/presto-jdbc/*
```

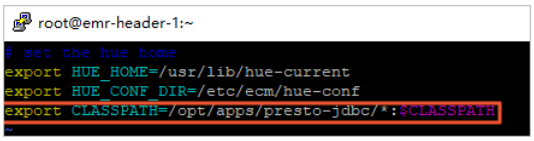
5. 执行以下命令，编辑文件**hue.sh**。

```
vim /etc/profile.d/hue.sh
```

6. 添加以下内容至文件**hue.sh**最后一行。

```
export CLASSPATH=/opt/apps/presto-jdbc/*:$CLASSPATH
```

如下图所示：



步骤二：加载Presto的JDBC驱动

1. 进入Hue配置页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务 > Hue**。
   vi. 在Hue服务页面，单击**配置**页签。
   vii. 在**服务配置**区域，单击**hue**。
2. 添加Hue自定义配置。
   i. 单击右上角的**自定义配置**。
   ii. 在**新增配置项**对话框中，添加配置信息。

参数	描述
notebook.interpreters.presto.name	固定值为Presto。
notebook.interpreters.presto.interface	固定值为jdbc。
notebook.interpreters.presto.options	固定值为{"url": "jdbc:presto://emr-header-1:9090/hive/default", "driver": "com.facebook.presto.jdbc.PrestoDriver", "user": "hadoop", "password": ""}。

   iii. 配置信息添加完成后，单击**确定**。
3. 保存配置。
   i. 单击右上角的**保存**。
   ii. 在**确认修改**对话框，填写执行原因，并开启**自动更新配置**。
   iii. 单击**确定**。
4. 部署配置。
   i. 单击上方的**部署客户端配置**。
   ii. 在**执行集群操作**对话框，选择执行范围和失败处理策略，填写执行原因。
   iii. 单击**确定**。
   iv. 在**确认**对话框，单击**确定**。
      a. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
      b. 在**确认**对话框，单击**确定**。
      c. 单击右上角**查看操作历史**查看任务进度，等待任务完成。
5. 重启Hue。
   i. 在Hue集群服务页面，选择**操作 > 重启Hue**。
   ii. 在**执行集群操作**对话框，选择执行范围和失败处理策略，填写执行原因。
   iii. 单击**确定**。

iv. 在**确认**对话框，单击**确定**。

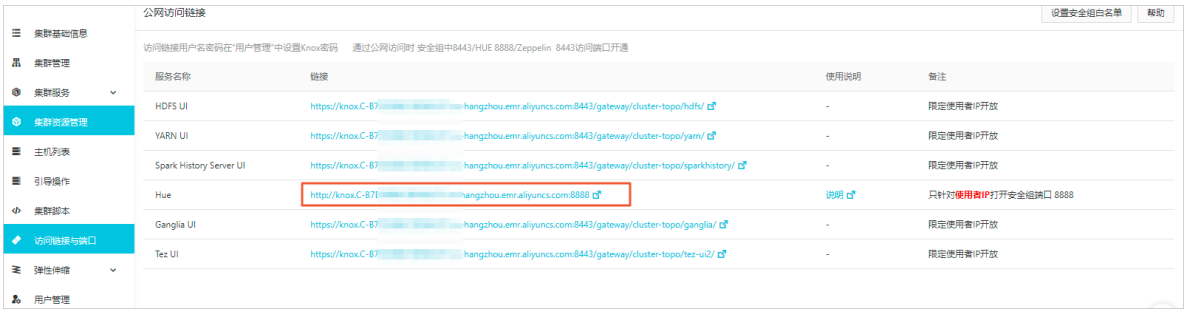
a. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。

b. 在**确认**对话框，单击**确定**。

c. 单击右上角**查看操作历史**查看任务进度，等待任务完成。

步骤三：通过Hue访问EMR集群的Presto服务

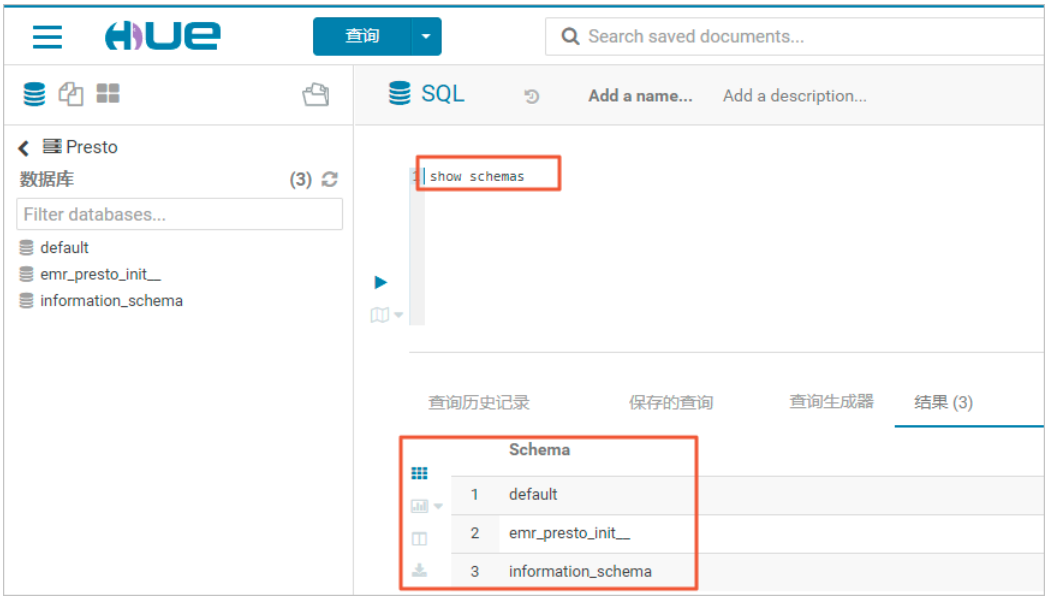
1. 在左侧导航栏，单击**访问链接与端口**。
2. 单击Hue所在行的链接。



使用默认管理员admin登录，初始密码请参见查看初始密码。

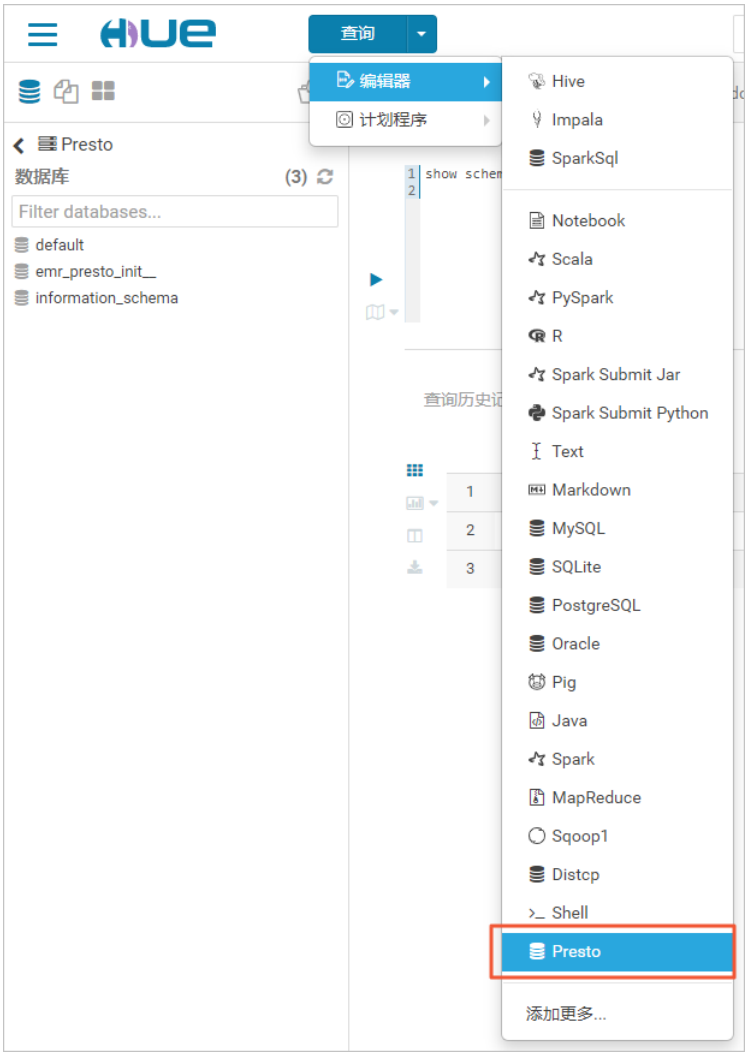
3. 在Hue页面已显示Presto。

表示您可以通过Hue访问EMR集群的Presto服务。



4. （可选）测试连通性。

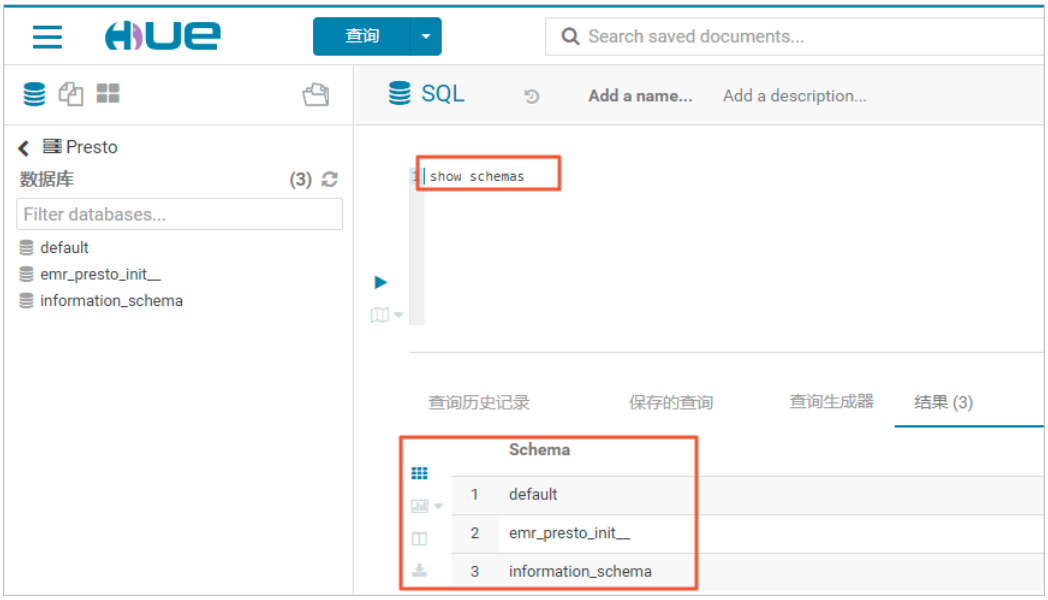
i. 选择**查询 > 编辑器 > Presto**。



ii. 输入以下命令，查询集群上Schema的列表。

```
show schemas
```

返回如下信息：



6.2.32.4.2. 在Hue WebUI使用HBase服务

当您需要使用图形化界面在集群中创建或查询HBase表格时，可以通过Hue实现。本文以EMR-4.9.0版本为例，为您介绍如何通过Hue WebUI创建或删除HBase表格。

前提条件

- 已创建集群，并选择了HBase服务，详细信息请参见创建集群。
- 已打开8888端口，详细信息请参见访问链接与端口。

> 注意 设置安全组规则时要针对有限的IP范围。禁止在配置的时候对0.0.0.0/0开放规则。

使用限制

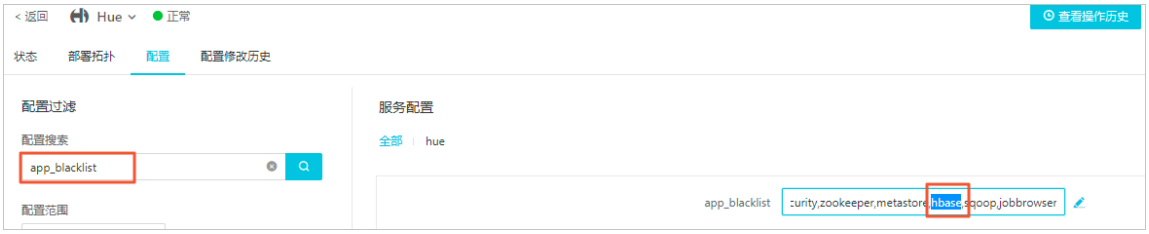
集群需要添加HBase服务。

注意事项

EMR-3.35.0及后续版本或EMR-4.9.0及后续版本的Hadoop集群，需要您在Hue配置页签，删除**app_blacklist**参数值中的*hbase*。

进入HBase Browser

1. 进入详情页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
2. 修改配置。
   i. 在左侧导航栏中，选择**集群服务 > Hue**。
   ii. 在**配置搜索**区域，搜索**app_blacklist**参数。
   iii. 删除**app_blacklist**参数值中的**hbase**。



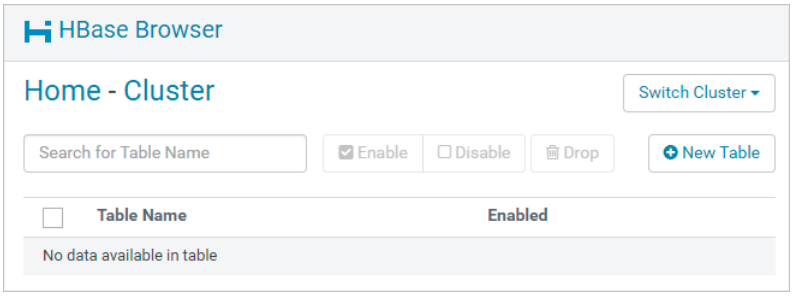
   iv. 单击右上角的**保存**。
   v. 在**确认修改**对话框中，输入**执行原因**，单击**确定**。
3. 重启Hue。
   i. 在右上角，选择**操作 > 重启Hue**。
   ii. 在**执行集群操作**对话框，输入**执行原因**，单击**确定**。
   iii. 在**确认**对话框，单击**确定**。
4. 在左侧导航栏中，单击**访问链接与端口**。
5. 在**访问链接与端口**页面，单击Hue服务所在行的链接。

   输入Hue的账户和密码，即可正常的访问Web UI页面。

   > 说明 初次登录Hue WebUI时，admin账号和密码的获取方法，请参见查看初始密码。

6. 在Hue的左侧导航栏，单击图标。

   进入HBase Browser页面。



创建HBase表格

1. 在HBase Browser页面，单击右上角的**New Table**。
2. 在**Create New Table**对话框中，填写配置信息。

参数	描述
Table Name	表格名称。
Column Families	列族参数。

3. 单击**Submit**。

删除HBase表格

1. 在HBase Browser页面，选中一个或多个表格。
2. 单击上方的**Drop**。
3. 在**Confirm Delete**对话框中，单击**Confirm**。

6.2.32.4.3. 在Hue WebUI使用文件浏览器

当您需要使用图形化界面管理HDFS中的文件时，可以通过Hue实现。本文以EMR-4.9.0版本为例，为您介绍如何通过Hue查看和操作HDFS中的文件和文件夹。

前提条件

- 已设置安全组访问，详情请参见管理安全组。

  > 注意 设置安全组规则时要针对有限的IP范围。禁止在配置的时候对0.0.0.0/0开放规则。

- 已打开8888端口，详情请参见访问链接与端口。

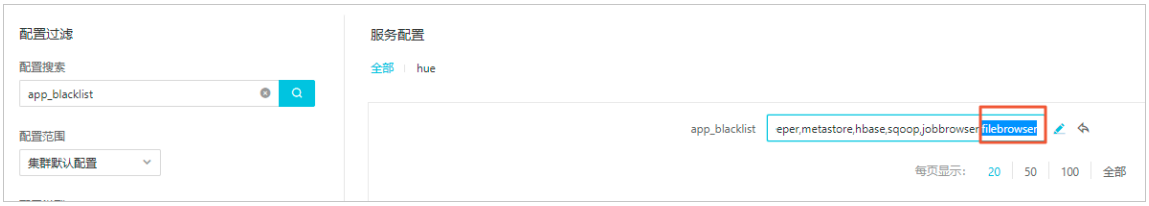
注意事项

EMR-3.35.0及后续版本或EMR-4.9.0及后续版本的Hadoop集群，需要您在Hue配置页签，删除**app_blacklist**参数值中的*filebrowser*，并启动HDFS服务的HttpFS组件。

访问文件浏览器

1. 进入详情页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
2. 修改配置。
   i. 在左侧导航栏中，选择**集群服务 > Hue**。
   ii. 在**配置搜索**区域，搜索**app_blacklist**参数。

iii. 删除app_blacklist参数值中的filebrowser。



iv. 单击右上角的**保存**。

v. 在**确认修改**对话框中，输入**执行原因**，单击**确定**。

3. 重启Hue。

i. 在右上角，选择**操作 > 重启Hue**。

ii. 在**执行集群操作**对话框，输入**执行原因**，单击**确定**。

iii. 在**确认**对话框，单击**确定**。

4. 启动HDFS服务的HttpFS组件。

i. 在左侧导航栏中，选择**集群服务 > HDFS**。

ii. 单击**部署拓扑**页签。

iii. 单击HttpFS组件**操作**列的**启动**。

iv. 在**执行集群操作**对话框，输入**执行原因**，单击**确定**。

v. 在**确认**对话框，单击**确定**。

您可以单击上方的**查看操作历史**，待Start HDFS HttpFS的任务状态为**成功**，表示HttpFS组件成功启动。

5. 在左侧导航栏中，单击**访问链接与端口**。

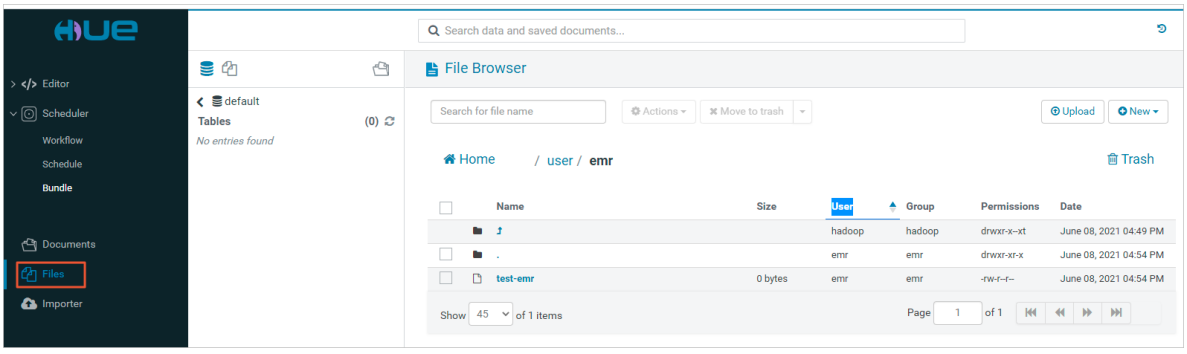
6. 在**访问链接与端口**页面，单击Hue服务所在行的链接。

输入Hue的账户和密码，即可正常的访问Web UI页面。

> **说明** 初次登录Hue WebUI时，admin账号和密码的获取方法，请参见查看初始密码。

7. 在Hue的左侧导航栏，单击 Files 图标。

进入当前登录用户的主目录，显示目录中的子目录或文件信息。

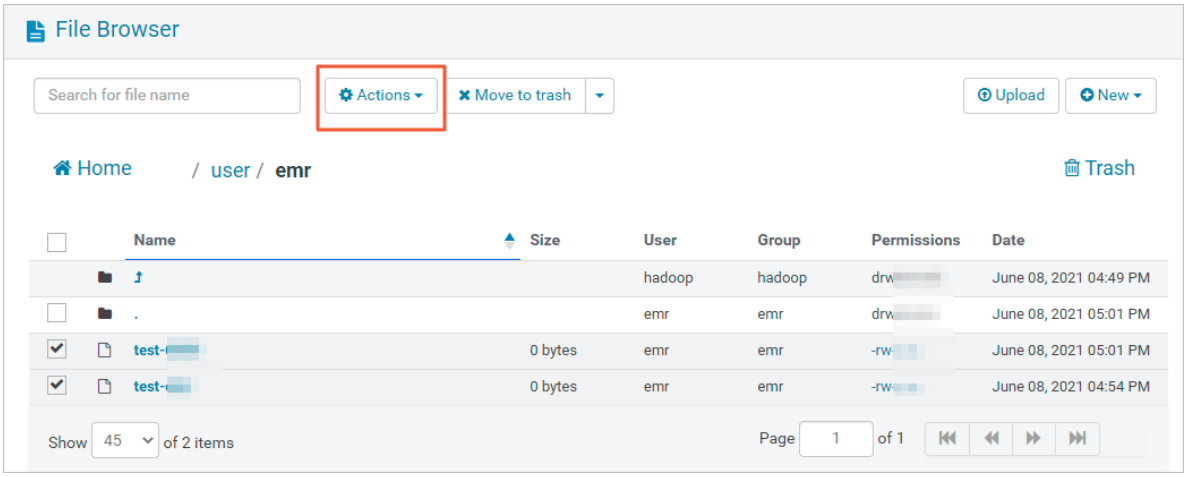


参数	描述
Name	目录或文件的名称。
Size	文件的大小。

参数	描述
User	目录或文件的属主。
Group	目录或文件的属组。
Permissions	目录或文件的权限设置。
Date	目录或文件创建时间。

执行动作

1. 在File Browser页面，选中一个或多个目录或文件。
2. 单击上方的Actions，可以执行以下操作。



操作	描述
Rename	重新命名一个目录或文件。
Move	移动文件，在Move to页面，选择新的目录并单击Move完成移动。
Copy	复制选中的文件或目录。
Download	下载文件至本地。
Change permissions	更改权限，修改选中目录或文件的访问权限。 ○ Read、Write和Execute：可以为属主、属组和其他用户设置Read、Write和Execute权限。 ○ Sticky：禁止HDFS的管理员、目录属主或文件属主以外的用户在目录中移动文件。 ○ Recursive：递归设置权限到子目录。
Summary	摘要，查看选中文件或目录的HDFS存储信息。
Set replication	为选定的文件设置复制因子。
Compress	压缩选定的文件或文件夹。 注意 使用此操作前，需要给EMR集群先添加Oozie服务，添加服务详情请参见添加服务。

上传用户文件

1. 在File Browser页面，单击Upload。
2. 在弹出的对话框中，单击Select files。
3. 选择待上传的文件。

创建新文件或者目录

- 创建新文件
  i. 在File Browser页面，选择New > File。

ii. 在Create File对话框中，输入File Name。

iii. 单击Create。

- 创建目录
  i. 在File Browser页面，选择New > Directory。
  ii. 在Create Directory对话框中，输入Directory Name。
  iii. 单击Create。

6.2.32.4.4. 在Hue WebUI使用编辑器

当您需要使用图形化界面在集群中执行HiveQL或Spark SQL语句时，可以通过Hue完成任务。本文以EMR-4.9.0版本为例，为您介绍如何在Hue WebUI中使用Hive编辑器和Spark SQL编辑器。

前提条件

- 已设置安全组访问，详情请参见管理安全组。

  > 注意 设置安全组规则时要针对有限的IP范围。禁止在配置的时候对0.0.0.0/0开放规则。

- 已打开8888端口，详情请参见访问链接与端口。

使用Spark SQL编辑器

1. 进入详情页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
2. 在左侧导航栏中，单击**访问链接与端口**。
3. 在**访问链接与端口**页面，单击Hue服务所在行的链接。
   输入Hue的账户和密码，即可正常的访问Web UI页面。

   > 说明 Hue账号和密码的获取方法，请参见使用说明。

4. 进入SparkSql编辑器。
   i. 在Hue的左侧导航栏，单击</>图标。
   ii. 选择**Sparksql**。
   即可进入SparkSql编辑器。
5. 执行SparkSql语句。
   i. 在SparkSql语句编辑区输入以下语句，创建表格。

```
CREATE TABLE IF NOT EXISTS `store_sales`(
  `ss_sold_date_sk` bigint,
  `ss_sold_time_sk` bigint,
  `ss_item_sk` bigint);
```

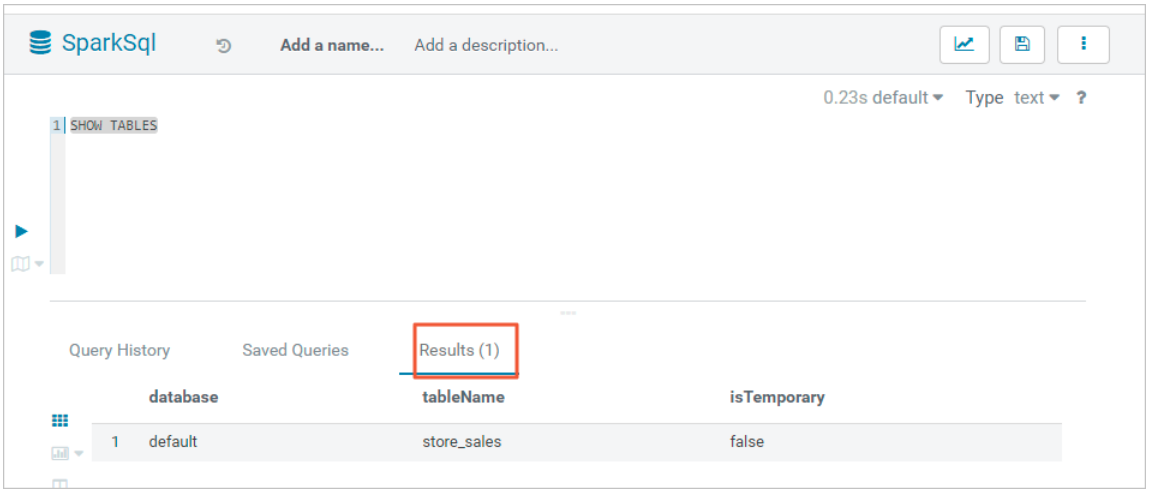
ii. 单击▶图标，开始执行SparkSql语句。

返回如下图所示信息，表示表格创建成功。

iii. 在SparkSql语句编辑区输入以下语句，查询表格。

```
SHOW TABLES
```

返回如下图所示信息，可以查看已创建的表格。



使用Hive编辑器

1. 进入详情页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
2. 在左侧导航栏中，单击**访问链接与端口**。
3. 在**访问链接与端口**页面，单击Hue服务所在行的链接。
   输入Hue的账户和密码，即可正常的访问Web UI页面。

   > **说明** Hue账号和密码的获取方法，请参见使用说明。

4. 进入Hive编辑器。
   i. 在Hue的左侧导航栏，单击</>图标。

ii. 选择**Hive**。

即可进入Hive编辑器。

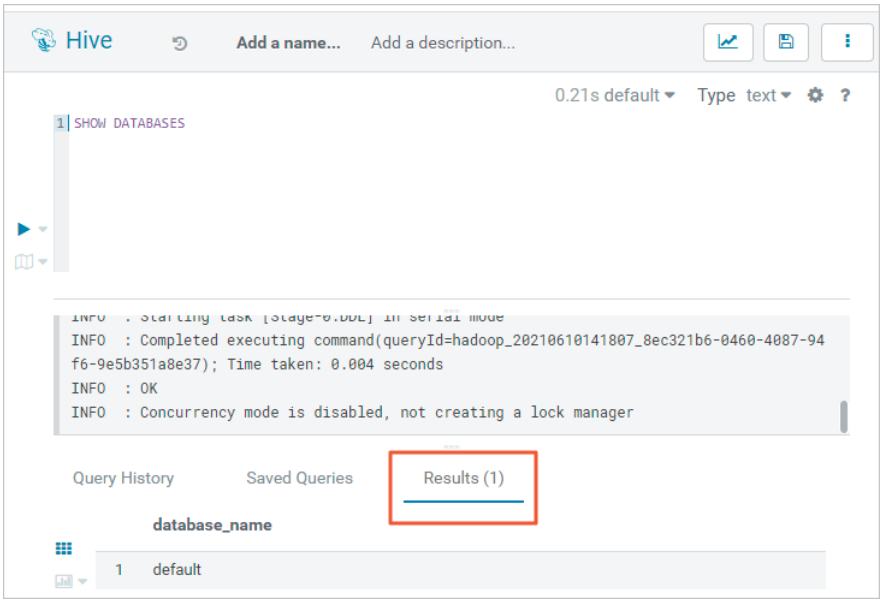
5. 执行Hive语句。

i. 在Hive语句编辑区输入以下语句，查询集群上的数据库列表。

```
SHOW DATABASES
```

ii. 单击 ▶ 图标，开始执行Hive语句。

可以查看已有的数据库列表,返回如下图所示信息。

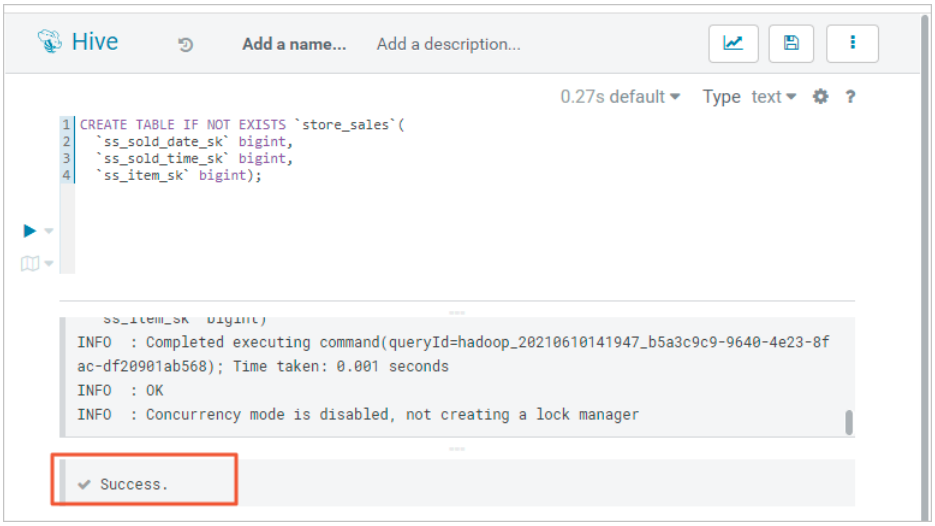


iii. 在Hive语句编辑区输入以下语句，创建表格。

```
CREATE TABLE IF NOT EXISTS `store_sales`(
  `ss_sold_date_sk` bigint,
  `ss_sold_time_sk` bigint,
  `ss_item_sk` bigint);
```

iv. 单击 ▶ 图标，开始执行Hive语句。

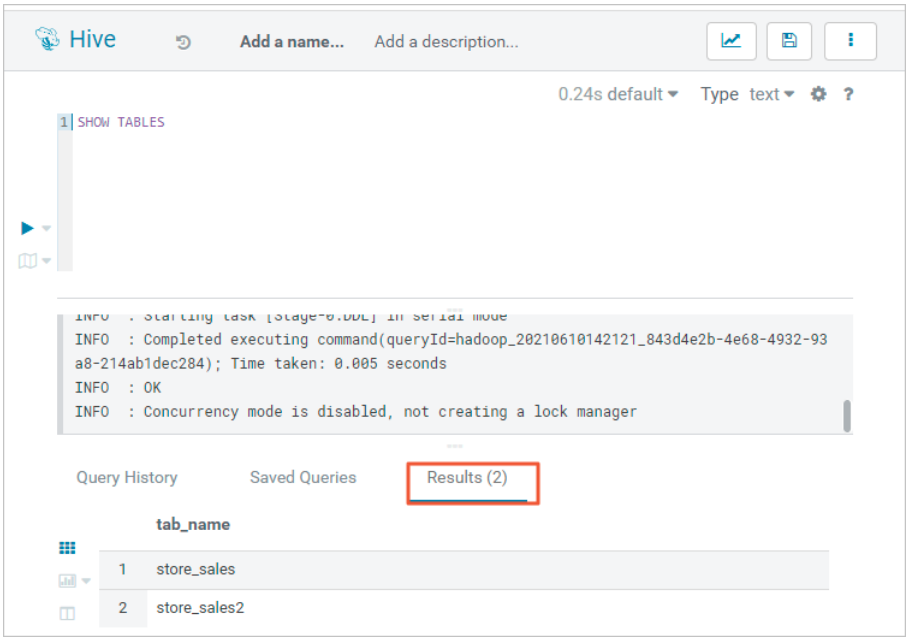
返回如下图所示信息，表示表格创建成功。



v. 在Hive语句编辑区输入以下语句，查询表格。

```
SHOW TABLES
```

可以查看已创建的表格，返回如下图所示信息。

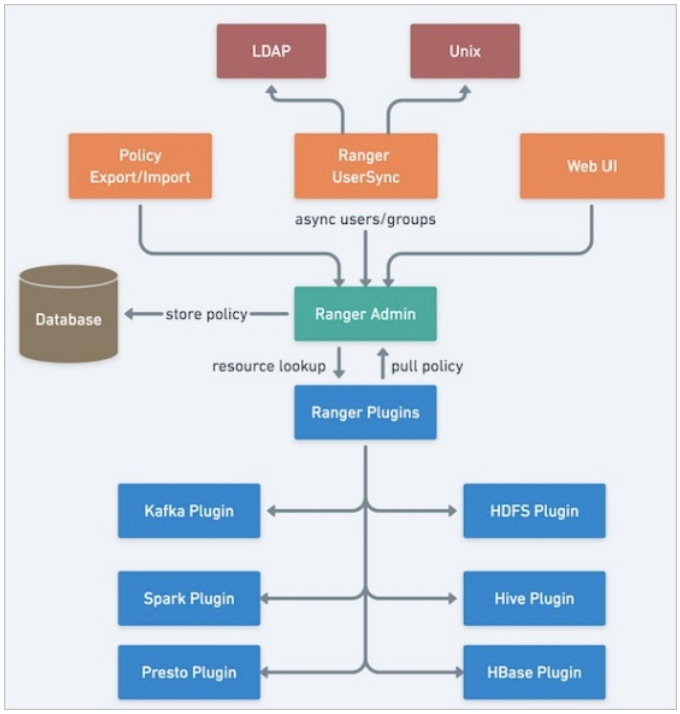


6.2.33. Ranger

6.2.33.1. 概述

Apache Ranger提供集中式的权限管理框架，可以对Hadoop生态中的HDFS、Hive和YARN等组件进行细粒度的权限访问控制，并且提供了Web UI方便管理员操作。

介绍



Ranger主要由三个组件组成：

- Ranger Admin

您可以创建和更新安全访问策略，这些策略被存储在数据库中。各个组件的Plugin定期对这些策略进行轮询。

- Ranger Plugins

  Plugin嵌入在各个集群组件的进程里，是一个轻量级的Java程序。例如，Ranger对Hive的组件，就被嵌入在Hiveserver2里。这些Plugin从Ranger Admin服务端拉取策略，并把它们存储在本地文件中。当接收到来自组件的用户请求时，对应组件的Plugin会拦截该请求，并根据安全策略对其进行评估。

- Ranger UserSync

  Ranger提供了一个用户同步工具。您可以从Unix或者LDAP中拉取用户和用户组的信息。这些用户和用户组的信息被存储在Ranger Admin的数据库中，可以在定义策略时使用。

创建集群

- 新建集群时，在E-MapReduce控制台可选服务区域，直接勾选Ranger组件。

- 已有集群时，可以在**集群管理**页面添加Ranger服务。

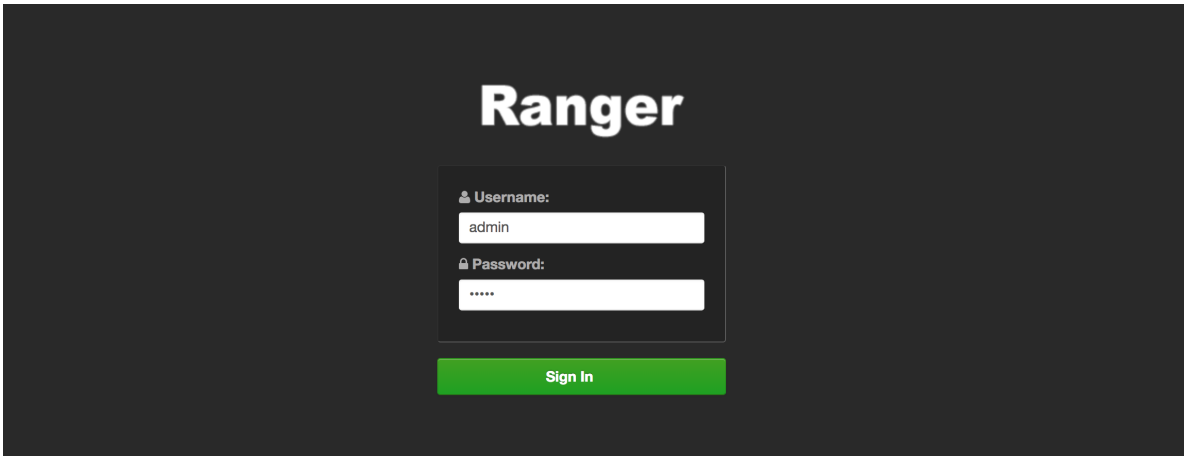
> **说明** Ranger可以设置集群中的Linux用户和LDAP用户的用户策略。

访问Ranger UI

在访问Ranger UI之前，需要确认已设置安全组，即Hadoop集群允许您通过当前网络访问该集群。具体详情请参见访问链接与端口。

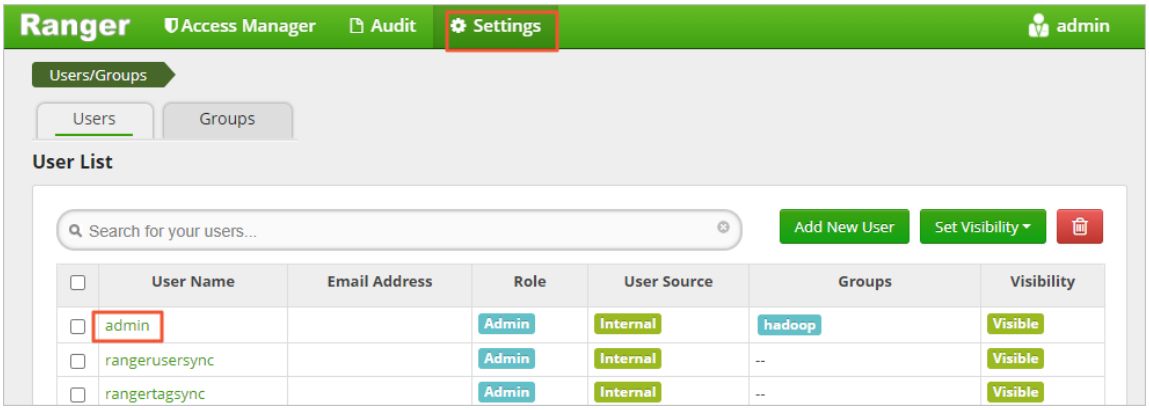
1. 进入集群详情页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
2. 在左侧导航栏中，选择**访问链接与端口**。
3. 单击**RANGER UI**所在行的链接。

   您可以通过链接访问Ranger UI。
4. 在Ranger UI登录界面，输入用户名和密码（默认的用户名和密码均为admin）。

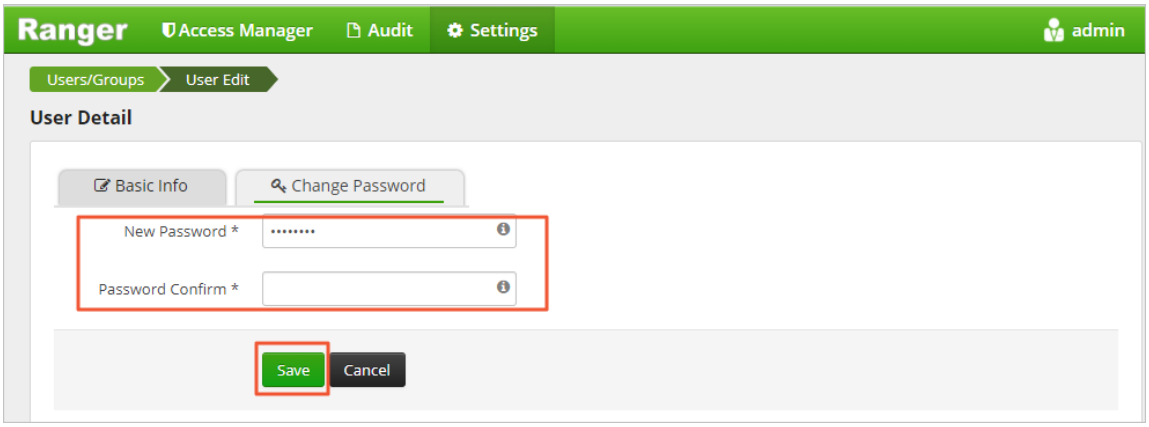


说明 如果默认密码被修改无法登录时，请参见常见问题。

5. 首次登录后，需要修改密码。
   i. 单击上方的Settings。



   ii. 修改admin的密码。



   iii. 单击右上角的admin > Log Out 。
   使用新的密码登录即可。

组件集成Ranger

通过插件的方式，Ranger与集群中的开源组件进行集成。通过Ranger可以对组件进行细粒度的访问权限控制。

您可以将集群中的组件集成到Ranger，以控制相关权限，详情请参见以下配置：

- HDFS配置
- HBase配置
- Hive配置

- Spark配置
- Presto配置
- Impala配置
- YARN配置
- JindoFS配置
- JindoFS OSS配置

用户管理

您可以使用Ranger对用户或用户组进行权限管理。

用户或用户组可以来自本地Unix系统或者LDAP，推荐使用LDAP：

- Ranger Admin集成LDAP
- Ranger Usersync集成LDAP

常见问题

Q：默认密码无法登录Ranger UI时，如何处理？

A：如果默认密码被修改，您可以按照如下方法处理：

1. 登录集群的Master节点，详情请参见登录集群。
2. 执行如下命令重置admin的密码。

```
mysql -urangeradmin -prangeradmin
update ranger.x_portal_user set password="ceb4f32325eda6142bd65215f4c0f371" where login_id="admin";
```

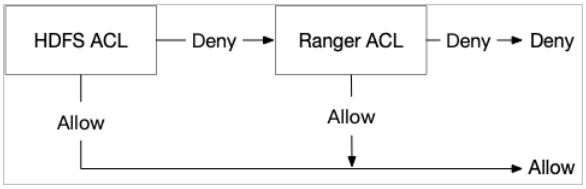
6.2.33.2. 组件集成

6.2.33.2.1. HDFS配置

本文介绍如何将HDFS集成到Ranger，以及如何配置权限。

背景信息

Ranger HDFS权限控制与HDFS ACL共同生效，且优先级低于HDFS ACL。只有在HDFS ACL权限校验被拒绝时才会校验Ranger HDFS权限。鉴权流程如下图所示。



前提条件

已在E-MapReduce上创建集群，并选择了Ranger服务，详情请参见创建集群。

HDFS集成Ranger

1. 进入集群详情页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
2. Ranger启用HDFS。
   i. 在左侧导航栏选择**集群服务 > RANGER**。

ii. 单击右侧的**操作**下拉菜单，选择**启用HDFS**。

iii. 执行集群操作。

a. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。

b. 在**确认**对话框，单击**确定**。

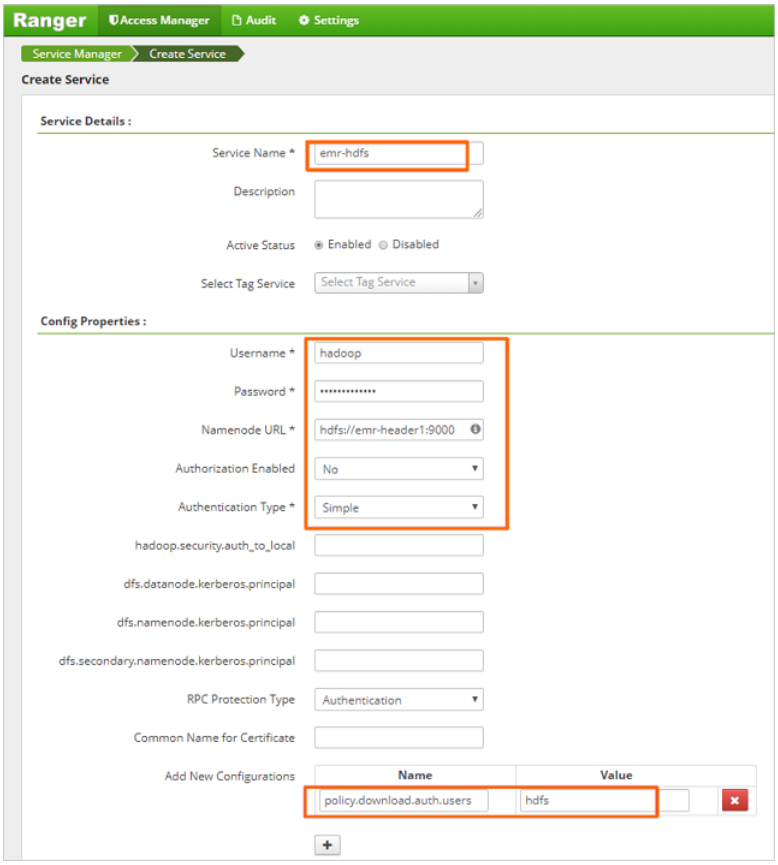
c. 单击右上角**查看操作历史**查看任务进度，等待任务完成。

3. Ranger UI添加HDFS Service。

i. 进入Ranger UI页面，详情请参见概述。

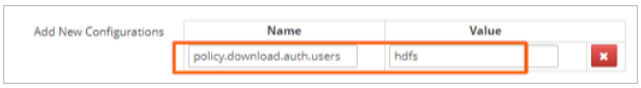
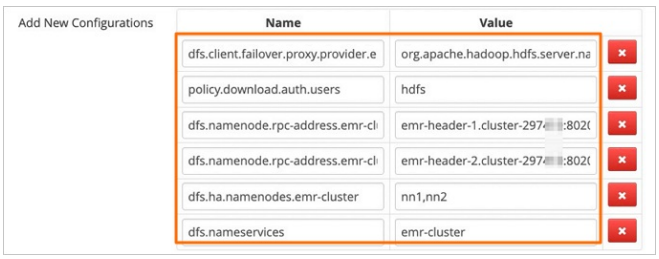
ii. 在Ranger UI页面，单击HDFS所在行的 图标，添加HDFS Service。

iii. 配置相关参数。



参数	描述
Service Name	固定值emr-hdfs。
Username	固定值hadoop。
Password	自定义。
Namenode URL	■ 非高可用集群：*hdfs://emr-header-1:9000*。 ■ 高可用集群：*hdfs://emr-cluster*。
Authorization Enabled	普通集群选择**No**；高安全集群选择**Yes**。
Authentication Type	■ Simple：表示普通集群。 ■ Kerberos：表示高安全集群。

dfs.datanode.kerberos.principal
dfs.namenode.kerberos.principal
dfs.secondary.namenode.kerberos.principal
	普通集群时不填写；高安全集群时填写hdfs/_HOST@${REALM}。
	说明 ${REALM}为KDC的Realm值，EMR高安全集群的Realm可以通过查看任意节点下*/etc/krb5.conf*文件的kdc_realm配置获取。
	例如，查看到的kdc_realm配置如下图。
	[libdefaults]
	kdc_realm = EMR.298[redacted].COM
	default_realm = EMR.298[redacted].COM
	udp_preference_limit = 4096
	kdc_tcp_port = 88
	kdc_udp_port = 88
	dns_lookup_kdc=false
	
	[realms]
	EMR.298[redacted].COM = {
	kdc = 192.168.[redacted]:88
	}

参数	描述
Add New Configurations	需要添加参数**policy.download.auth.users**，参数值为hdfs。  如果您的集群为高可用集群，还需额外添加以下配置： ■ **dfs.nameservices**：固定值emr-cluster。 ■ **dfs.ha.namenodes.emr-cluster** ■ **dfs.namenode.rpc-address.emr-cluster.nn1** ■ **dfs.namenode.rpc-address.emr-cluster.nn2** ■ **dfs.client.failover.proxy.provider.emr-cluster**  参数值请根据实际环境信息填写。您可以在EMR控制台HDFS服务的配置页面获取。 

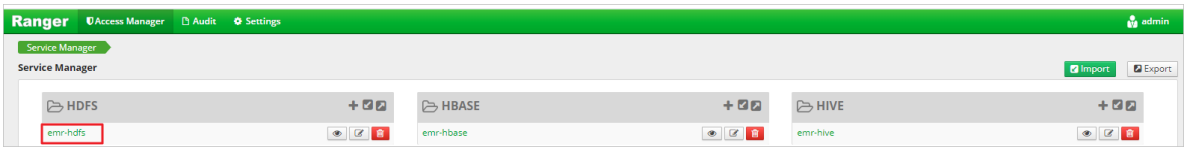
iv. 单击**Add**。

4. 重启HDFS。
   i. 左侧导航栏单击**集群服务 > HDFS**。
   ii. 单击右上角**操作**下拉菜单，选择**重启NameNode**。
   iii. 执行集群操作。
      a. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
      b. 在**确认**对话框，单击**确定**。
      c. 单击右上角**查看操作历史**查看任务进度，等待任务完成。

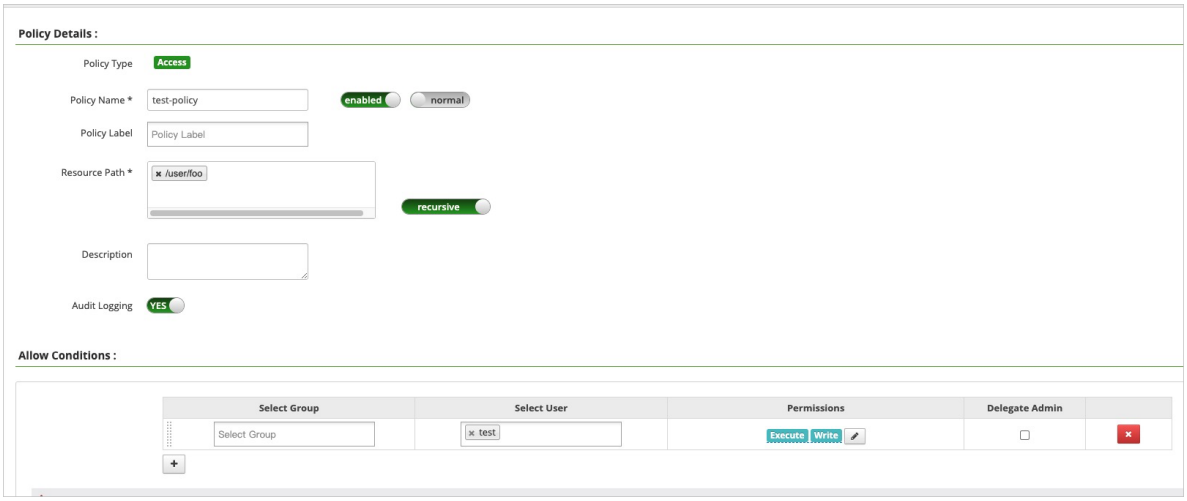
权限配置示例

例如，授予test用户*/user/foo*路径的Write和Execute权限。

1. 进入Ranger UI页面，详情请参见概述。
2. 在Ranger UI页面，单击配置好的**emr-hdfs**。



3. 单击右上角的**Add New Policy**。
4. 配置相关参数。



参数	描述
Policy Name	策略名称，可以自定义。
Resource Path	资源路径。例如，/user/foo。
recursive	子目录或文件是否集成权限。
Select Group	指定添加此策略的用户组。
Select User	指定添加此策略的用户。例如，test。
Permissions	选择授予的权限。例如，Write和Execute。

5. 单击**Add**。

   添加Policy后，实现了对test用户的授权。test用户对HDFS路径*/user/foo*拥有了Write和Execute权限。

   > **说明** 添加、删除或修改Policy后，需要等待约一分钟至授权生效。

6.2.33.2.2. HBase配置

本文介绍如何将HBase集成到Ranger，以及如何配置权限。

前提条件

已创建集群，并选择了HBase和Ranger服务，详情请参见创建集群。

HBase集成Ranger

1. Ranger启用HBase。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏选择**集群服务 > RANGER**。

vi. 单击右侧的**操作**下拉菜单，选择**启用HBase**。

vii. 执行集群操作。

a. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。

b. 在**确认**对话框，单击**确定**。

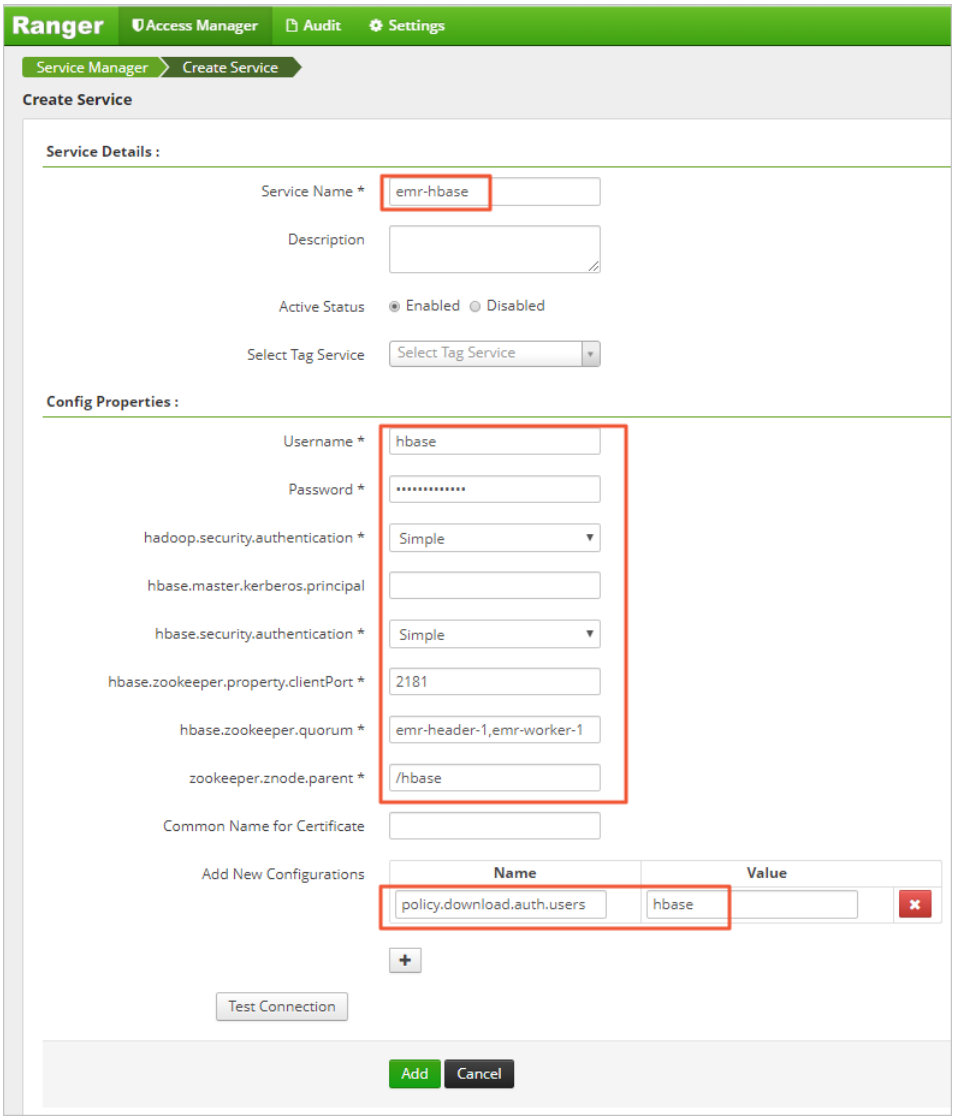
c. 单击右上角**查看操作历史**查看任务进度，等待任务完成。

2. Ranger UI添加HBase Service。

i. 进入Ranger UI页面，详情请参见概述。

ii. 在Ranger UI页面，添加HBase Service。

iii. 配置相关参数。



参数	说明
Service Name	固定值emr-hbase。
Username	固定填写hbase。
Password	自定义。
hadoop.security.authentication	■ 标准集群（非高安全集群）：选择Simple。 ■ 高安全集群：选择Kerberos。
hbase.master.kerberos.principal	标准集群时不填写；高安全集群时填写hbase/_HOST@EMR.${id}.COM。 说明 ${id} 可登录机器执行 hostname 命令，hostname 中的数字即为${id}的值。
hbase.security.authentication	■ 标准集群（非高安全集群）：选择Simple。 ■ 高安全集群：选择Kerberos。
hbase.zookeeper.property.clientPort	固定值2181。
hbase.zookeeper.quorum	固定值emr-header-1,emr-worker-1。

参数	说明
zookeeper.znode.parent	固定值/hbase。
Add New Configurations	■ Name：固定值policy.download.auth.users。 ■ Value：固定值hbase。

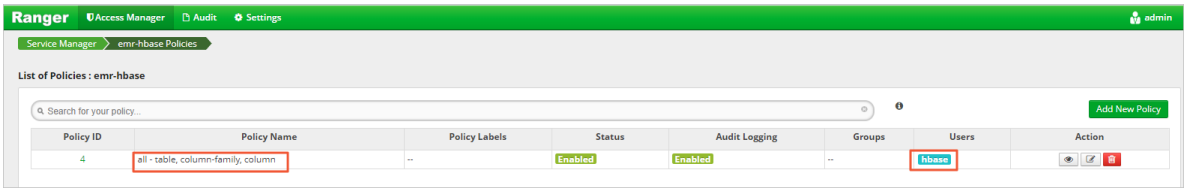
iv. 单击**Add**。

3. 重启HBase。
   i. 左侧导航栏单击**集群服务 > HBase**。
   ii. 单击右上角**操作**下拉菜单，选择**重启All Components**。
   iii. 执行集群操作。
      a. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
      b. 在**确认**对话框，单击**确定**。
      c. 单击右上角**查看操作历史**查看任务进度，等待任务完成。

设置管理员账号

1. 在Service Manager页面，单击创建好的emr-hbase。

2. 设置管理员账号的权限（admin权限）。
   用于执行管理命令，例如balance、compaction、flush或split等。
   因为当前服务已经存在权限策略，所以您只需要单击右侧的图标，在User中添加需要设置的账号即可。另外也可以修改其中的权限（例如只保留admin权限）。HBase账号必须默认设置为管理员账号。



如果使用Phoenix，则需在Ranger的HBase中新增如下策略。

参数	说明
HBase Column-family	星号（*）
HBase Column	星号（*）
Select Group	public
Permissions	Read、Write、Create和Admin

权限配置示例

例如给test用户授予表foo_ns:test的Create、Write和Read权限。

1. 进入Ranger UI页面，详情请参见概述。
2. 在Ranger UI页面，单击配置好的**emr-hbase**。

3. 单击右上角的**Add New Policy**。

4. 配置相关权限。

参数	说明
Policy Name	策略名称，可以自定义。
HBase Table	表对象，格式为**${namespace}:${tablename}**。可输入多个，填写一个需按一次Enter键。 如果是default的namespace，不需要加default。支持通配符星号（*），例如，**foo_ns:***表示foo_ns下的所有表。 **说明** 目前不支持**default:***。
HBase Column-family	列簇。
HBase Column	列名。
Select Group	指定添加此策略的用户组。
Select User	指定添加此策略的用户。例如，test。
Permissions	选择授予的权限。

5. 单击**Add**。

   添加Policy后，实现对test用户的授权。test用户即可以对foo_ns:test表进行访问。

   说明 添加、删除或修改Policy后，需要等待约一分钟至授权生效。

6.2.33.2.3. Hive配置

本文介绍如何将Hive集成到Ranger，以及如何配置权限。

前提条件

已创建集群，并选择了Ranger服务，详情请参见创建集群。

Hive访问模型

访问Hive数据，包括HiveServer2、Hive Client和HDFS三种方式：

- HiveServer2方式
  - 场景： 您可以通过HiveServer2访问Hive数据。
  - 方式：使用Beeline客户端或者JDBC代码通过HiveServer2执行Hive脚本。
  - 权限设置：

    Hive官方自带的Hive授权针对HiveServer2使用场景进行权限控制。

    Ranger中对Hive的表或列级别的权限控制也是针对HiveServer2的使用场景。如果您还可以通过Hive Client或者HDFS访问Hive数据，仅对表或列层面做权限控制还不够，需要选择下面任一方式以进一步控制权限。
- Hive Client方式
  - 场景： 您可以通过Hive Client访问Hive数据。
  - 方式： 使用Hive Client访问。
  - 权限设置

    Hive Client会请求HiveMetaStore进行DDL操作，例如 `Alter Table Add Columns` ，也可以通过提交MapReduce作业读取并处理HDFS中数据。

    Hive官方自带的Hive授权可以针对Hive Client使用场景进行权限控制，它会根据SQL中表的HDFS路径的读写权限，来决定您是否可以进行DDL或DML操作，例如 `ALTER TABLE test ADD COLUMNS(b STRING)` 。

    由于Ranger可以对Hive表的HDFS路径进行权限控制，HiveMetaStore可以配置Storage Based Authorization，因此二者结合可以实现对Hive Client访问场景的权限控制。

    说明 Hive Client场景的DDL操作权限通过底层HDFS控制权限，所以如果您有HDFS权限，则对应也会有表的DDL操作权限（例如Drop Table或Alter Table等）。
- HDFS方式

- 场景： 您可以通过HDFS访问数据。
- 方式： HDFS客户端或代码等。
- 权限设置：

  您可以直接访问HDFS，需要对Hive表的底层HDFS数据增加HDFS的权限控制。

  通过Ranger对Hive表底层的HDFS路径进行权限控制，详情请参见权限配置示例。

Hive集成Ranger

1. Ranger启用Hive。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏选择**集群服务 > RANGER**。
   vi. 单击右侧的**操作**下拉菜单，选择**启用Hive**。

   vii. 执行集群操作。

   单击右上角**查看操作历史**查看任务进度，等待任务完成。

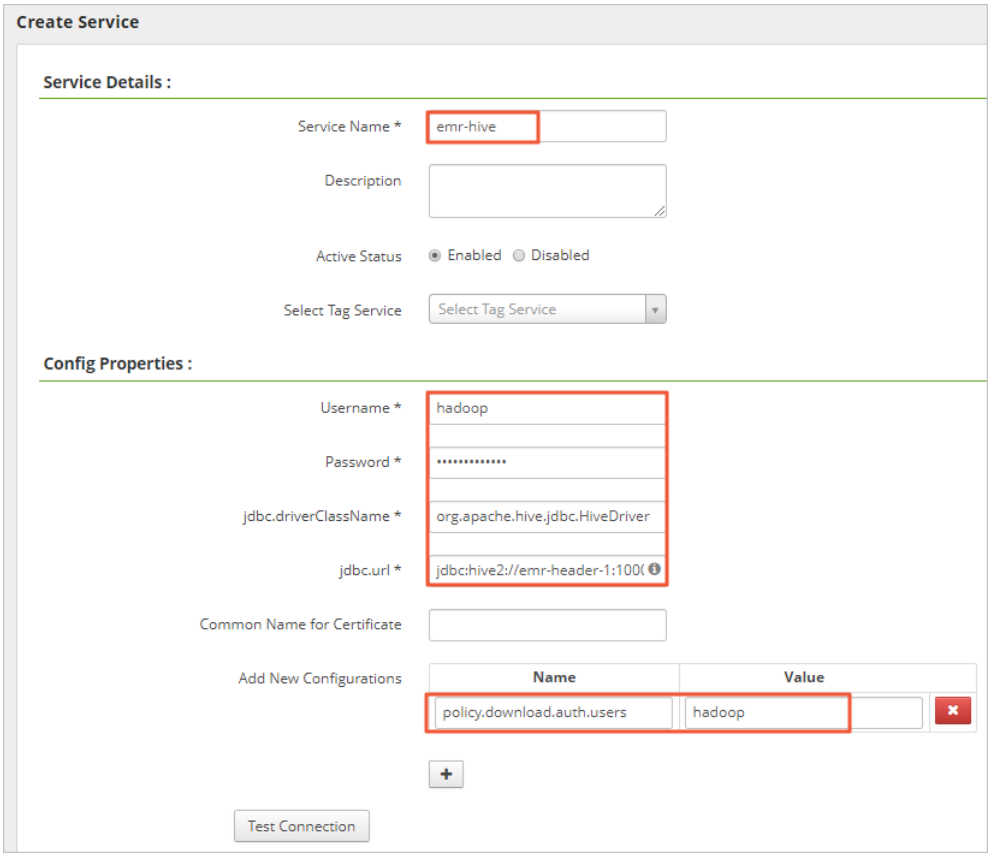
> **说明** Ranger启用Hive后，HiveServer2场景下，您需要使用Ranger进行权限配置。HiveClient场景下，您需要借助HDFS权限进行控制。开启HDFS的权限请参见HDFS配置。

2. Ranger UI添加Hive Service。
   i. 进入Ranger UI页面，详情请参见概述。

ii. 在Ranger UI页面，添加Hive Service。

当您的Ranger版本为2.1.0版本时，截图如下：

iii. 配置参数。



参数	说明
Service Name	固定值emr-hive。
Username	固定值hadoop。
Password	自定义。
jdbc.driverClassName	默认值org.apache.hive.jdbc.HiveDriver。无需修改。
jdbc.url	■ 标准集群：jdbc:hive2://emr-header-1:10000/ ■ 高安全集群：jdbc:hive2://${master1_fullhost}:10000/;principal=hive/${master1_fullhost}@EMR.$id.COM 说明 ${master1_fullhost}为master 1的长域名，可登录master 1执行 hostname 命令获取，${master1_fullhost} 中的数字即为$id的值。
Add New Configurations	■ Name：固定值policy.download.auth.users。 ■ Value：hadoop（标准集群）、hive（高安全集群）。

iv. 单击Add。

3. 重启Hive。

上述任务完成后，需要重启Hive才生效。

i. 左侧导航栏单击**集群服务** > **Hive**。

ii. 单击右上角**操作**下拉菜单，选择**重启 All Components**。

iii. 执行集群操作。

a. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。

b. 在**确认**对话框，单击**确定**。

c. 单击右上角**查看操作历史**查看任务进度，等待任务完成。

权限配置示例

- 示例一：给foo用户授予表testdb.test的a列Select权限

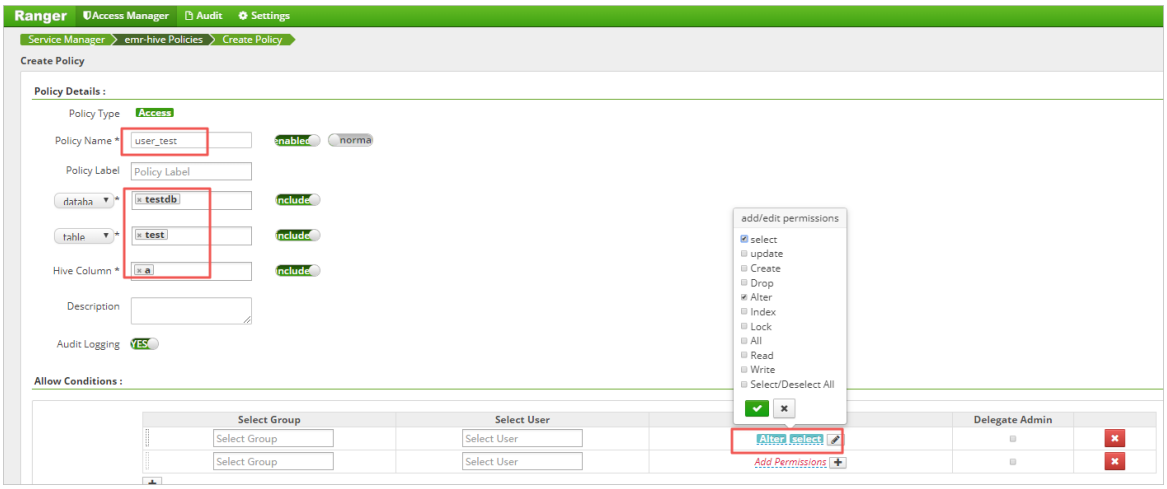
i. 进入Ranger UI页面，详情请参见概述。

ii. 在Ranger UI页面，单击配置好的**emr-hive**。

当您的Ranger版本为2.1.0版本时，截图如下：

iii. 单击右上角的**Add New Policy**。

iv. 配置权限。



参数	说明
Policy Name	策略名称，可以自定义。
database	添加Hive中的数据库，例如testdb。
table	添加表，例如test。
Hive Column	添加列名。填写星号（*）时表示所有列。
Select Group	指定添加此策略的用户组。
Select User	指定添加此策略的用户。例如，test。
Permissions	选择授予的权限。

v. 单击**Add**。

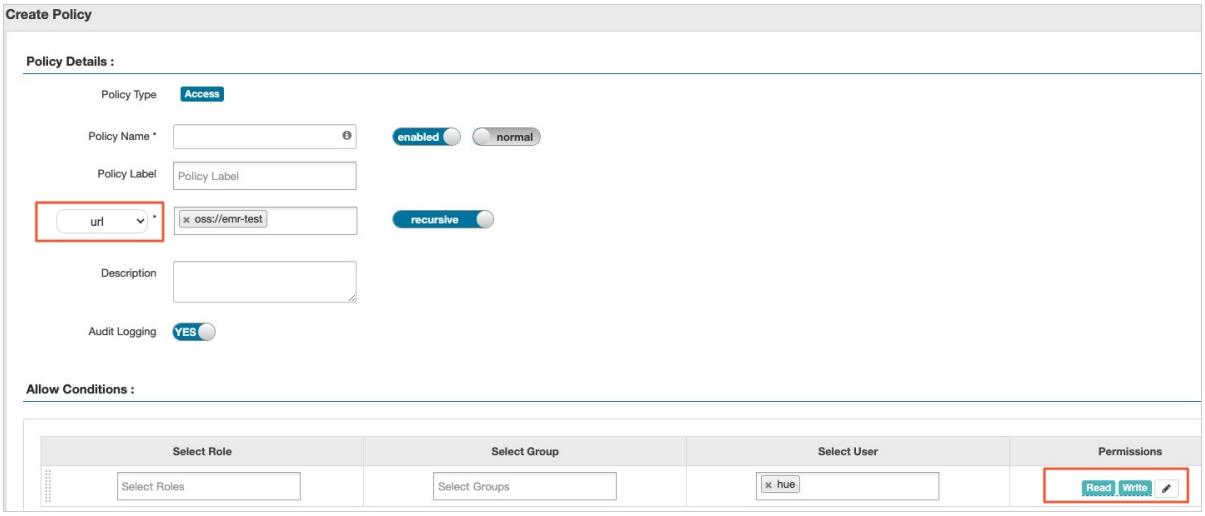
添加Policy后，实现对foo的授权。foo用户即可以访问testdb.test表。

> **说明** 添加、删除或修改Policy后，需要等待约一分钟至授权生效。

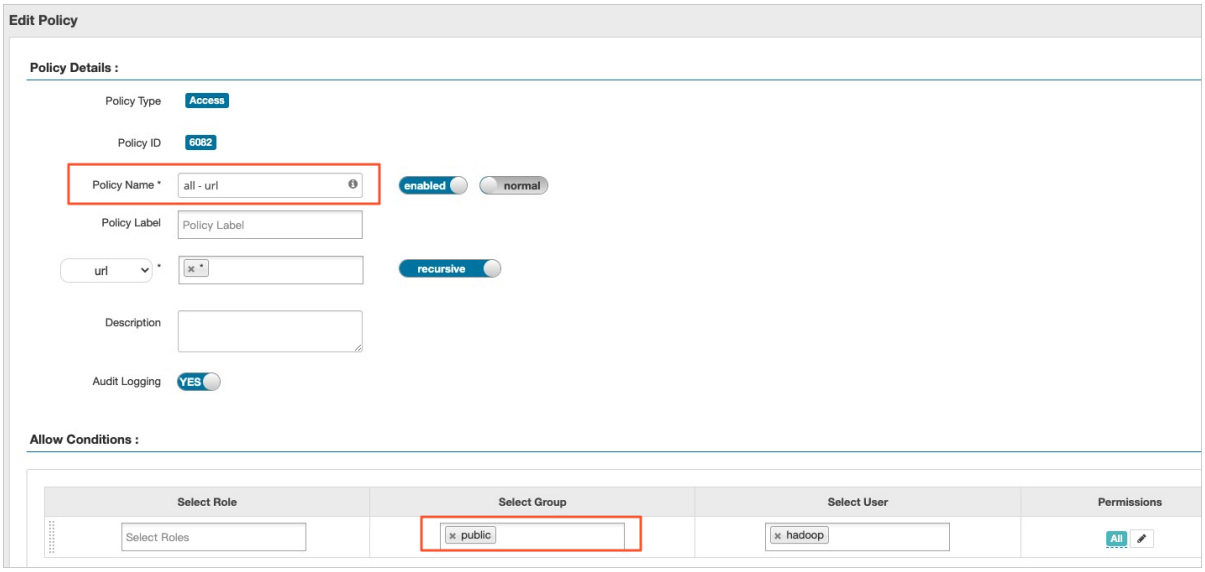
- 示例二：配置URL权限

> **说明** 本示例截图Ranger版本为2.1.0，其余版本以实际界面为准。

Hive开启Ranger权限访问控制后，在访问外表时（例如，表数据存储在OSS上），需要校验URL权限。添加URL权限Policy时，需要单击Policy配置页面的**database**，切换为url权限，然后添加对应外表的路径，配置对应用户的Read和Write权限。



如果您不需要对URL权限进行控制，可以在Ranger Hive默认创建的all - url权限中，在**Select Group**下拉列表中选择**public**，表示所有用户都能通过URL鉴权。



6.2.33.2.4. Spark配置

本文介绍如何将Spark集成到Ranger，以及相关的权限配置。

背景信息

Spark集成到Ranger进行权限控制，仅适用于通过Spark ThriftServer执行Spark SQL作业。例如，使用Spark的Beeline客户端或JDBC接口，通过Spark ThriftServer提交Spark SQL作业。

前提条件

已创建集群，并选择了Ranger服务，详情请参见创建集群。

注意事项

- 高安全集群（Kerberos集群模式）无法使用该文档功能。

- 因为Ranger和Delta不兼容，所以Spark Delta无法使用该文档功能。

Spark SQL集成Ranger

1. 在阿里云E-MapReduce控制台，配置Hive集成Ranger，详情请参见Hive配置。
   在Ranger中，因为Spark SQL与Hive共享权限配置，所以只需要将Hive集成Ranger，便可以通过Ranger控制Spark SQL的权限。
2. Ranger启用Spark。
   i. 在阿里云E-MapReduce控制台的**集群管理**页面，单击相应集群所在行的**详情**。
   ii. 在左侧导航栏单击**集群服务 > RANGER**。
   iii. 单击右侧的**操作**下拉菜单，选择**启用Spark**。

   iv. 执行集群操作。
      a. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
      b. 在**确认**对话框，单击**确定**。
      c. 单击右上角**查看操作历史**查看任务进度，等待任务完成。
3. 重启Spark ThriftServer。
   i. 在左侧导航栏单击**集群服务 > Spark**。
   ii. 在Spark**集群服务**页面，单击右侧的**操作 > 重启ThriftServer**。
   iii. 执行集群操作。
      a. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
      b. 在**确认**对话框，单击**确定**。
      c. 单击右上角**查看操作历史**查看任务进度，等待任务完成。

权限配置示例（Ranger UI配置相关权限）

例如：给用户foo授予表testdb.test的*a*列Select权限。

6.2.33.2.5. Presto配置

本文介绍如何将Presto集成到Ranger，以及如何配置权限。

前提条件

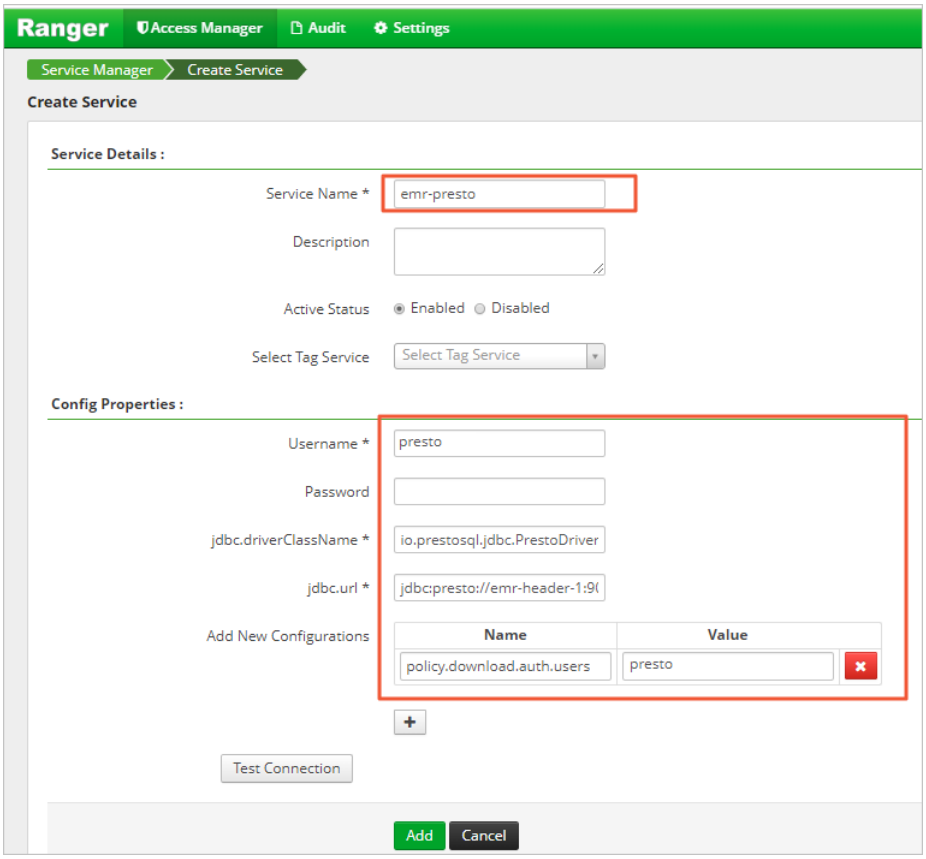
已创建E-MapReduce的集群，并且选择了Ranger和Presto服务。详情请参见创建集群。

Presto集成Ranger

1. Ranger启用Presto。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏选择**集群服务 > RANGER**。
   vi. 单击右侧的**操作**下拉菜单，选择**启用Presto**。
   vii. 执行集群操作。
      a. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
      b. 在**确认**对话框，单击**确定**。
      c. 单击右上角**查看操作历史**查看任务进度，等待任务完成。
2. Ranger UI添加Presto Service。
   i. 进入Ranger UI页面，详情请参见概述。

ii. 在Ranger UI页面，单击Presto所在行的 图标，添加Presto Service。

iii. 配置参数。



参数	说明
Service Name	固定值为emr-presto。
Username	固定值为presto。
Password	不填写。
jdbc.driverClassName	固定值为io.prestosql.jdbc.PrestoDriver。

参数	说明
jdbc.url	需要根据您的集群情况配置如下信息： ■ 普通集群：固定值为jdbc:presto://emr-header-1:9090。 ■ 高安全集群：填写格式为jdbc:presto://<your_hostname>:7778?SSL=true&SSLKeyStorePath=/etc/ecm/presto-conf/keystore&SSLKeyStorePassword=<your_keystore_password>&KerberosRemoteServiceName=presto&KerberosPrincipal=presto@EMR.<your_id>.COM&KerberosKeytabPath=/etc/ecm/presto-conf/jdbc.keytab 其中涉及参数如下所示： ■ <your_hostname>：您可以在集群的emr-header-1节点上执行 `hostname` 获取。 ■ <your_keystore_password>：您可以在集群的emr-header-1节点上执行 `sed -n 's/http-server.https.keystore.key=\([^;]*\)/\1/p' /etc/ecm/presto-conf/config.properties` 获取。 ■ <your_id>：您可以在集群的emr-header-1节点上执行 `hostname | grep -Eo '[0-9]+$'` 获取。 ■ jdbc.keytab文件创建步骤如下： a. 在集群的emr-header-1节点上执行命令 `sh /usr/lib/has-current/bin/admin-local.sh /etc/ecm/has-conf -k /etc/ecm/has-conf/admin.keytab` 。 b. 执行命令 `addprinc -randkey presto` 以生成Principal。 c. 执行命令 `xst -k /etc/ecm/presto-conf/jdbc.keytab presto` 以导出keytab文件。
Add New Configurations	■ **Name**：固定值为policy.download.auth.users。 ■ **Value**：固定值为presto。

iv. 单击**Add**。

3. 重启Presto Master。
   i. 在左侧导航栏，选择**集群服务 > Presto**。
   ii. 单击右上角**操作**下拉菜单，选择**重启PrestoMaster**。
   iii. 执行集群操作。
      a. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
      b. 在**确认**对话框，单击**确定**。
      c. 单击右上角**查看操作历史**查看任务进度，等待任务完成。

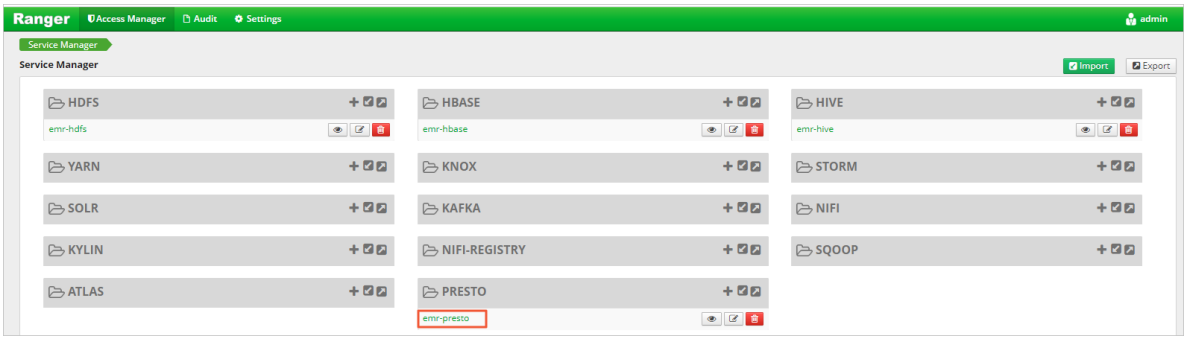
权限配置示例

Ranger Presto权限控制与Ranger Hive和Ranger Hbase等权限控制不同，Ranger Presto采用的是权限分层次控制的策略。

> 注意
> - 配置的权限应该与所属的层次保持一致。如果配置的权限与所属的层次不相符，则该权限配置将不起作用。
> - Presto会对用户进行两次权限检查，首先检查该用户是否有访问Catalog的权限，其次检查本次访问所涉及到的权限。

示例一：给用户liu授予访问hive表testdb.test的a列的Select权限。

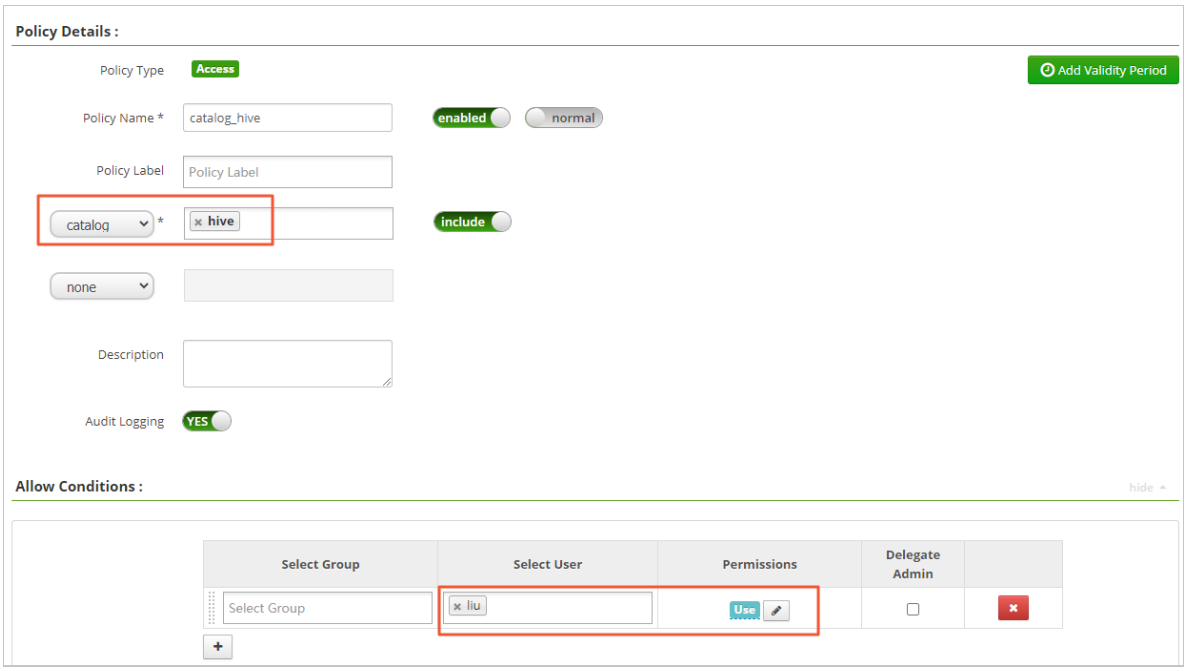
1. 进入Ranger UI页面，详情请参见概述。
2. 在Ranger UI页面，单击配置好的**emr-presto**。



3. 单击右上角的**Add New Policy**。

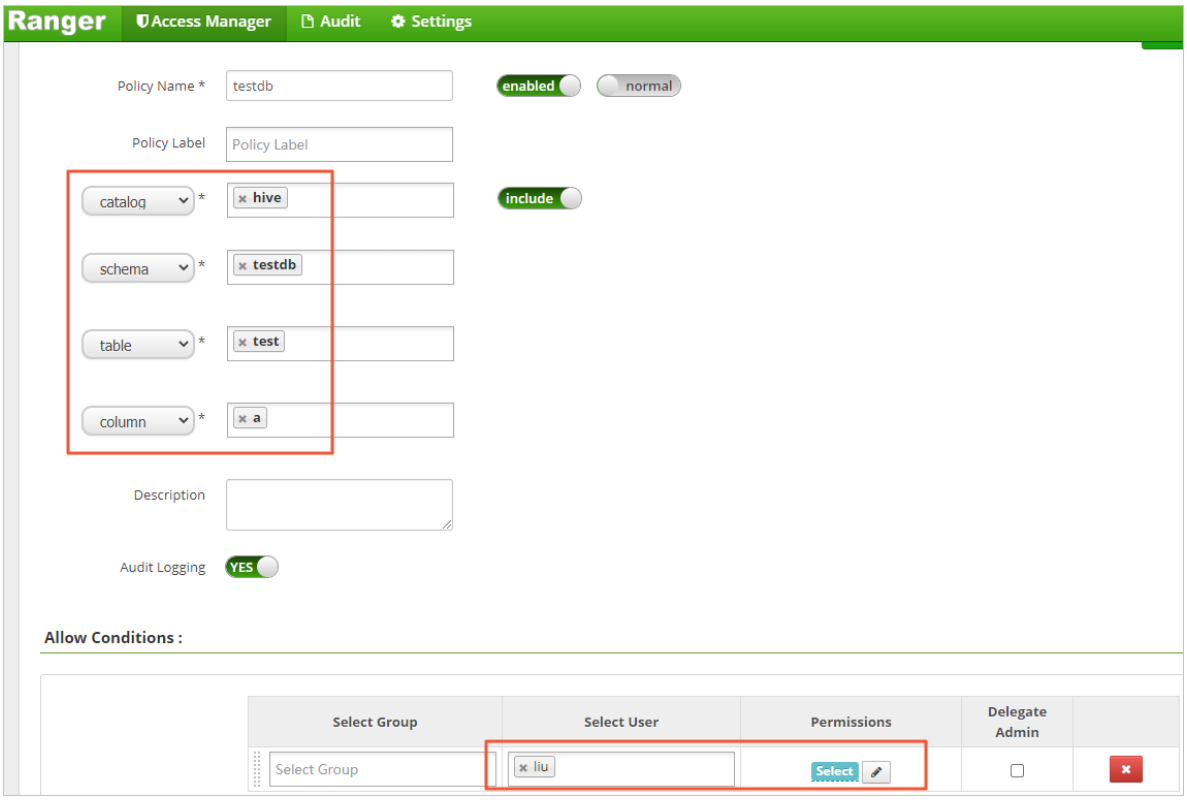
   添加Policy对**catalog**的访问权限进行控制。因为该权限为**catalog**层次，所以只需要配置到**catalog**层次即可。授权用户liu访问**catalog**的Use权限。

> 注意 EMR-3.28.1之前版本需要设置Select权限。



4. 单击**Add**。
5. 单击右上角的**Add New Policy**。

配置liu用户对表testdb.test的a列的Select权限。由于表的Select权限属于**column**层次，因此需要单击 none 图标，依次添加**schema**、**table**和**column**层次的内容。



参数	说明
Policy Name	策略名称，可自定义。例如testdb。
catalog	添加**catalog**的名称，可自定义。例如hive。
schema	添加**schema**的名称，例如testdb。星号（*）表示所有schema。
table	添加**table**的名称，例如test。星号（*）表示所有table。
column	添加**column**的名称，例如a。星号（*）表示所有column。
Select User	指定添加此策略的用户，例如liu。
Permissions	选择授予的权限，例如Select。

6. 单击**Add**。

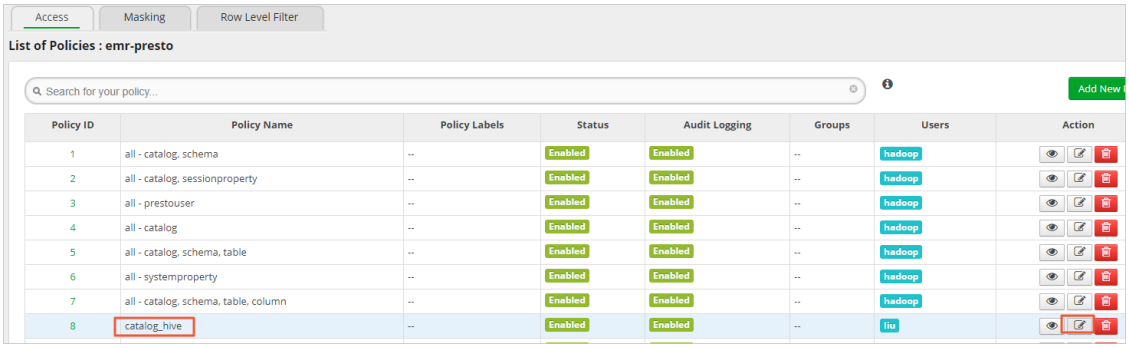
   添加Policy后，实现对liu用户的授权。liu用户即可以对testdb.test表的a列进行访问。

   > **说明** 添加、删除或修改Policy后，需要等待约一分钟至授权生效。

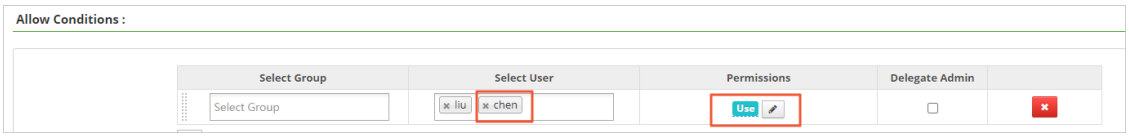
示例二：给用户chen添加创建hive表testdb.test的Create权限。

1. 进入Ranger UI页面，详情请参见概述。
2. 给用户chen添加Catalog访问权限（此例中Catalog=hive）。
   - 如果已经添加了Catalog层次的Policy，则直接编辑该Policy，在Select User中增加用户chen。
     a. 单击配置好的**emr-presto**。

b. 单击catalog_hive所在行的 图标。



c. 授权用户chen访问catalog的Use权限。



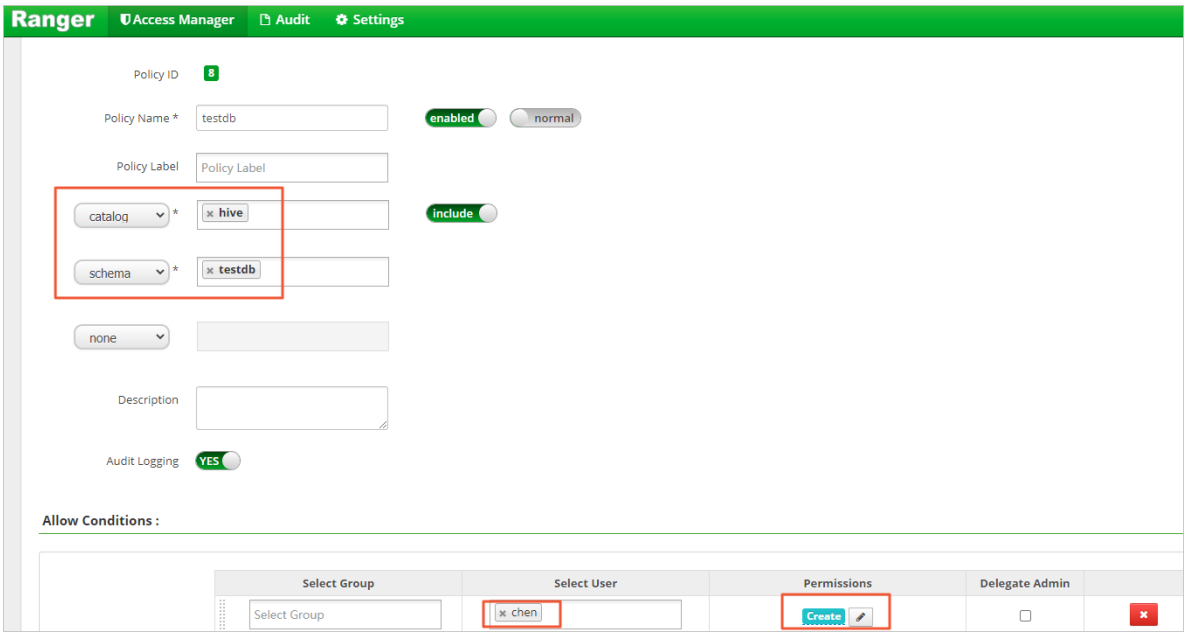
d. 单击Add。

- 如果还未添加Catalog层次的Policy，则按照示例一中的方法，授权用户chen访问catalog的Use权限。

> 注意 EMR-3.28.1之前版本需要设置Select权限。

3. 单击右上角的Add New Policy。

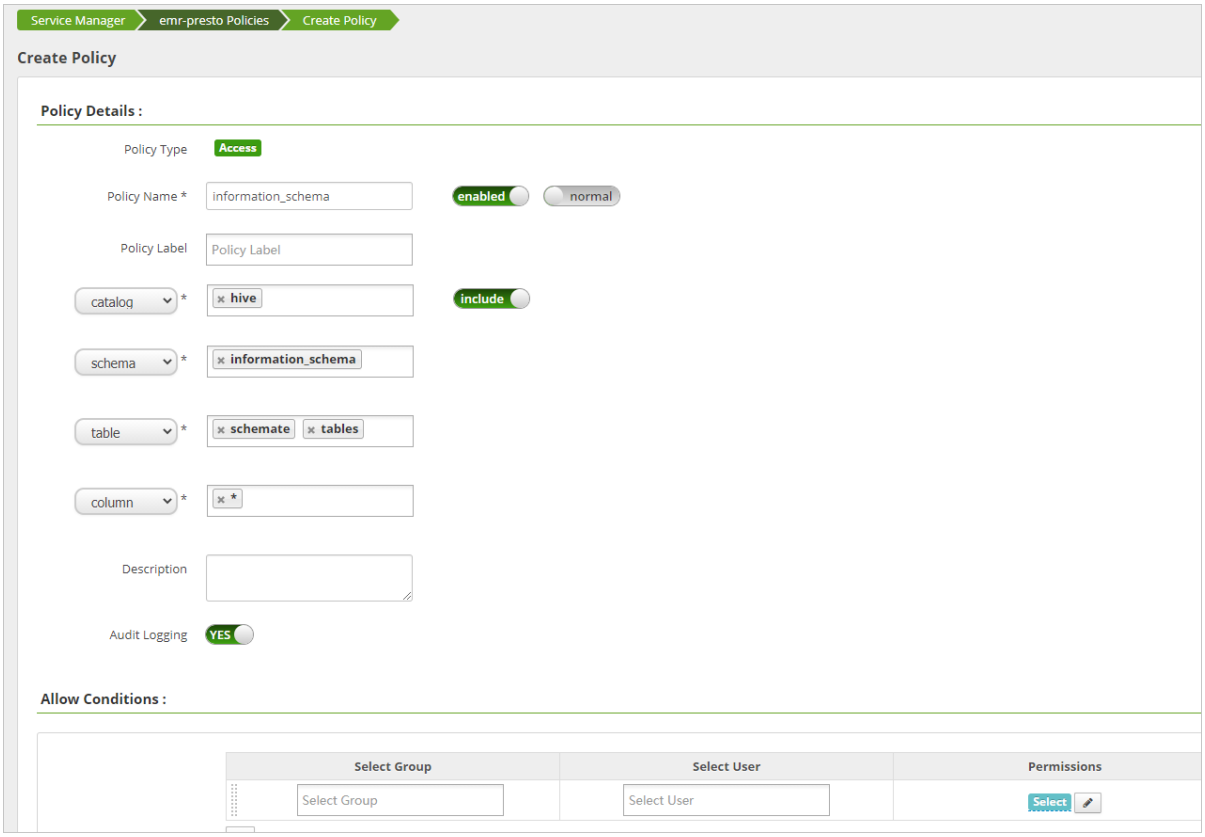
由于表的Create权限属于schema层次，因此您需要配置权限到schema层次。单击 none 按钮，添加schema层次的内容。



示例三：在Ranger UI页面，配置 `show schemas` 和 `show tables` 权限。

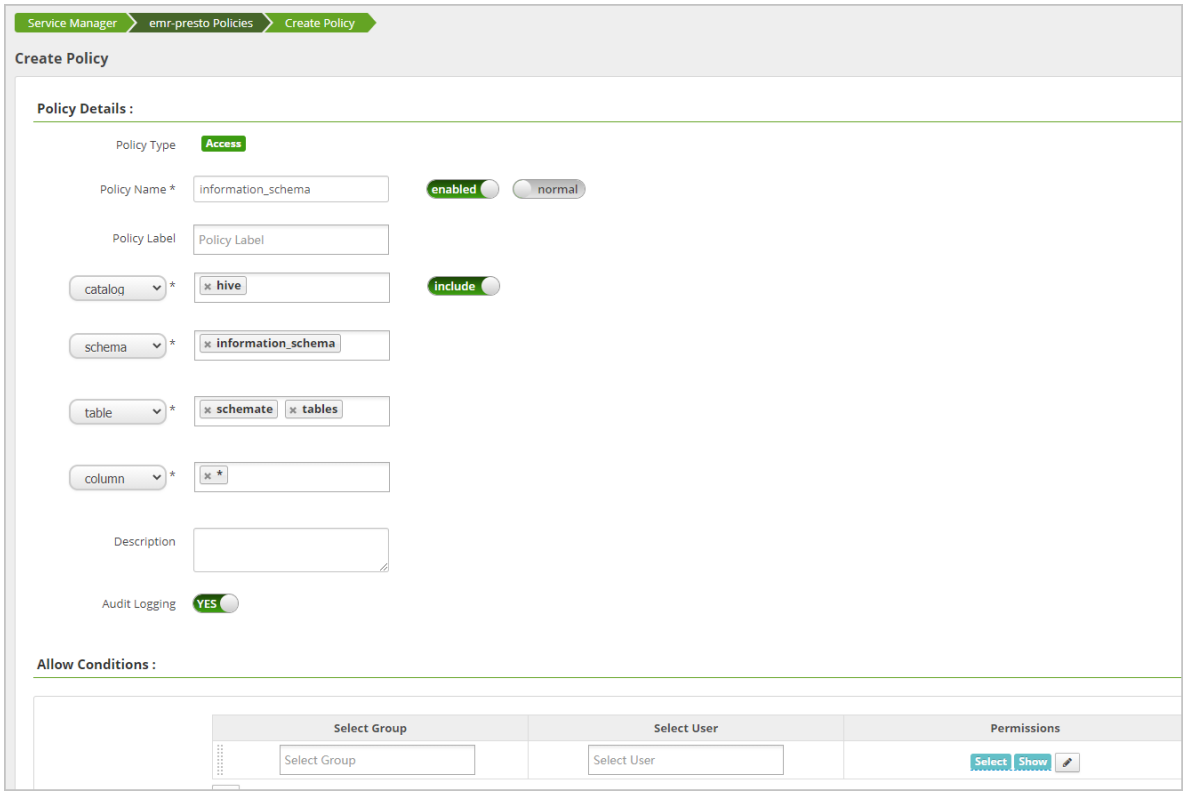
- EMR-3.28.1之前版本

您需要配置 `schema=information_schema,table=schemate,column=*` 的Select来获取 `show schemas` 执行的结果，配置 `schema=information_schema,table=tables,column=*` 的Select来获取 `show tables` 执行的结果，两者可以配置在一个Policy当中，示例如下。



- EMR-3.28.1及后续版本和EMR-4.4.1版本
  - 如果您需要执行 `show schemas` 命令，则需要配置对Catalog的Show权限。同理，如果您需要执行 `show tables` 命令，则需要配置对Schema的Show权限。
  - 因为Presto在鉴权完成后，还会对获取到的Schema和Table的列表进行一次筛选，只显示具有Select权限的Schema和Table，所以您还需要配置Schema和Table的Select权限。当您在执行Show命令时，只会显示出具有Select权限的Schema和Table。

- 只有拥有Catalog的Select权限时，才能显示出该Catalog下您具有权限的Schema和Table。



Ranger Presto与Ranger Hive共享权限配置

在部分场景下，Presto和Hive中配置的权限是一致的。由于EMR的Ranger提供了可以让Ranger Presto和Ranger Hive共享权限的方案，因此只需要在Ranger的Hive service中配置相关权限，Ranger Presto即可直接使用该权限配置用于检查用户权限。

> **说明** Ranger Presto与Ranger Hive共享权限配置，仅适用于Catalog为hive时的场景。

配置前需要关注以下事项：

- 已正确配置Ranger Hive，详情请参见Hive配置。
- 已在Ranger UI中添加Ranger Presto service。
- 与Ranger Presto一样，如果需要使用 `show schemas` 或 `show tables` 命令，则在Ranger Hive service中配置用户对 `database=information_schema,table=*,column=*` 的Select权限。

使用及配置方法如下：

1. 登录Master节点，修改配置文件*/etc/ecm/presto-conf/ranger-presto-security.xml*中的 `ranger.plugin.hive.authorization.enable` 修改为 `true` 。
2. 在左侧导航栏中，选择**集群服务 > Presto**。
3. 单击右上角的**操作 > 重启PrestoMaster**。
4. 执行集群操作。
   i. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
   ii. 在**确认**对话框，单击**确定**。
   iii. 单击右上角**查看操作历史**查看任务进度，等待任务完成。

6.2.33.2.6. Impala配置

本文介绍如何将Impala集成至Ranger，以及如何配置权限。

背景信息

Impala集成Ranger后，支持通过Impala-shell、Hue和JDBC方式在访问Hive表时进行权限控制。

前提条件

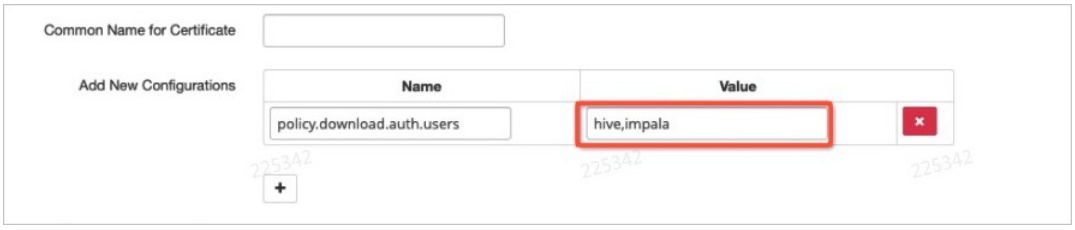
已创建EMR-4.4.1及后续版本的Hadoop集群，并且选择了Ranger和Impala服务。详情请参见创建集群。

Impala集成Ranger

1. 在阿里云E-MapReduce控制台配置Hive集成Ranger，详情请参见Hive配置。

   > **说明** 在Ranger中，因为Impala和Hive使用同一个Ranger Service（emr-hive）进行权限控制，所以需要先配置好Ranger Hive。

   因为Impala需要下载emr-hive中的Policy，所以在emr-hive配置**Add New Configurations**参数时，需要在**policy.download.auth.users**的**Value**中添加impala用户。

2. Ranger启用Impala。
   i. 在阿里云E-MapReduce控制台的**集群管理**页面，单击相应集群所在行的**详情**。
   ii. 在左侧导航栏单击**集群服务 > RANGER**。
   iii. 在Ranger集群服务页面，单击右上角的**操作 > 启用Impala**。

   iv. 执行集群操作。
      a. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
      b. 在**确认**对话框，单击**确定**。
      c. 单击右上角**查看操作历史**查看任务进度，等待任务完成。
3. 重启Impala。
   i. 在左侧导航栏单击**集群服务 > Impala**。
   ii. 在Impala集群服务页面，单击右侧的**操作 > 重启All Components**。
   iii. 执行集群操作。
      a. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
      b. 在**确认**对话框，单击**确定**。
      c. 单击右上角**查看操作历史**查看任务进度，等待任务完成。

权限配置示例

> **说明** Ranger中Impala暂不支持Hive Row-level Filter及Role权限功能。

例如：给用户foo授予表testdb.test的*a*列Select权限。

关闭Impala集成Ranger

当您不需要使用Ranger控制Impala权限时，可以执行如下方法关闭Impala集成Ranger。

1. Ranger禁用Impala。
   i. 在阿里云E-MapReduce控制台的**集群管理**页面，单击相应集群所在行的**详情**。

ii. 在左侧导航栏单击**集群服务 > RANGER**。
iii. 在Ranger集群服务页面，单击右上角的**操作 > 禁用Impala**。
iv. 执行集群操作。
a. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
b. 在**确认**对话框，单击**确定**。
c. 单击右上角**查看操作历史**查看任务进度，等待任务完成。

2. 重启Impala。
i. 在左侧导航栏单击**集群服务 > Impala**。
ii. 在Impala集群服务页面，单击右上角的**操作 > 重启All Components**。
iii. 执行集群操作。
a. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
b. 在**确认**对话框，单击**确定**。
c. 单击右上角**查看操作历史**查看任务进度，等待任务完成。

6.2.33.2.7. YARN配置

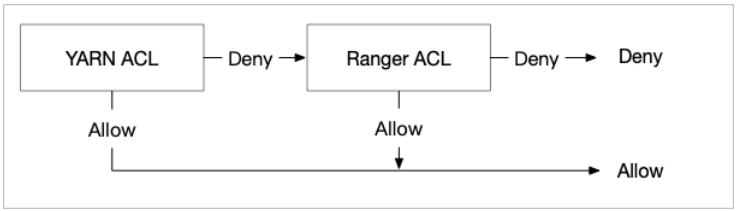
本文介绍如何将YARN集成至Ranger，以及如何实现YARN队列的权限控制。

前提条件

已创建EMR-3.34.0及后续版本或EMR-4.8.0及后续版本的集群，并且选择了Ranger服务，详情请参见创建集群。

背景信息

Ranger YARN仅支持Scheduler队列的权限控制，不支持Fair队列的权限控制。Ranger YARN队列权限控制与YARN自带的Capacity Scheduler配置共同生效，且优先级低于Capacity Scheduler配置。只有在YARN自带的Capacity Scheduler配置校验被拒绝时，才会校验Ranger YARN权限。鉴权流程如下图所示。



YARN集成Ranger

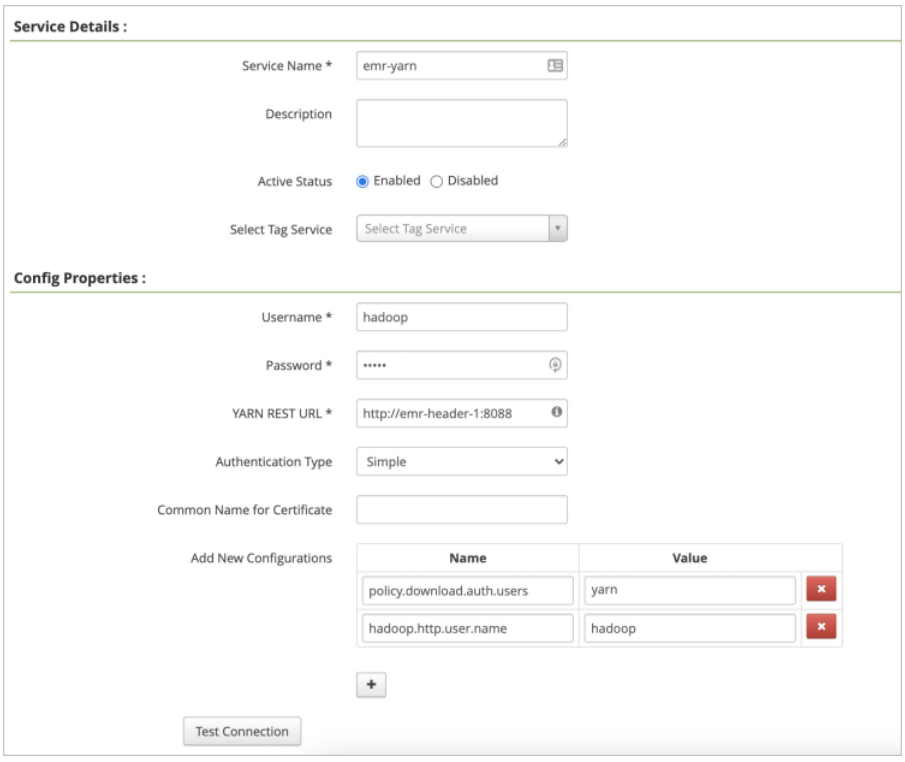
> **注意** 请谨慎执行以下操作，确保在配置过程中集群不会提交YARN作业，并在开启Ranger YARN后，给提交YARN作业的用户配置相应的队列权限，否则会出现作业无法提交的问题。

1. 启用YARN。
i. 登录阿里云E-MapReduce控制台。
ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
iii. 单击上方的**集群管理**页签。
iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
v. 在左侧导航栏中，选择**集群服务 > RANGER**。
vi. 单击右侧的**操作**下拉菜单，选择**启用YARN**。
vii. 在**执行集群操作**对话框，填写**执行原因**，单击**确定**。
viii. 在**确认**对话框，单击**确定**。
a. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
b. 在**确认**对话框，单击**确定**。
c. 单击右上角**查看操作历史**查看任务进度，等待任务完成。

2. Ranger UI添加YARN Service。
i. 进入Ranger UI页面，详情请参见概述。

ii. 在Ranger UI页面，单击YARN所在行的 图标，添加YARN Service。

iii. 配置参数。



参数	说明
Service Name	固定值为emr-yarn。
Username	固定值为hadoop。
Password	自定义。
Authentication Type	■ 普通集群选择**Simple**。 ■ 高安全集群选择**Kerberos**。
YARN REST URL	*http://emr-header-1:8088*。
Add New Configurations	■ **Name1**：固定值为policy.download.auth.users。 ■ **Value1**：固定值为yarn。 ■ **Name2**：固定值为hadoop.http.user.name。 ■ **Value2**：固定值为hadoop。

iv. 单击**Add**。

3. 重启YARN ResourceManager。

i. 在YARN集群服务页面，选择**操作 > 重启ResourceManager**。

ii. 在**执行集群操作**对话框，填写**执行原因**，单击**确定**。

iii. 在**确认**对话框，单击**确定**。

a. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。

b. 在**确认**对话框，单击**确定**。

c. 单击右上角**查看操作历史**查看任务进度，等待任务完成。

4. 修改capacity-scheduler配置。

i. 在YARN集群服务页面，单击**配置**页签。

ii. 在**服务配置**区域，单击**capacity-scheduler**页签。

iii. 修改**xml-direct-to-file-content**中的内容。

> **说明** 建议您可以拷贝**xml-direct-to-file-content**中的全部内容在文本编辑器中编辑。

a. 删除以下内容：

```
<property>
  <name>yarn.scheduler.capacity.root.default.acl_submit_applications</name>
  <value>*</value>
  <description>The ACL of who can submit jobs to the default queue.</description>
</property>
<property>
  <name>yarn.scheduler.capacity.root.default.acl_administer_queue</name>
  <value>*</value>
  <description>The ACL of who can administer jobs on the default queue.</description>
</property>
```

b. 添加以下内容：

```
<property>
  <name>yarn.scheduler.capacity.root.acl_submit_applications</name>
  <value> </value>
  <description>The ACL of who can submit jobs to the root queue.</description>
</property>
<property>
  <name>yarn.scheduler.capacity.root.acl_administer_queue</name>
  <value> </value>
  <description>The ACL of who can administer jobs on the root queue.</description>
</property>
```

> **说明** 添加内容中的 `<value> </value>`，中间是有空格的，表示不允许任何用户向root队列提交作业和管理root队列。

5. 保存配置。

i. 单击右上角的**保存**。

ii. 在**确认修改**对话框中，输入**执行原因**，开启**自动更新配置**。

iii. 单击**确定**。

6. 刷新Queues。

i. 在YARN集群服务页面，选择**操作 > 刷新Queues**。

ii. 在**执行集群操作**对话框，填写**执行原因**，单击**确定**。

iii. 在**确认**对话框，单击**确定**。

a. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。

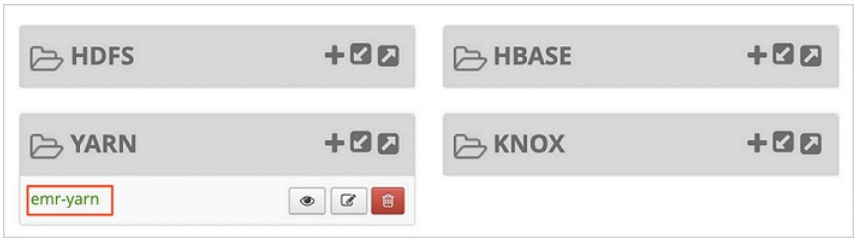
b. 在**确认**对话框，单击**确定**。

c. 单击右上角**查看操作历史**查看任务进度，等待任务完成。

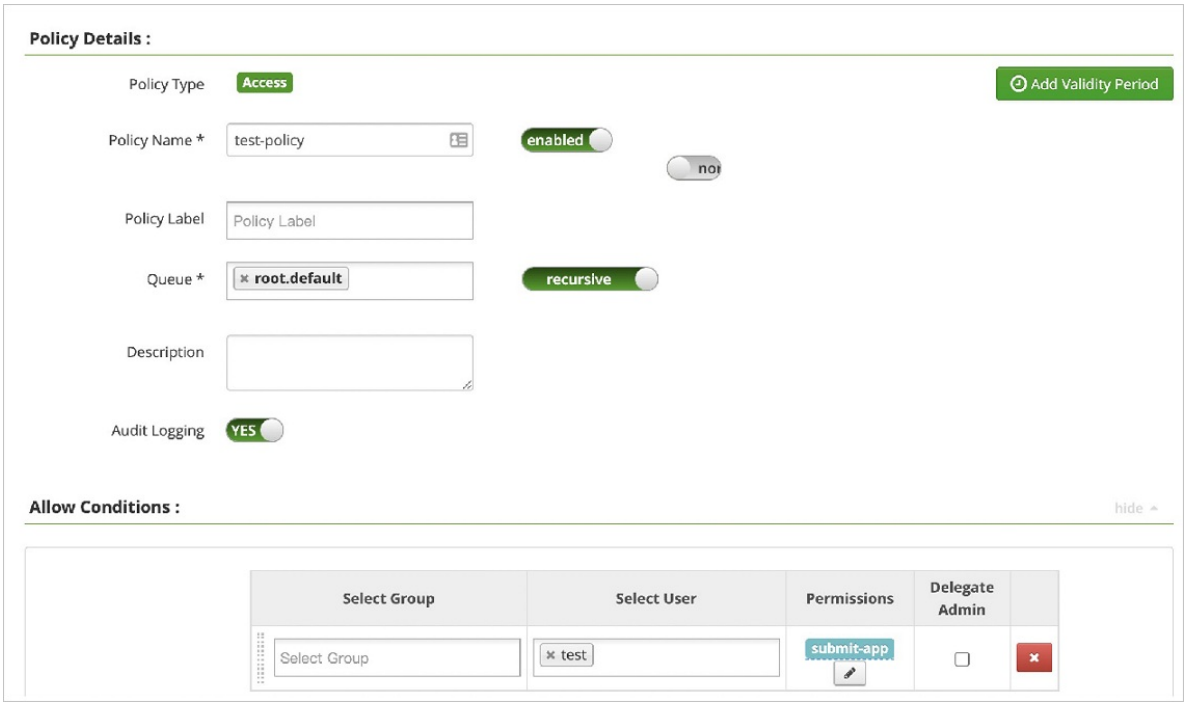
权限配置示例

例如，授予test用户向default队列提交作业的权限。

1. 进入Ranger UI页面，详情请参见概述。
2. 在Ranger UI页面，单击配置好的**emr-yarn**。



3. 单击右上角的Add New Policy。
4. 配置权限。



参数	说明
Policy Name	策略名称，可以自定义。
Queue	队列名称，例如root.default。
recursive	子队列是否继承该权限。
Select Group	指定添加此策略的用户组。
Select User	指定添加此策略的用户。例如，test。
Permissions	选择授予的权限。

5. 单击Add。

   添加Policy后，实现对test的授权。test用户可以向default队列提交作业。

> **说明** 添加、删除或修改Policy后，需要等待约一分钟至授权生效。

6.2.33.2.8. JindoFS配置

本文介绍如何将JindoFS集成到Ranger，以及如何配置权限。

前提条件

启用JindoFS Ranger权限

1. 进入SmartData服务。
   i. 登录阿里云E-MapReduce控制台。

ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
iii. 单击上方的**集群管理**页签。
iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
v. 在左侧导航栏，选择**集群服务 > SmartData**。

2. 添加Ranger。
   i. 在**namespace**页签，单击**自定义配置**。
   ii. 在**新增配置项**对话框中，设置**Key**为jfs.namespaces.<namespace>.permission.method，**Value**为ranger。
   iii. 保存配置。
      a. 单击右上角的**保存**。
      b. 在**确认修改**对话框中，输入执行原因，开启**自动更新配置**。
      c. 单击**确定**。
   iv. 重启配置。
      a. 单击右上角的**操作 > 重启 Jindo Namespace Service**。
      b. 输入执行原因，单击**确定**。
3. 配置Ranger。
   i. 进入Ranger UI页面。
   详情请参见概述。
   ii. Ranger UI添加HDFS service。

   iii. 配置相关参数。

参数	描述
Service Name	固定格式：jfs-{namespace_name}。 例如：jfs-test。
Username	自定义。
Password	自定义。
Namenode URL	输入*jfs:///{namespace_name}/*。
Authorization Enabled	使用默认值No。
Authentication Type	使用默认值Simple。
dfs.datanode.kerberos.principal	不填写。
dfs.namenode.kerberos.principal	
dfs.secondary.namenode.kerberos.principal	
Add New Configurations	

   iv. 单击**Add**。

启用JindoFS Ranger权限+LDAP用户组

如果您在Ranger UserSync上开启了从LDAP同步用户组信息的功能，则JindoFS也需要修改相应的配置，以获取LDAP的用户组信息，从而对当前用户组进行Ranger权限的校验。

1. 在**namespace**页签，单击**自定义配置**。
2. 在**新增配置项**对话框中，参见以下示例设置参数来配置LDAP，单击**确定**。
   以下配置项请遵循开源HDFS内容，详情请参见core-default.xml。

参数	示例
hadoop.security.group.mapping	org.apache.hadoop.security.CompositeGroupsMapping

参数	示例
hadoop.security.group.mapping.providers	shell4services,ad4users
hadoop.security.group.mapping.providers.combined	true
hadoop.security.group.mapping.provider.shell4services	org.apache.hadoop.security.ShellBasedUnixGroupsMapping
hadoop.security.group.mapping.provider.ad4users	org.apache.hadoop.security.LdapGroupsMapping
hadoop.security.group.mapping.ldap.url	ldap://emr-header-1:10389
hadoop.security.group.mapping.ldap.search.filter.user	(&(objectClass=person)(uid={0}))
hadoop.security.group.mapping.ldap.search.filter.group	(objectClass=groupOfNames)
hadoop.security.group.mapping.ldap.base	o=emr

3. 保存配置。
   i. 单击右上角的**保存**。
   ii. 在**确认修改**对话框中，输入执行原因，开启**自动更新配置**。
   iii. 单击**确定**。
4. 重启配置。
   i. 单击右上角的**操作 > 重启 All Components**。
   ii. 输入执行原因，单击**确定**。
5. 通过SSH登录emr-header-1节点，配置Ranger UserSync并启用LDAP选项。
   详情请参见Ranger Usersync集成LDAP。

6.2.33.2.9. JindoFS OSS配置

Apache Ranger提供了集中式的权限管理框架，可以对Hadoop生态中的多个组件进行细粒度的权限访问控制。当您将数据存放在阿里云OSS时，则通过阿里云RAM产品创建或管理RAM用户，对RAM用户实现OSS资源的访问控制。本文介绍如何将OSS集成到Ranger，以及如何配置权限。

前提条件

已创建高安全集群，并且选择了Ranger服务，创建集群详情请参见创建集群。

> **说明** 高安全集群，即在创建集群时，在**软件配置**页面的**高级设置**区域中，打开**Kerberos集群模式**开关。

使用限制

该功能支持的集群版本如下：

- EMR-5.4.2版本或EMR-3.38.2版本
- EMR-5.6.0及后续版本或EMR-3.40.0及后续版本

OSS集成Ranger

1. 进入集群详情页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
2. 启用OSS。
   i. 在左侧导航栏中，选择**集群服务 > RANGER**。
   ii. 在RANGER服务页面，选择**操作 > 启用OSS**。
   iii. 在**执行集群操作**对话框中，填写**执行原因**，单击**确定**。
   iv. 在**确认**对话框，单击**确定**。
3. 部署客户端配置。

- EMR-5.6.0及后续版本或EMR-3.40.0及后续版本
  a. 在左侧导航栏中，选择**集群服务 > HDFS**。
  b. 在HDFS服务页面，单击**配置**页签。
  c. 在HDFS服务页面，单击右上角的**部署客户端配置**。
  d. 在y中，输入执行原因，单击**确定**。
  e. 在**确认**对话框中，单击**确定**。
  您可以单击上方的**查看操作历史**，查看执行状态和进度。
- EMR-5.4.2版本或EMR-3.38.2版本
  a. 在左侧导航栏中，选择**集群服务 > SmartData**。
  b. 在SmartData服务页面，单击**配置**页签。
  c. 在SmartData服务页面，单击右上角的**部署客户端配置**。
  d. 在**执行集群操作**中，输入执行原因，单击**确定**。
  e. 在**确认**对话框中，单击**确定**。
  您可以单击上方的**查看操作历史**，查看执行状态和进度。

4. 重启Jindofsx Namespace Service或Jindo Namespace Service。
   - EMR-5.6.0及后续版本或EMR-3.40.0及后续版本
     a. 在**JindoData**服务页面的右上角，选择**操作 > 重启Jindofsx Namespace Service**。
     b. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
     c. 在弹出的**确认**对话框中，单击**确定**。
   - EMR-5.4.2版本或EMR-3.38.2版本
     a. 在**SmartData**服务页面的右上角，选择**操作 > 重启Jindo Namespace Service**。
     b. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
     c. 在弹出的**确认**对话框中，单击**确定**。
5. 重启HiveServer2。
   i. 在左侧导航栏中，选择**集群服务 > Hive**。
   ii. 在Hive服务页面的右上角，选择**操作 > 重启HiveServer2**。
   iii. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
   iv. 在弹出的**确认**对话框中，单击**确定**。
6. 创建用户Principal。
   i. 通过SSH方式连接集群的emr-header-1节点，详情请参见登录集群。
   ii. 执行如下命令，进入Kerberos的admin工具。
   ```
   sh /usr/lib/has-current/bin/admin-local.sh /etc/ecm/has-conf -k /etc/ecm/has-conf/admin.keytab
   ```
   iii. 执行如下命令，创建用户名为test的Principal。
   本示例密码设置为123456。
   ```
   addprinc -pw 123456 test
   ```
   > **说明** 需要记录用户名和密码，在创建TGT时会用到。如果您不想记录用户名和密码，则可以执行下一步，将Principal的用户名和密码导入到keytab文件中。
   iv. （可选）执行如下命令，生成keytab文件。
   ```
   ktadd -k /root/test.keytab test
   ```
   执行`quit`命令，可以退出Kerberos的admin工具。
7. 创建TGT。
   创建TGT的机器，可以是任意一台需要运行Hive Client的机器。
   i. 使用root用户执行以下命令，创建test用户。
   ```
   useradd test
   ```
   ii. 执行以下命令，切换为test用户。
   ```
   su test
   ```

iii. 生成TGT。

- 方式一：使用用户名和密码方式，创建TGT。

  执行 `kinit` 命令，回车后输入test的密码123456。

  ```
  [root@emr-header-1 ~]# su test
  [test@emr-header-1 root]$ kinit
  Password for test@EMR.238075.COM:
  [test@emr-header-1 root]$
  ```

- 方式二：使用keytab文件，创建TGT。

  在步骤6中的*test.keytab*文件，已经保存在emr-header-1机器的*/root/*目录下，需要先使用 `cp /root/test.keytab /home/test/` 命令拷贝到当前机器的*/home/test/*目录下，再执行以下命令创建TGT。

  ```
  kinit -kt /home/test/test.keytab test
  ```

iv. 查看TGT。

使用 `klist` 命令，返回如下信息。

```
Ticket cache: FILE:/tmp/krb5cc_1012
Default principal: test@EMR.23****.COM
Valid starting       Expires              Service principal
07/24/2021 13:20:44  07/25/2021 13:20:44  krbtgt/EMR.23****.COM@EMR.23****.COM
        renew until 07/25/2021 13:20:44
```

权限配置示例

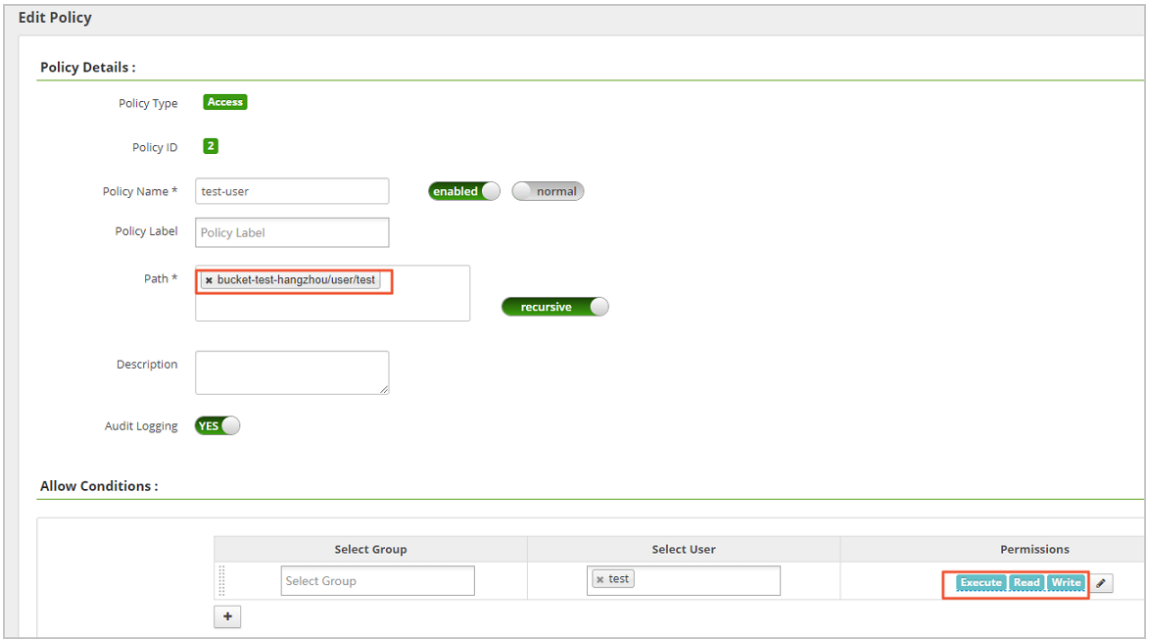
本文以EMR-3.38.2版本操作为例，其余版本请以实际界面为准。

配置test用户拥有访问*oss://bucket-test-hangzhou/user/test*目录的所有权限。

1. 进入Ranger UI页面，详情请参见概述。
2. 在Ranger UI页面，单击已有的**emr-oss**。

3. 配置*test*目录的ALL权限。
   i. 单击右上角的**Add New Policy**。

ii. 在**Edit Policy**页面，配置下表参数。



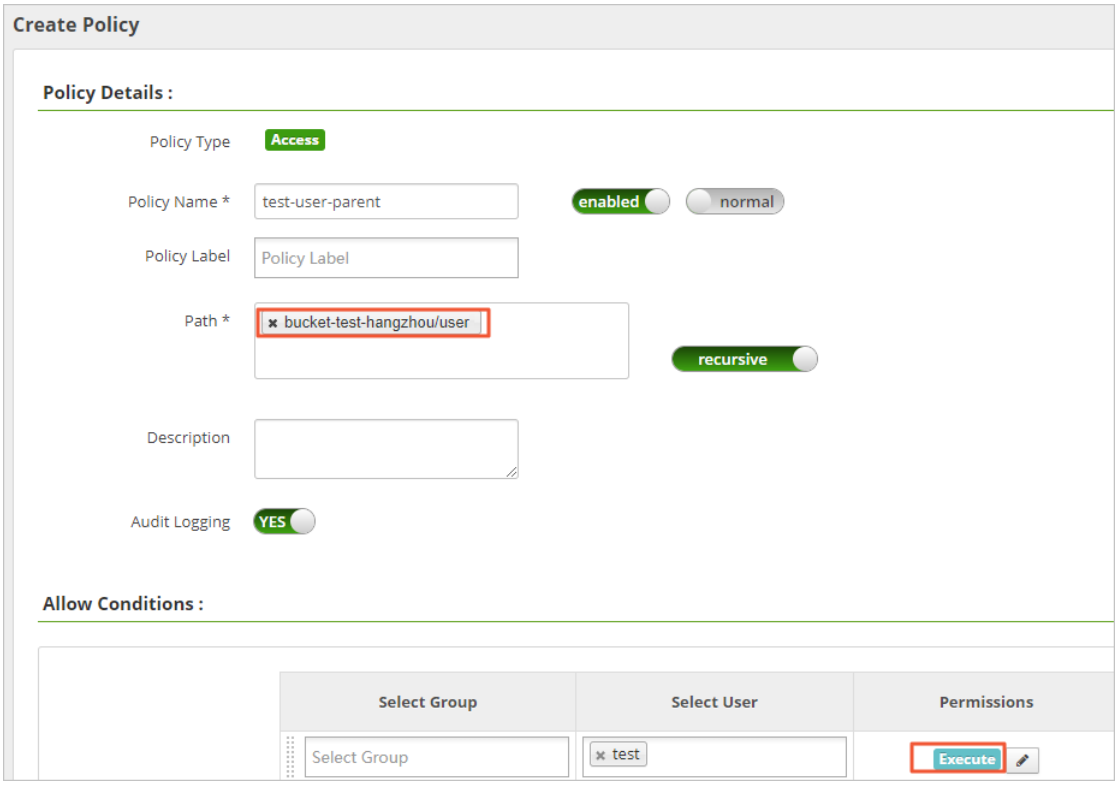
参数	描述
Policy Name	策略名称，可以自定义。
Path	OSS的路径。路径无需oss://前缀。例如，本示例为bucket-test-hangzhou/user/test。 注意 ■ 路径末尾无需带正斜线（/）。 ■ 禁止关闭**recursive**开关。
Select User	指定添加此策略的用户。 本示例设置为test用户。
Permissions	选择授予的权限。 本示例设置访问权限为ALL（Read、Write和Execute）。

iii. 单击**Add**。

4. 配置*test*目录的父目录（*user*目录）的Execute权限。

i. 单击右上角的**Add New Policy**。

ii. 在Edit Policy页面，配置下表参数。



参数	描述
Policy Name	策略名称，可以自定义。
Path	OSS的路径。路径无需oss://前缀。例如，本示例为bucket-test-hangzhou/user。 注意 ■ 路径末尾无需带正斜线（/）。 ■ 禁止关闭recursive开关。
Select User	指定添加此策略的用户。 本示例设置为test用户。
Permissions	选择授予的权限。 本示例设置访问权限为Execute。

iii. 单击Add。

5. 访问OSS。

i. 通过SSH方式连接集群的emr-header-1节点，详情请参见登录集群。

ii. 执行以下命令，切换为本文示例创建的test用户。

```
su test
```

iii. 执行以下命令，访问OSS目录。

```
hadoop fs -ls oss://bucket-test-hangzhou/user/test
```

如果当您访问Ranger没有授权的路径，则会提示以下错误信息。

```
org.apache.hadoop.security.AccessControlException: Permission denied: user=test, access=READ_EXECUTE, resourcePath="bucket-test-hangzhou/"
```

6.2.33.3. 高阶功能

6.2.33.3.1. Ranger Usersync集成LDAP

本文介绍Ranger Usersync如何集成LDAP，以便于您在配置Ranger的Policy时，可以授权LDAP中的用户或用户组访问组件。

前提条件

已创建集群，详情请参见创建集群。

使用限制

EMR-3.34.0及后续版本或EMR-4.8.0及后续版本的Hadoop集群，支持一键开启LDAP认证。

注意事项

EMR的OpenLDAP没有配置用户组，如果您需要使用LDAP配置用户组，需要自行配置。如果您需要同步LDAP用户组至Ranger，请根据LDAP实际配置，自行配置LDAP各项参数。

EMR-3.28.0及后续版本（EMR 3.x系列）和EMR-4.3.0及后续版本配置方法

1. 进入**配置**页签。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击待操作集群所在行的**详情**。
   v. 在左侧导航栏，单击**集群服务 > RANGER**。
   vi. 单击**配置**页签。
2. 在**ranger-ugsync-site**页面，配置各项参数。
   i. 在**服务配置**区域，单击**ranger-ugsync-site**页签。
   ii. 配置如下参数，同步LDAP用户至Ranger。

参数	描述
ranger.usersync.sync.source	固定值为ldap。
ranger.usersync.ldap.binddn	固定值为uid=admin,o=emr。
ranger.usersync.ldap.ldapbindpassword	LDAP管理员密码。您可以在OpenLDAP的配置页面，获取**manager_password**的值，即为LDAP管理员密码。 获取方式如下图。 
ranger.usersync.ldap.searchBase	固定值为o=emr。
ranger.usersync.ldap.url	固定值为ldap://emr-header-1:10389。
ranger.usersync.ldap.user.nameattribute	固定值为cn。
ranger.usersync.ldap.user.objectclass	固定值为person。
ranger.usersync.ldap.user.searchbase	固定值为ou=people,o=emr。
ranger.usersync.source.impl.class	固定值为org.apache.ranger.ldapusersync.process.LdapUserGroupBuilder。
ranger.usersync.sleeptimeinmillisbetweensynccycle	固定值为3600000。

> **说明** 当您创建的是高安全集群时，请配置**ranger.usersync.ldap.user.searchfilter**为 (!(cn=*/*))，以便于过滤掉OpenLDAP中已创建的组件服务的Kerberos Principal记录。

iii. （可选）如果您需要同步LDAP用户组至Ranger，请根据LDAP实际信息配置如下参数。

> 注意 EMR的OpenLDAP没有配置用户组，如果您需要使用LDAP配置用户组，需要自行配置。

参数	示例
ranger.usersync.group.memberattributename	member
ranger.usersync.group.nameattribute	cn
ranger.usersync.group.objectclass	groupofnames
ranger.usersync.group.searchbase	ou=groups,o=emr
ranger.usersync.group.searchenabled	true
ranger.usersync.group.usermapsyncenabled	true
ranger.usersync.sleeptimeinmillisbetweensynccycle	3600000

3. 重启Ranger UserSync，使配置生效。
   i. 在左侧导航栏，单击**集群服务 > RANGER**。
   ii. 在**组件列表**区域，单击**RangerUserSync**所在行的**重启**。
   iii. 在**执行集群操作**对话框中，配置各项参数。
   iv. 单击**确定**。
   v. 在**确认**对话框中，单击**确定**。

EMR-3.28.0之前版本（EMR 3.x系列）和EMR-4.3.0之前版本（EMR 4.x系列）配置方法

1. 登录集群的emr-header-1节点，详情请参见登录集群。
2. 编辑*install.properties*文件。

```
cd /usr/lib/ranger-usersync-current
vim install.properties
```

3. 修改如下配置项。

```
SYNC_SOURCE = ldap
SYNC_LDAP_URL = ldap://emr-header-1:10389
SYNC_LDAP_BIND_DN = uid=admin,o=emr
SYNC_LDAP_BIND_PASSWORD = [password]
SYNC_LDAP_USER_SEARCH_BASE = ou=people,o=emr
```

> 说明 当您创建的是高安全集群时，请配置SYNC_LDAP_USER_SEARCH_FILTER为 (!(cn=*/*))，以便于过滤掉OpenLDAP中已创建的组件服务的Kerberos Principal记录。

本示例给出的配置值均为对接的EMR OpenLDAP的值。如果您需要对接自建的LDAP，需要修改各参数值为自建LDAP对应的值。各参数详细解释，请参见Ranger Usersync官方安装教程。

参数	描述
`SYNC_LDAP_URL`	LDAP服务的地址。例如 `ldap://ldap.example.com:389` 。
`SYNC_LDAP_BIND_DN`	连接LDAP进行用户和用户组查询的dn。例如 `cn=ldapadmin,ou=users,dc=example,dc=com` 。
`SYNC_LDAP_BIND_PASSWORD`	用于连接的dn对应的密码。
`EARCH_BASE`	LDAP中用户搜索域。例如 `ou=users,dc=example,dc=com` 。

4. （可选）如果您需要同步LDAP用户组至Ranger，请根据LDAP实际信息修改如下参数。

> 注意 EMR的OpenLDAP没有配置用户组，如果您需要使用LDAP配置用户组，需要自行配置。

```
SYNC_LDAP_USER_GROUP_NAME_ATTRIBUTE = gitNumber
SYNC_GROUP_SEARCH_ENABLED = true
SYNC_GROUP_USER_MAP_SYNC_ENABLED = true
SYNC_GROUP_SEARCH_BASE = ou=group,o=emr
SYNC_GROUP_OBJECT_CLASS = posixGroup
SYNC_GROUP_NAME_ATTRIBUTE = cn
SYNC_GROUP_MEMBER_ATTRIBUTE_NAME = memberUid
```

参数	描述
SYNC_LDAP_USER_GROUP_NAME_ATTRIBUTE	用户条目（Entry）中表示用户组属性（Attribute）的名称。例如 gitNumber(user objectClass=posixAccount) 。
SYNC_GROUP_SEARCH_ENABLED	是否根据条目（Entry）中记录的用户组属性（Attribute）来确定用户组信息。 true 。
SYNC_GROUP_USER_MAP_SYNC_ENABLED	用户与用户组之间的映射关系是否通过LDAP搜索进行确定。例如 true 。
SYNC_GROUP_SEARCH_BASE	LDAP中用户搜索域。例如 ou=groups,dc=example,dc=com 。
SYNC_GROUP_OBJECT_CLASS	用户组的对象类（ObjectClass）。例如 posixGroup 。
SYNC_GROUP_NAME_ATTRIBUTE	用户组条目（Entry）中用户组名的标识。例如 cn 。
SYNC_GROUP_MEMBER_ATTRIBUTE_NAME	用户组条目（Entry）中标识用户组成员的属性（Attribute）名称。例如 memberUid 。

5. 在emr-header-1节点的*/usr/lib/ranger-usersync-current*路径下执行 setup.sh 。

```
cd /usr/lib/ranger-usersync-current
sh setup.sh
```

6. 重启Ranger UserSync，使配置生效。
   i. 在左侧导航栏，单击**集群服务 > RANGER**。
   ii. 在**组件列表**区域，单击**RangerUserSync**所在行的**重启**。
   iii. 在**执行集群操作**对话框中，配置各项参数。
   iv. 单击**确定**。
   v. 在**确认**对话框中，单击**确定**。

6.2.33.3.2. Ranger Admin集成LDAP

本文介绍Ranger Admin如何集成LDAP，以便于您使用LDAP中的用户登录Ranger WebUI。

背景信息

Ranger的用户可以分为Internal User和External User。LDAP和UNIX用户同步至Ranger后属于External User，在Ranger WebUI中创建的用户属于Internal User。在Ranger中配置权限时，可以给Internal User和External User授权。

Ranger Admin集成LDAP后，LDAP中的用户可以登录Ranger WebUI。登录后，Ranger会在用户管理中同步创建该用户为External User。初始状态下，该用户仅能查看Ranger Service和Policy。管理员用户admin可以在用户管理中，升级普通用户为管理员用户。

使用限制

EMR-3.34.0及后续版本或EMR-4.8.0及后续版本的Hadoop集群，支持一键开启LDAP认证。

EMR-3.28.0及后续版本（EMR 3.x系列）和EMR-4.3.0及后续版本配置方法

1. 进入**配置**页签。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击待操作集群所在行的**详情**。
   v. 在左侧导航栏，单击**集群服务 > RANGER**。
   vi. 单击**配置**页签。
2. 在**ranger-admin-site**页面，配置各项参数。

i. 在**服务配置**区域，单击**ranger-admin-site**页签。

ii. 配置如下参数，同步LDAP用户至Ranger。

参数	描述
ranger.ldap.bind.dn	固定值为uid=admin,o=emr。
ranger.ldap.bind.password	LDAP管理员密码。您可以在OpenLDAP的配置页面，获取**manager_password**的值，即为LDAP管理员密码。 获取方式如下图。 
ranger.ldap.base.dn	固定值为ou=people,o=emr。
ranger.authentication.method	固定值为LDAP。
ranger.ldap.url	固定值为ldap://emr-header-1:10389。
ranger.ldap.user.dnpattern	固定值为uid={0},ou=people,o=emr。

3. 重启Ranger Admin，使配置生效。

i. 在左侧导航栏，单击**集群服务 > RANGER**。

ii. 在**组件列表**区域，单击**RangerAdmin**所在行的**重启**。

iii. 在**执行集群操作**对话框中，配置各项参数。

iv. 单击**确定**。

v. 在**确认**对话框中，单击**确定**。

EMR-3.28.0之前版本（EMR 3.x系列）和EMR-4.3.0之前版本（EMR 4.x系列）配置方法

1. 登录集群的emr-header-1节点，详情请参见登录集群。
2. 编辑*install.properties*文件。

```
cd /usr/lib/ranger-usersync-current
vim install.properties
```

3. 修改如下配置项。

```
authentication_method = LDAP
xa_ldap_url = ldap://emr-header-1:10389
xa_ldap_userDNpattern = uid={0},ou=people,o=emr
xa_ldap_base_dn = ou=people,o=emr
xa_ldap_bind_dn = uid=admin,o=emr
xa_ldap_bind_password = [password]
```

本示例给出的配置值均为对接的EMR OpenLDAP的值。如果您需要对接自建的LDAP，需要修改各参数值为自建LDAP对应的值。各参数详细解释，请参见Ranger Admin官方安装教程。

配置项	描述
`xa_ldap_url`	LDAP服务的地址。例如 `ldap://ldap.example.com:389` 。
`xa_ldap_userDNpattern`	登录用户匹配LDAP dn的pattern。例如 `uid={0},ou=users,dc=example,dc=com` ，表示当用户hadoop在WebUI登录时，其对应检查的LDAP dn为 `uid=hadoop,ou=users,dc=example,dc=com` 。
`xa_ldap_base_dn`	LDAP中用户搜索域。例如 `ou=users,dc=example,dc=com` 。
`xa_ldap_bind_dn`	连接LDAP进行用户和用户组查询的dn。例如 `cn=ldapadmin,ou=users,dc=example,dc=com` 。
`xa_ldap_bind_password`	用于连接的dn对应的密码。

4. 在emr-header-1节点的*/usr/lib/ranger-usersync-current*路径下执行 `setup.sh` 。

```
cd /usr/lib/ranger-usersync-current
sh setup.sh
```

5. 重启Ranger Admin，使配置生效。
   i. 在左侧导航栏，单击**集群服务 > RANGER**。
   ii. 在**组件列表**区域，单击**RangerAdmin**所在行的**重启**。
   iii. 在**执行集群操作**对话框中，配置各项参数。
   iv. 单击**确定**。
   v. 在**确认**对话框中，单击**确定**。

6.2.33.3.3. 管理LDAP认证

服务开启LDAP认证功能后，访问服务时需要提供LDAP身份认证（LDAP用户名和密码），以便于提升服务的安全性。开启LDAP功能对接的LDAP为E-MapReduce自带的OpenLDAP。开启LDAP认证的功能可以方便您使用LDAP认证，避免了复杂的配置过程。本文为您介绍如何一键开启和关闭LDAP认证。

前提条件

- 已创建Hadoop集群，详情请参见创建集群。
- 创建Knox账号，详情请参见管理用户。
- 已开启8443端口，开启详情请参见设置安全组访问。

背景信息

本文为您介绍如何开启和关闭Ranger Admin和Ranger UserSync的LDAP认证功能。

- Ranger Admin开启LDAP认证
- Ranger Admin关闭LDAP认证
- Ranger UserSync开启LDAP认证
- Ranger UserSync关闭LDAP认证

使用限制

EMR-3.34.0及后续版本或EMR-4.8.0及后续版本的Hadoop集群，支持一键开启LDAP认证。

Ranger Admin开启LDAP认证

1. 进入Ranger页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务 > Ranger**。
2. 开启LDAP认证。
   i. 在Ranger服务页面，选择右上角的**操作 > Admin开启LDAP认证**。
   ii. 在**执行集群操作**对话中，单击**确认**。
3. 单击上方的**查看操作历史**。
   直至操作状态显示**成功**。
4. 重启RangerAdmin。
   i. 在Ranger服务页面，选择右上角的**操作 > 重启RangerAdmin**。
   ii. 在**执行集群操作**对话中，输入**执行原因**，单击**确定**。
   iii. 在**确认**对话中，单击**确定**。
   Ranger Admin开启LDAP后，登录Ranger Web UI时需要使用LDAP的用户名和密码，原有的admin用户还可以继续使用。登录的LDAP用户默认只有user权限，需要admin用户给LDAP用户赋予需要的权限。

Ranger Admin关闭LDAP认证

1. 进入Ranger页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。

iii. 单击上方的**集群管理**页签。
iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
v. 在左侧导航栏，选择**集群服务 > Ranger**。

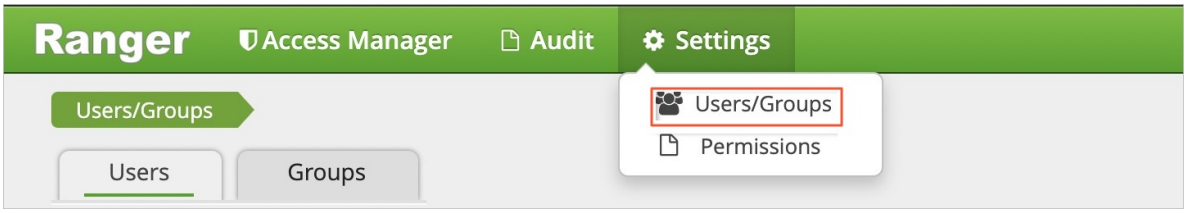
2. 关闭LDAP认证。
   i. 在Ranger服务页面，选择右上角的**操作 > Admin关闭LDAP认证**。
   ii. 在**执行集群操作**对话中，单击**确认**。
3. 单击上方的**查看操作历史**。
   直至操作状态显示**成功**。
4. 重启RangerAdmin。
   i. 在Ranger服务页面，选择右上角的**操作 > 重启RangerAdmin**。
   ii. 在**执行集群操作**对话中，输入**执行原因**，单击**确定**。
   iii. 在**确认**对话中，单击**确定**。

Ranger UserSync开启LDAP认证

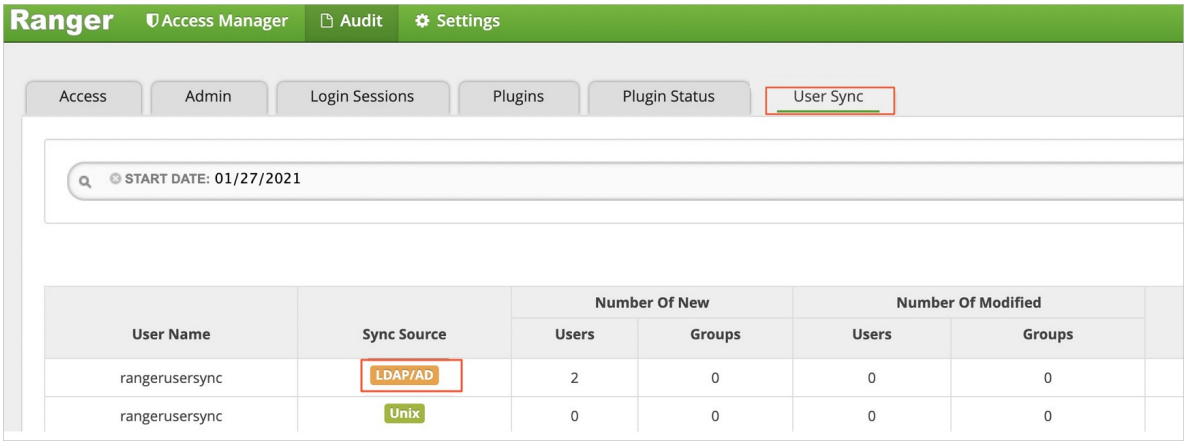
1. 进入Ranger页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务 > Ranger**。
2. 开启LDAP认证。
   i. 在Ranger服务页面，选择右上角的**操作 > UserSync开启LDAP认证**。
   ii. 在**执行集群操作**对话中，单击**确认**。
3. 单击上方的**查看操作历史**。
   直至操作状态显示**成功**。
4. 重启RangerUserSync。
   i. 在Ranger服务页面，选择右上角的**操作 > 重启RangerUserSync**。
   ii. 在**执行集群操作**对话中，输入**执行原因**，单击**确定**。
   iii. 在**确认**对话中，单击**确定**。

   Ranger UserSync开启LDAP认证后，能够同步LDAP中的用户信息至Ranger中，以便于在配置Ranger的Policy时，可以授权LDAP中的用户或用户组访问组件。

   开启LDAP认证后，您可以在Ranger Web UI的**Settings > Users/Groups**中查看从LDAP中同步过来的用户。



   您也可以在Ranger Web UI中的**Audit > User Sync**中查看，当**Sync Source**为**LDAP/AD**时，表示同步用户来源为LDAP。



Ranger UserSync关闭LDAP认证

1. 进入Ranger页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务 > Ranger**。
2. 关闭LDAP认证。
   i. 在Ranger服务页面，选择右上角的**操作 > UserSync关闭LDAP认证**。
   ii. 在**执行集群操作**对话中，单击**确认**。
3. 单击上方的**查看操作历史**。
   直至操作状态显示**成功**。
4. 重启RangerUserSync。
   i. 在Ranger服务页面，选择右上角的**操作 > 重启RangerUserSync**。
   ii. 在**执行集群操作**对话中，输入**执行原因**，单击**确定**。
   iii. 在**确认**对话中，单击**确定**。

6.2.33.3.4. Hive数据脱敏

Ranger支持对Hive数据的脱敏处理（Data Masking），即可以对Select的返回结果脱敏，以屏蔽敏感信息。

背景信息

该功能只针对HiveServer2的场景（例如，Beeline、JDBC和Hue等途径执行的Select语句）。

配置Data Mask Policy

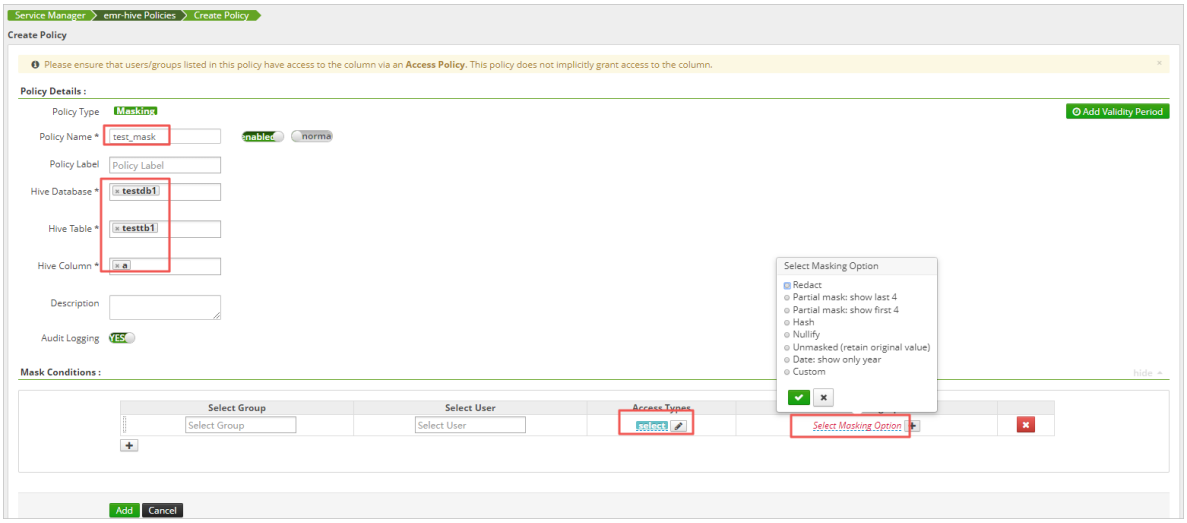
在Ranger UI配置页面的**emr-hive**页签，您可以对Hive数据进行脱敏处理：

- 支持多种脱敏处理方式。例如，显示开始的4个字符、显示最后的4个字符或Hash处理等。
- 配置Mask Policy时不支持通配符。例如Policy中Table或Column不能配置星号（*）。
- 每个Policy只能配置一个列的Mask策略，多个列时需要配置各个列的Mask Policy。

1. Hive组件配置Ranger，详情请参见文档 Hive配置。
2. 在Ranger页面，单击已创建的**emr-hive**。

3. 单击上方的**Masking**。

4. 单击右上角的**Add New Policy**。
5. 配置相关参数。



参数	说明	示例
Policy Name	Policy的名称。可以自定义。	test_mask
database	添加Hive中的数据库。	testdb1
table	添加表。	testtb1
Hive Column	可添加列名。	a
Access Types	选择授予的权限。	SELECT
Select Masking Option	选择脱敏方式。	show first 4

6. 单击add。

测试数据脱敏

- 场景：

  用户test使用 `select` 语句查看表testdb1.testtbl中a列的数据时，只显示最开始的4个字符。

- 流程：

  i. 配置Policy

  配置Policy流程请参见配置Data Mask Policy。

  ii. 脱敏验证

  test用户使用Beeline连接HiveServer2，执行 `select a from testdb1.testtbl;` 。

```
[test@emr-header-1 ~]$ /usr/lib/hive-current/bin/beeline
Beeline version 2.3.3 by Apache Hive
beeline> !connect jdbc:hive2://emr-header-1:10000
Connecting to jdbc:hive2://emr-header-1:10000
Enter username for jdbc:hive2://emr-header-1:10000: test
Enter password for jdbc:hive2://emr-header-1:10000:
Connected to: Apache Hive (version 2.3.3)
Driver: Hive JDBC (version 2.3.3)
Transaction isolation: TRANSACTION_REPEATABLE_READ
0: jdbc:hive2://emr-header-1:10000> select a from testdb1.testtbl;
+--------------------+
|         a          |
+--------------------+
| abcdxxxxxxxxxxxxx  |
+--------------------+
1 row selected (0.565 seconds)
0: jdbc:hive2://emr-header-1:10000>
```

如上图所示，test用户执行Select命令后，列a显示的数据只有前面4个字符是正常显示，后面字符全部用 `x` 脱敏处理。

6.2.33.3.5. Hive数据按行过滤

Ranger支持对Hive数据按行进行过滤（Row Level Filter），即可以对Select返回的结果按行进行过滤，只显示满足指定条件的行。本文以EMR-4.9.0版本（Ranger 2.1.0）为例，介绍如何将Hive数据按行进行过滤。

前提条件

- 已创建集群，并选择了Ranger服务，详情请参见创建集群。
- 已创建按行过滤的表。

使用限制

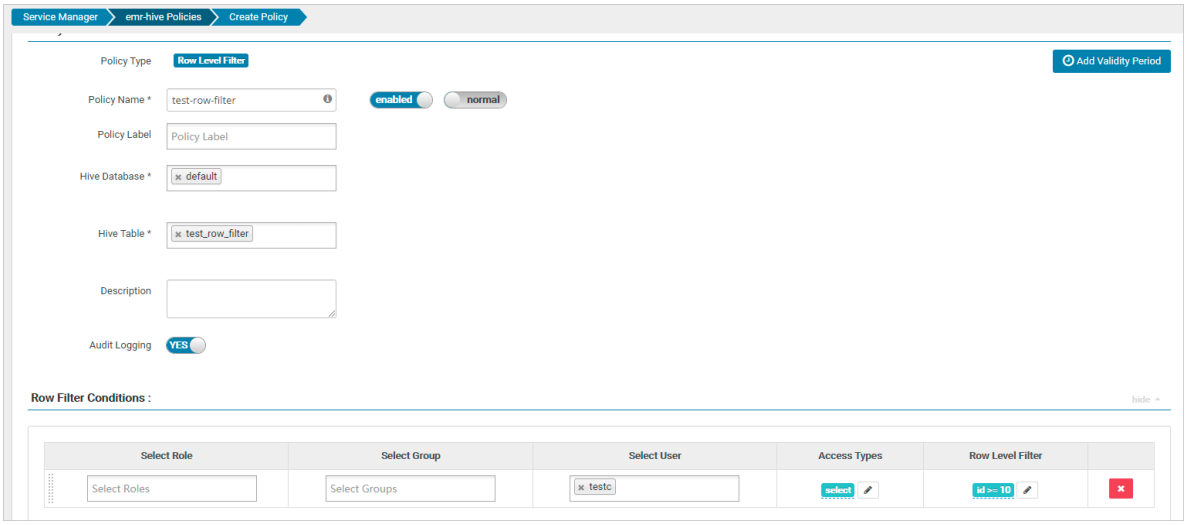
该功能仅适用于HiveServer2的场景。例如，Beeline、JDBC和Hue等途径执行的Select语句。

配置Row Level Filter Policy

1. Hive组件配置Ranger，详情请参见文档 Hive配置。
2. 在Ranger WebUI页面，单击已创建的**emr-hive**。

3. 单击上方的**Row Level Filter**。

4. 单击右上角的**Add New Policy**。
5. 在**Create Policy**页面，配置相关参数。



参数	说明	示例
Policy Name	Policy的名称。您可以自定义。	test-row-filter
Hive Database	添加Hive中的数据库。	default
Hive Table	添加表。	test_row_filter
Select User	选择配置按行过滤的用户。	testc
Access Types	选择授予的权限。	select
Row Level Filter	填写过滤的函数。	`id>=10`

6. 单击**add**。

测试

- 场景：testc用户使用 `select` 语句，查看表*default.test_row_filter*中的数据时，只显示符合过滤条件的行。
- 流程：
  i. 没有配置按行过滤且有权限访问表的用户（例如hadoop用户）使用Beeline连接HiveServer2，执行 `select * from default.test_row_filter;` 命令，查询*default.test_row_filter*表的数据，可以显示所有的行。

```
0: jdbc:hive2://emr-header-1:10000> select * from default.test_row_filter;
INFO  : Compiling command(queryId=hadoop_20210706103835_487cfe47-0d17-4fb0-a
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:test_ro
e, type:string, comment:null)], properties:null)
INFO  : Completed compiling command(queryId=hadoop_20210706103835_487cfe47-0
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Executing command(queryId=hadoop_20210706103835_487cfe47-0d17-4fb0-a
INFO  : Completed executing command(queryId=hadoop_20210706103835_487cfe47-0
INFO  : OK
INFO  : Concurrency mode is disabled, not creating a lock manager
+---------------------+-----------------------+
| test_row_filter.id  | test_row_filter.name  |
+---------------------+-----------------------+
| 1                   | a                     |
| 2                   | b                     |
| 11                  | c                     |
| 12                  | d                     |
+---------------------+-----------------------+
4 rows selected (0.309 seconds)
```

  ii. testc用户使用Beeline连接HiveServer2，执行 `select * from default.test_row_filter;` 命令，查询*default.test_row_filter*表的数据，只能显示id≥10的行。

```
0: jdbc:hive2://emr-header-1:10000> select * from default.test_row_filter;
INFO  : Compiling command(queryId=hadoop_20210706105641_b46f53c5-4af1-4741
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:test_
e, type:string, comment:null)], properties:null)
INFO  : Completed compiling command(queryId=hadoop_20210706105641_b46f53c5
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Executing command(queryId=hadoop_20210706105641_b46f53c5-4af1-4741
INFO  : Completed executing command(queryId=hadoop_20210706105641_b46f53c5
INFO  : OK
INFO  : Concurrency mode is disabled, not creating a lock manager
+---------------------+-----------------------+
| test_row_filter.id  | test_row_filter.name  |
+---------------------+-----------------------+
| 11                  | c                     |
| 12                  | d                     |
+---------------------+-----------------------+
2 rows selected (0.305 seconds)
```

6.2.33.3.6. 查看Ranger审计日志信息

EMR-3.27.0及后续版本，支持在Ranger WebUI上查看审计日志信息。本文以EMR-4.9.0版本（Ranger 2.1.0）为例，介绍如何查看Ranger审计日志信息。

前提条件

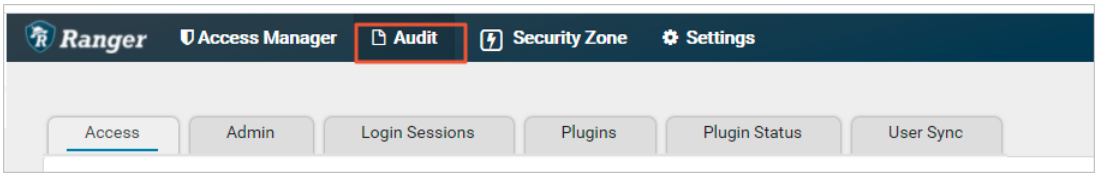
已创建集群，并选择了Ranger服务，详情请参见创建集群。

使用限制

EMR-3.27.0及后续版本，支持在Ranger WebUI上查看审计日志信息。

操作步骤

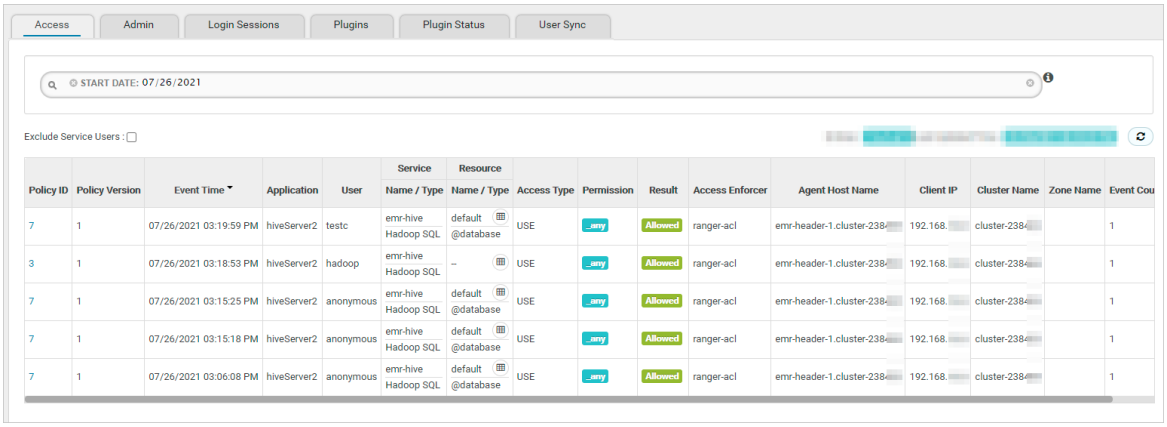
1. 访问Ranger WebUI，详情请参见访问Ranger UI。
2. 在Ranger WebUI页面，单击上方的**Audit**。



默认显示**Access**页签，支持您查看以下日志：

- Access日志

  在**Access**页签，可以查看对接Ranger的组件的访问信息。

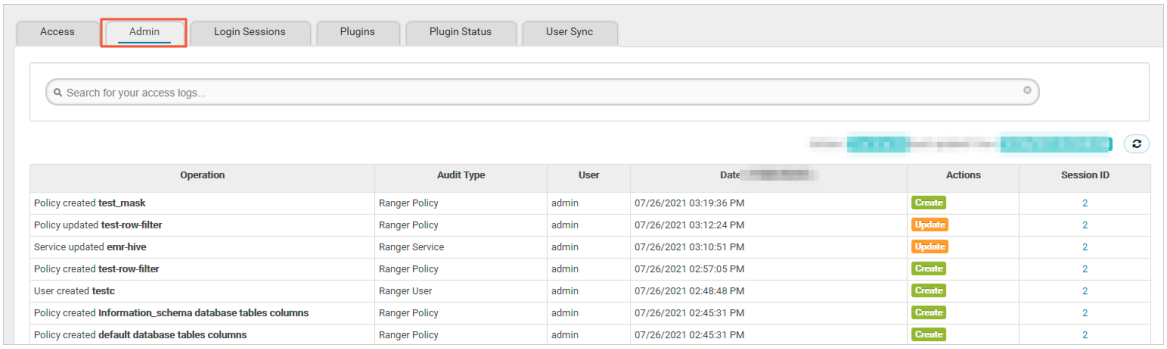


  重要参数描述如下。

参数	描述
Policy ID	访问所触发的Ranger Policy的ID。
Policy Version	访问所触发的Ranger Policy的版本。
Event Time	访问发生的时间。
User	访问的用户。
Service	访问的服务对接的Ranger Service名称以及类型。
Resource	访问的数据信息。例如，Hive的库表列和HDFS的路径等。 单击图标，可以查看具体的Query信息。
Access Type	访问的种类。
Permission	访问触发的权限。
Result	访问的结果。
Access Enforcer	访问触发的权限控制对象。例如ranger-acl（Ranger权限控制）和hadoop-acl（HDFS自带权限控制）等。 **说明** HDFS鉴权首先会查看HDFS自带权限控制（hadoop-acl），只有自带权限控制拒绝访问时才会去检验Ranger所配置的权限（ranger-acl）。您可以根据该字段判断权限控制是被hadoop-acl允许或拒绝的，还是被ranger-acl允许或拒绝的。
Agent Host Name	访问触发的Ranger Plugin的Hostname。
Client IP	访问者客户端的IP地址。

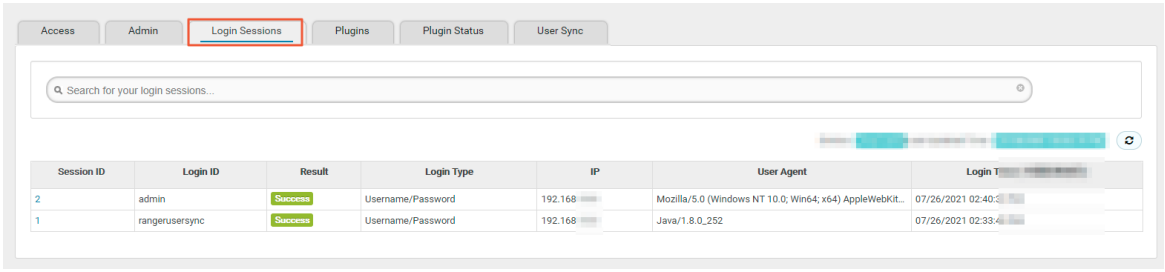
- Admin日志

单击**Admin**页签，可以查看对接Ranger的组件的访问信息。



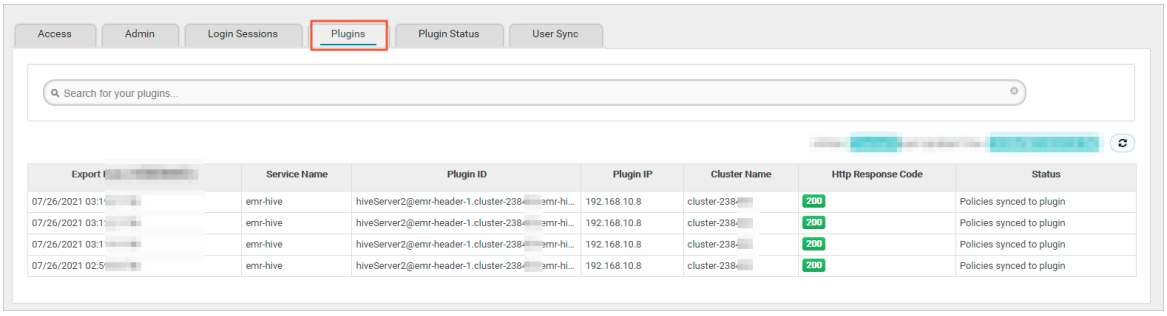
- Login Sessions日志

  单击**Login Sessions**页签，可以查看访问或登录Ranger Admin的日志信息。



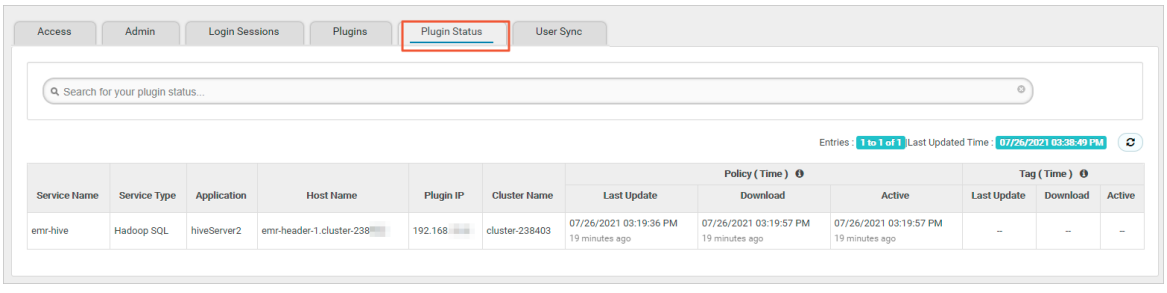
- Plugins日志

  单击**Plugins**页签，可以查看Ranger Plugin与Ranger Admin的交互信息，主要记录了Ranger Plugin同步Policy的时间信息。



- Plugin Status日志

  单击**Plugin Status**页签，可以查看每个Ranger Plugin的状态信息。



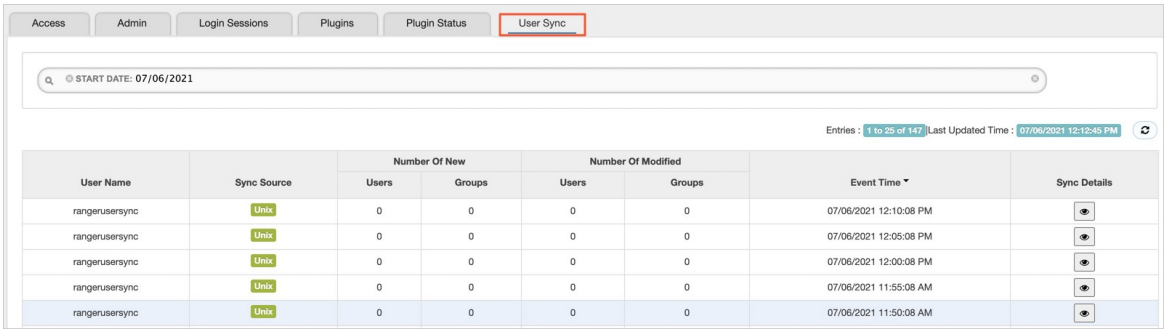
重要参数描述如下。

参数	描述
Service Name	Ranger Plugin对接的Ranger Service的名称。
Service Type	Ranger Plugin的Service种类。
Host Name	Ranger Plugin加载服务的Hostname。
Plugin IP	Ranger Plugin加载服务的IP地址。

参数	描述
Last Update	Policy最近的更新时间。
Download	Ranger Plugin最近的Policy下载时间。
Active	Ranger Plugin最近的活动时间。

- User Sync日志

  单击**User Sync**页签，可以查看Ranger UserSync服务同步用户的日志。



  重要参数描述如下。

参数	描述
Sync Source	同步的用户的来源，包括**Unix**和**LDAP/AD**。
Number Of New	同步新增的用户和用户组数量。
Number Of Modified	同步修改的用户和用户组数量。
Event Time	同步发生的时间，**Unix**用户同步通常为五分钟一次，**LDAP/AD**用户同步通常为一小时一次。
Sync Details	用户同步的详细信息。

6.2.33.3.7. Security Zone功能

Ranger 2.1.0版本开始支持配置Security Zone功能，可以将资源划分到不同的Security Zone中，给每个Security Zone分配不同的管理员进行权限管理，即可以将资源分类交由不同的管理员管理。本文以EMR-4.9.0版本（Ranger 2.1.0）为例，介绍如何创建Ranger Security Zone管理员用户，以及如何配置Security Zone。

背景信息

例如，某公司有部门A和部门B两个部门，部门A主要使用Hive database a，以及HDFS路径*/a*，部门B主要使用Hive database b，以及HDFS路径*/b*。如果要使用Security Zone功能，可以将Hive database a和HDFS路径*/a*划分到Zone a当中，将Hive database b和HDFS路径*/b*划分到Zone b当中，并分别设置管理员，统一配置Zone中资源的权限。

Zone a和Zone b配置如下。

```
Zone: a
  service: emr-hive; path=/a/*,
  service: emr-hdfs; database=a
Zone: b
  service: emr-hive; path=/b/*,
  service: emr-hdfs; database=b
```

Security Zone的管理员有权限配置Zone中资源的权限，如果配置属于其他Zone的资源的权限，则将不会生效。当Ranger Plugin鉴权时，首先会判断资源属于哪个Security Zone，然后只会使用该Security Zone中配置的权限进行鉴权。如果没有找到资源所属的Security Zone，则会使用未划分Security Zone的权限进行鉴权。

前提条件

- 已创建集群，并选择了Ranger服务，详情请参见创建集群。
- 已完成Hive配置。
- 已完成HDFS配置。

使用限制

仅EMR-4.5.x及后续版本和EMR-5.x系列版本，支持配置Security Zone功能。

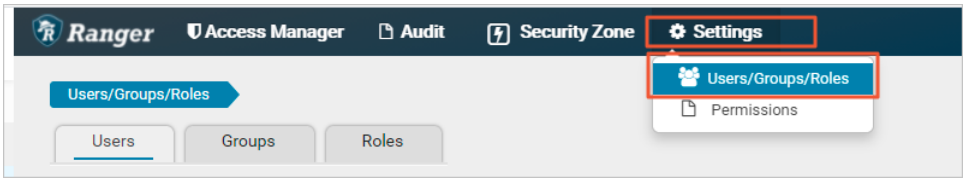
注意事项

没有划分到Security Zone中的资源的权限配置，需要取消Security Zone的选择后，再在对应的Service中配置才能生效。

单击**Security Zone**右侧的 图标，取消选中的Security Zone。

创建Security Zone管理员用户

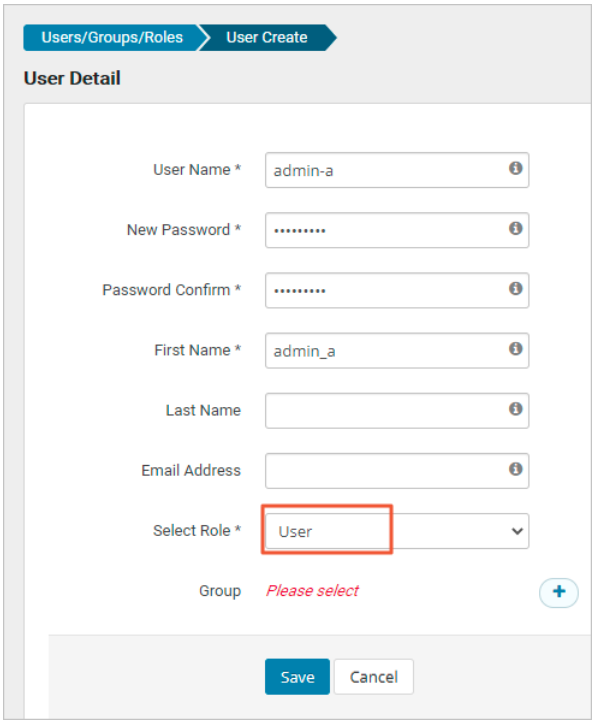
1. 访问Ranger WebUI，详情请参见访问Ranger UI。
2. 在Ranger WebUI页面上方，选择**Settings > Users/Groups/Roles**。



3. 在**User**页签，单击**Add New User**。

4. 在**User Detail**页面，填写用户信息，设置**Select Role**为**User**。

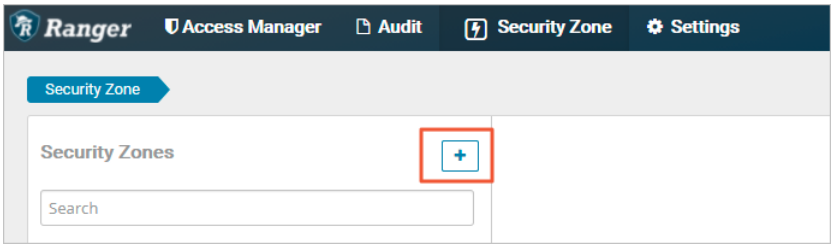
> 注意 **Select Role**必须设置为**User**，设置为**Admin**将会使得该用户成为超级管理员。超级管理员能够配置任何Security Zone的权限，就无法实现Security Zone管理员仅配置属于Zone的资源权限的功能。



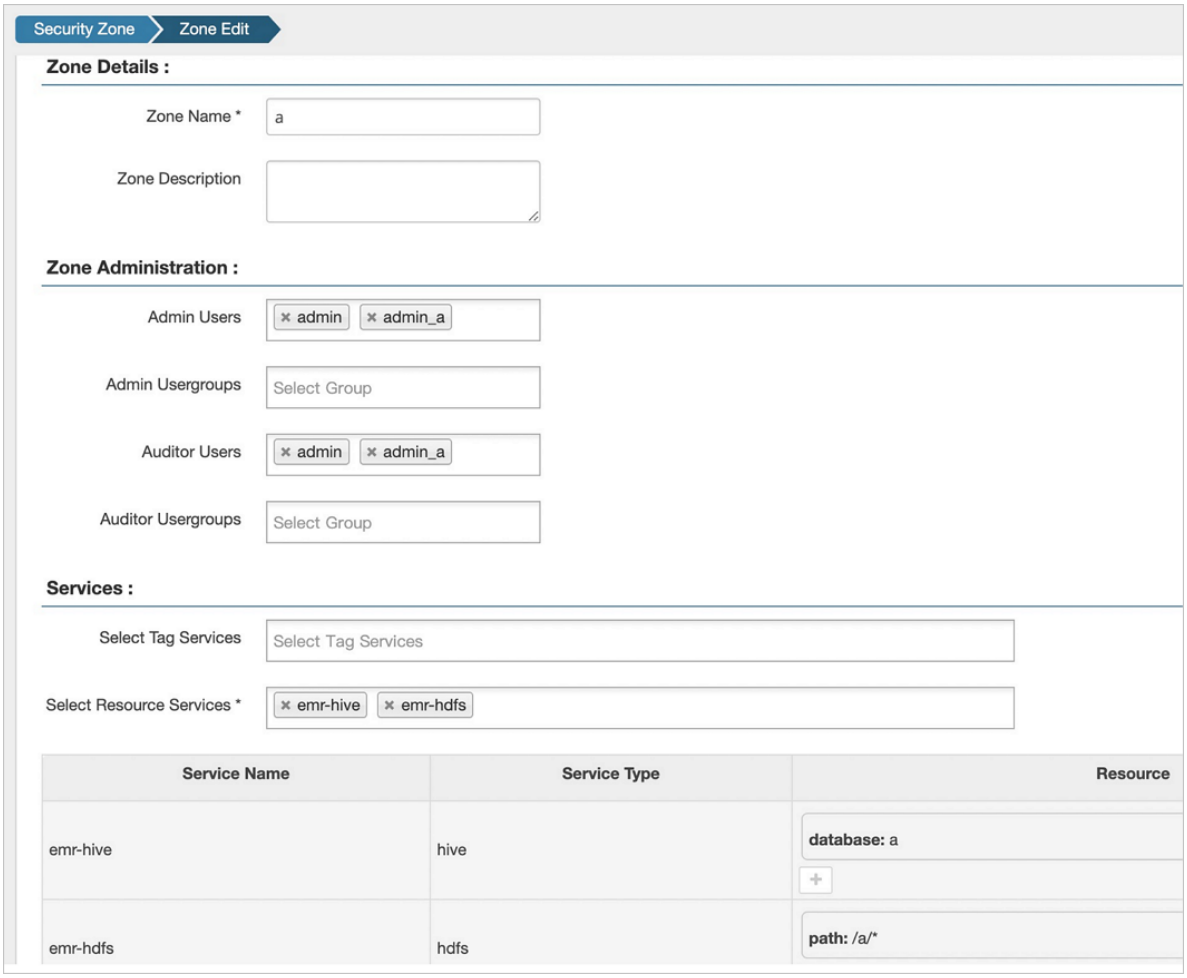
配置Security Zone

您可以按照以下步骤配置Zone a和Zone b。

1. 访问Ranger WebUI，详情请参见访问Ranger UI。
2. 在Ranger WebUI页面，单击上方的**Security Zone**。
3. 在**Security Zone**页面，单击**Security Zones**区域的 + 图标。



4. 在**Create Zone**页面，配置相关参数。

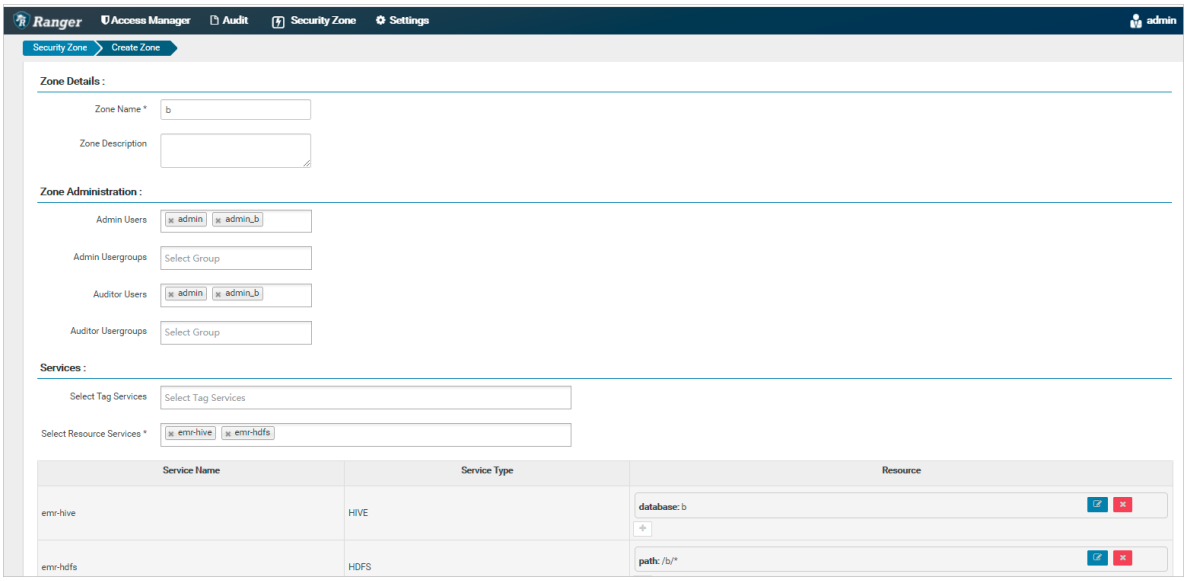


参数	描述	示例
Zone Name	Security Zone的名称。	a
Admin Users	Security Zone的管理员用户。该用户需要在Settings中进行创建，用户角色需要为User，详情请参见创建Security Zone管理员用户。	admin、admin_a
Auditor Users	允许查看Security Zone审计日志的用户。	admin、admin_a
Select Resource Services	选择Security Zone中的Service。	emr-hive、emr-hdfs
Resource	填写属于该Security Zone的资源。	database=a、path=/a/*

5. 单击**Save**。

完成Zone a的配置。

6. 重复步骤3~步骤5，完成Zone b的配置。



测试

1. 使用admin_a用户登录Ranger WebUI，详情请参见访问Ranger UI。
2. 在Ranger WebUI页面，在右上角的**Security Zone**区域，选择**a**。
3. 单击**emr-hive**，查看emr-hive Service中的权限。

   只能查看和修改Zone a中配置的权限，无法查看和修改Zone b中配置的权限。您可以配置一个属于Zone b的资源的权限，例如给test用户配置b.test表的select权限。使用Beeline进行测试，该权限配置是不会生效的，因为database=b属于Zone b，在鉴权时只会检验Zone b中配置的权限。在Zone a中配置database=b资源的权限是不会起作用的。

```
0: jdbc:hive2://emr-header-1:10000> select * from b.test;
Error: Error while compiling statement: FAILED: HiveAccessControlException Permission denied: user [test] does not
have [SELECT] privilege on [b/test/*] (state=42000,code=40000)
0: jdbc:hive2://emr-header-1:10000>
```

4. 使用admin_b用户登录Ranger WebUI，详情请参见访问Ranger UI。
5. 在Ranger WebUI页面，在右上角的**Security Zone**区域，选择**b**。
6. 单击**emr-hive**，查看emr-hive Service中的权限。

   只能查看和修改Zone b中配置的权限，无法查看和修改Zone a中配置的权限。您可以配置一个属于Zone b的资源的权限，例如给test用户配置b.test表的select权限。使用Beeline进行测试，该权限配置可以生效。

```
0: jdbc:hive2://emr-header-1:10000> select * from b.test;
INFO  : Compiling command(queryId=hadoop_20210706155423_3e80fd70-9e5c-438b-9
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:test.id
name:test.name, type:string, comment:null)], properties:null)
INFO  : Completed compiling command(queryId=hadoop_20210706155423_3e80fd70-9
: 0.122 seconds
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Executing command(queryId=hadoop_20210706155423_3e80fd70-9e5c-438b-9
INFO  : Completed executing command(queryId=hadoop_20210706155423_3e80fd70-9
: 0.001 seconds
INFO  : OK
INFO  : Concurrency mode is disabled, not creating a lock manager
+----------+------------+
| test.id  | test.name  |
+----------+------------+
| 1        | a          |
| 2        | b          |
| 3        | c          |
+----------+------------+
3 rows selected (0.227 seconds)
0: jdbc:hive2://emr-header-1:10000>
```

6.2.34. Kerberos

6.2.34.1. 概述

E-MapReduce从2.7.x和3.5.x版本开始支持创建安全类型的集群。高安全类型集群中的开源组件以Kerberos的安全模式启动，因此只有经过认证的客户端（Client）才能访问集群的服务（例如HDFS）。

背景信息

对于客户端而言，集群开启Kerberos之后，可以对可信任的客户端提供认证，使得可信任客户端能够正确提交作业，恶意用户无法伪装成其他用户侵入到集群当中，能够有效防止恶意冒充客户端提交作业的情况。对于服务端而言，集群开启Kerberos之后，集群中的服务都是可以信任的，集群服务之间使用密钥进行通信，避免了冒充服务的情况。

开启Kerberos能够提升集群的安全性，但是也会增加用户使用集群的复杂度，提交作业的方式与没有开启Kerberos前会有一些区别，需要对作业进行改造，增加Kerberos认证的相关内容。开启Kerberos前需要用户对Kerberos的原理、使用有一些了解，才能更好地使用Kerberos。此外，由于集群服务间的通信加入了Kerberos认证机制，认证过程会有一些轻微的时间消耗，相同作业相较于没有开启Kerberos的同规格集群执行速度会有一些下降。

开启Kerberos

创建集群时，在**软件配置**页面的**高级设置**区域中，打开**Kerberos集群模式**开关。

Kerberos组件列表

E-MapReduce中支持的Kerberos的组件列表如下所示。

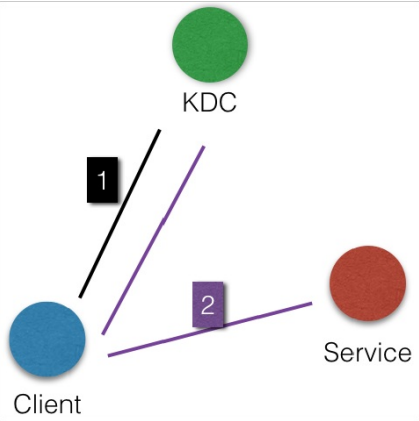
组件名称	组件版本
HDFS	2.8.5及后续版本
YARN	2.8.5及后续版本
Spark	2.4.3
Hive	2.3.5及后续版本
Tez	0.9.1及后续版本
Zookeeper	3.5.5及后续版本
Hue	4.4.0
Zeppelin	0.8.1
Oozie	5.1.0
Sqoop	1.4.7
HBase	1.4.9及后续版本
Phoenix	4.14.1及后续版本
Druid	0.13.0及后续版本
Flink	1.5.6及后续版本
Impala	2.12.2及后续版本
Kafka	2.11/1.1.1及后续版本
Presto	prestodb 0.213及以上/prestosql 310
Ranger	1.0.0及后续版本

组件名称	组件版本
Storm	1.2.2

Kerberos身份认证原理

Kerberos是一种基于对称密钥技术的身份认证协议，可以为其他服务提供身份认证功能，且支持SSO（即客户端身份认证后，可以访问多个服务，例如HBase和HDFS）。

Kerberos协议过程主要有两个阶段，第一个阶段是KDC对Client身份认证，第二个阶段是Service对Client身份认证。



- KDC：Kerberos的服务端程序。
- Client：需要访问服务的用户（Principal），KDC和Service会对用户的身份进行认证。
- Service：集成了Kerberos的服务。例如，HDFS、YARN和HBase。
- 第一阶段：KDC对Client身份认证

  当客户端用户（Principal）访问一个集成了Kerberos的服务之前，需要先通过KDC的身份认证。

  如果身份认证通过，则客户端会获取到一个TGT（Ticket Granting Ticket），后续就可以使用该TGT去访问集成了Kerberos的服务。
- 第二阶段：Service对Client身份认证

  当用户获取TGT后，就可以继续访问Service服务。

  使用TGT以及需要访问的服务名称（例如HDFS）去KDC获取SGT（Service Granting Ticket），然后使用SGT去访问Service。Service会利用相关信息对Client进行身份认证，认证通过后就可以正常访问Service服务。

Kerberos实践

E-MapReduce的Kerberos安全集群中的服务，在创建集群的时候会以Kerberos安全模式启动。

- Kerberos服务端程序为HASServer
  - 在**集群管理**页面，单击待操作集群所在行的**详情**，然后在**集群服务 > Has**页面，您可以执行查看、修改配置和重启等操作。
  - 非HA集群部署在emr-header-1节点，HA集群部署在emr-header-1和emr-header-2两个节点。
- HASServer支持以下三种身份认证方式

  HASServer可以同时支持以下三种身份认证方式，客户端可以通过配置相关参数来指定HASServer使用哪种方式进行身份认证。
  - 兼容MIT Kerberos的身份认证方式

    客户端配置：
    - 如果在集群的某个节点上执行客户端命令，则需要将*/etc/ecm/hadoop-conf/core-site.xml*中的hadoop.security.authentication.use.has设置为false。
    - 如果通过控制台运行作业，您可以使用 `export HADOOP_CONF_DIR=/etc/has/hadoop-conf` 命令，临时Export环境变量，设置hadoop.security.authentication.use.has为false。

    访问方式：Service的客户端包完全可使用开源的。例如HDFS客户端。详情请参见兼容MIT Kerberos认证。

- RAM身份认证

  客户端配置：

  - 如果在集群的某个节点上执行客户端命令，则需要将*/etc/ecm/hadoop-conf/core-site.xml*中的hadoop.security.authentication.use.has设置为false，*/etc/has/has-client.conf*中的auth_type设置为RAM。
  - 如果有通过控制台运行作业，您可以 `export HAS_CONF_DIR=/path/to/has-client.conf` 临时Export环境变量，设置*has-client.conf*的auth_type为RAM。

  访问方式： 客户端需要使用集群中的软件包（例如Hadoop和HBase），详情请参见RAM认证。

- 数据开发认证

  如果您使用E-MapReduce控制台的数据开发提交作业，则不能修改emr-header-1节点的配置，即使用默认配置。

  客户端配置：

  在emr-header-1节点上，设置*/etc/ecm/hadoop-conf/core-site.xml*中hadoop.security.authentication.use.has为true，*/etc/has/has-client.conf*中auth_type设置为EMR。

  访问方式：跟非Kerberos安全集群使用方式一致，详情请参见数据开发认证。

- 其他

  登录Master节点访问集群

  集群管理员也可以登录Master节点访问集群服务，登录Master节点切换到has账号（默认使用兼容MIT Kerberos的方式），即可访问集群服务，例如排查问题或者运维等。

```
sudo su has
hadoop fs -ls /
```

> **说明** 您也可以登录其他账号操作集群，前提是该账号可以通过Kerberos认证。另外，如果在Master节点上需要使用兼容MIT Kerberos的方式，需要在该账号下先Export一个环境变量 `export HADOOP_CONF_DIR=/etc/has/hadoop-conf/` 。

6.2.34.2. EMR对接外部的MIT Kerberos

E-MapReduce（简称EMR）支持对接用户自建的外部MIT Kerberos，可以将普通集群切换成Kerberos集群（高安全集群）。本文为您介绍如何在EMR上对接外部的MIT Kerberos。

前提条件

- 已在EMR上创建普通集群，详情请参见创建集群。
- 已获取正确的KDC的IP地址，Kadmin的IP地址以及Principal的名称和密码。

> **说明** 请确保获取的内容正确，否则在对接过程中会出现错误。

注意事项

- 对接外部MIT Kerberos功能，仅支持普通集群切换为Kerberos集群（高安全集群）的场景。
- 需要确保EMR集群与自建外部MIT Kerberos间网络和端口的连通性。例如，TCP 88端口、749端口和UDP 88端口。

  您可以在对接前，先在EMR集群的header-1节点先测试好连通性，然后再进行对接。
- 集群对接过程中所有服务都会重启，会影响集群正在运行的作业。

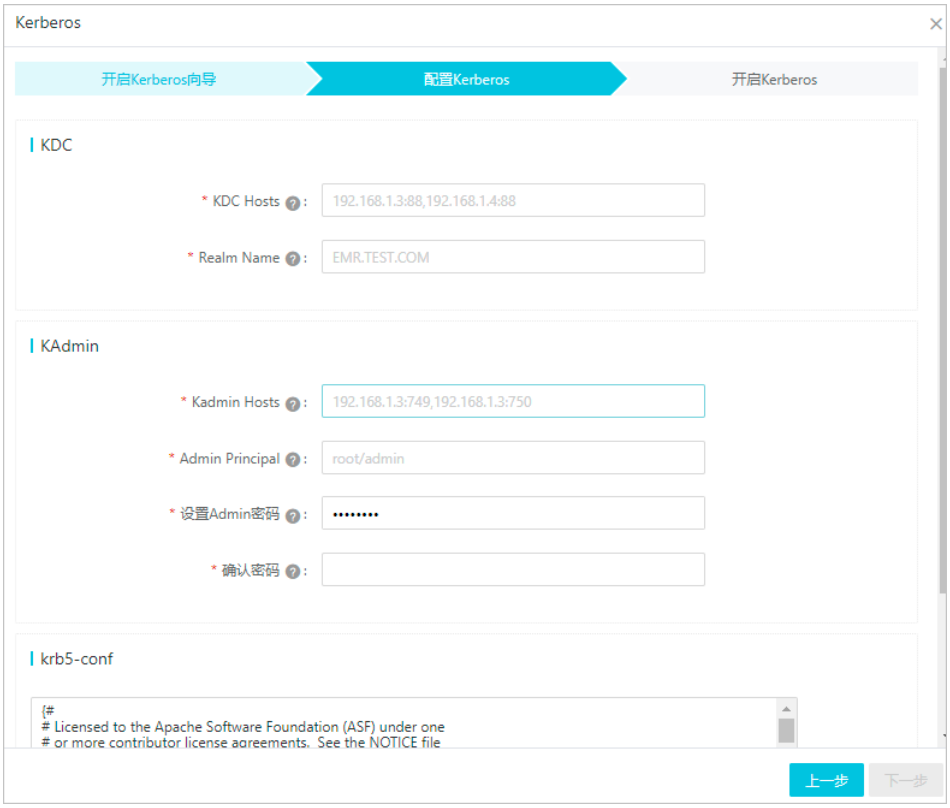
  集群对接过程中会先停止所有服务，并连接用户自建的MIT Kerberos，创建出集群服务所需的Kerberos Principal并导出keytab文件到集群上，自动配置集群上支持Kerberos的服务，然后启动集群上的所有服务。

操作步骤

1. 进入管理页面。
   i.
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击待操作的集群ID。
2. 在**集群与服务管理**页面，单击右上角的**开启Kerberos**。
3. 在Kerberos对话框中，执行以下操作。
   i. 在**开启Kerberos向导**中，单击**MIT KDC**。

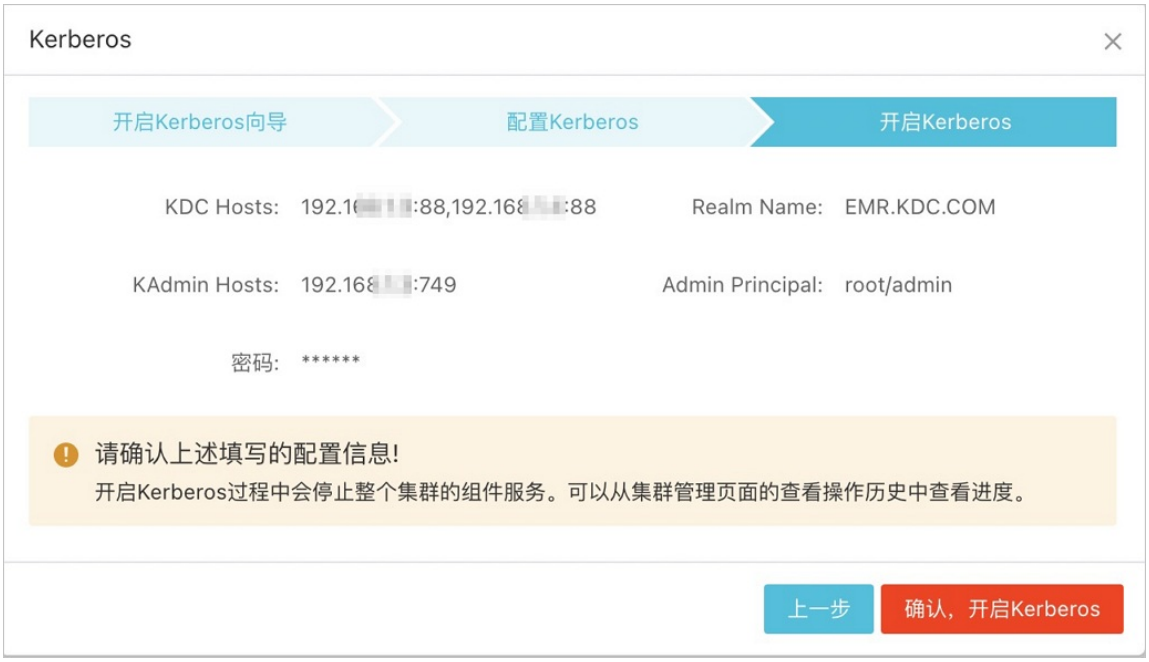
ii. 勾选图中事项，并单击**下一步**。

iii. 在**配置Kerberos**中，配置如下参数，单击**下一步**。



参数	描述
KDC Hosts	KDC的IP地址和端口。 多个IP地址时，使用英文逗号（,）隔开。例如，192.168.**.**:88,192.168.**.**:88。 **注意** 确保EMR集群与KDC地址及端口的连通性。
Realm Name	KDC Realm名称。
Kadmin Hosts	Kadmin服务的IP地址和端口。 多个IP地址时，使用英文逗号（,）隔开。例如，192.168.**.**:749。 **注意** 确保EMR集群对Kadmin服务地址及端口的连通性。
Admin Principal	用于连接Kadmin服务的Principal名称。 **注意** 确保该Principal具有admin权限，能够创建Principal和导出*keytab*文件。建议您使用*root/admin*。
设置Admin密码	Principal对应的密码。
确认密码	
krb5-conf	*krb5.conf*模板文件，没有特殊需求无需修改。

iv. 在**开启Kerberos**中，确认信息无误后，单击**确认，开启Kerberos**。



您可以在集群管理页面，单击右上角的**查看操作历史**，待所有操作的**状态**显示为**成功**时，再对集群进行其他操作。

6.2.34.3. 兼容MIT Kerberos认证

本文通过访问HDFS服务为您介绍如何兼容MIT Kerberos认证。

前提条件

已创建集群，详情请参见创建集群。

通过hadoop命令访问HDFS

以test用户访问HDFS服务为例介绍。

1. 在Gateway节点配置*krb5.conf*文件。

```
scp root@emr-header-1:/etc/krb5.conf /etc/
```

2. 配置**hadoop.security.authentication.use.has**的值为**false**。
   i. 登录集群的emr-header-1节点。

   > **说明** HA集群也需要登录emr-header-1节点。

   登录详情请参见登录集群。
   ii. 执行以下命令，编辑*core-site.xml*文件。

```
vim /etc/ecm/hadoop-conf/core-site.xml
```

   iii. 查看hadoop.security.authentication.use.has的值。
   - 如果值为true，修改hadoop.security.authentication.use.has的值为false。
   - 如果值为false，直接执行步骤3。

3. 添加Principal。
   i. 执行如下命令，进入Kerberos的admin工具。
   - EMR-3.30.0及后续版本和EMR-4.5.1及后续版本：

```
sh /usr/lib/has-current/bin/admin-local.sh /etc/ecm/has-conf -k /etc/ecm/has-conf/admin.keytab
```

   - EMR-3.30.0之前版本和EMR-4.5.1之前版本：

```
sh /usr/lib/has-current/bin/hadmin-local.sh /etc/ecm/has-conf -k /etc/ecm/has-conf/admin.keytab
```

ii. 执行如下命令，添加test的Principal。

本示例密码设置为123456。

```
addprinc -pw 123456 test
```

iii. 执行如下命令，导出keytab文件。

```
ktadd -k /root/test.keytab test
```

4. 获取Ticket。

在待执行HDFS命令的客户端机器上执行命令，本文以Gateway节点为例介绍。

i. 添加Linux账号test。

```
useradd test
```

ii. 安装MIT Kerberos客户端工具。

您可以使用MIT Kerberos工具进行相关操作（例如，kinit和klist），详情请参见MIT Kerberos。

```
yum install krb5-libs krb5-workstation -y
```

iii. 切到test账号执行kinit。

```
su test
```

- 如果没有keytab文件，则执行 `kinit` ，回车后输入test的密码123456。
- 如果有keytab文件，则执行如下命令。

```
#使用指定Keytab文件中的指定Principal进行认证。
kinit -kt test.keytab test
#查看ticket的生命周期。
klist
```

执行 `klist` ，回车后输入test的密码123456。返回类似如下信息：

```
Valid starting       Expires              Service principal
03/30/2021 10:48:47  03/31/2021 10:48:47  krbtgt/EMR.209749.COM@EMR.209749.COM
        renew until 03/31/2021 10:48:47
```

iv. （可选）如果需要设置ticket的生命周期，您可以执行如下操作：

a. 设置ticket的生命周期。

```
kinit -l 5d
```

b. 执行 `klist` 命令，查看ticket的生命周期。

```
Valid starting       Expires              Service principal
03/30/2021 10:50:51  04/04/2021 10:50:51  krbtgt/EMR.209749.COM@EMR.209749.COM
        renew until 04/01/2021 10:50:51
```

5. 在Gateway节点上执行以下命令，导入环境变量。

```
export HADOOP_CONF_DIR=/etc/has/hadoop-conf
```

6. 执行HDFS命令。

```
hadoop fs -ls /
```

返回如下类似信息。

```
Found 6 items
drwxr-xr-x   - hadoop    hadoop          0 2021-03-29 11:16 /apps
drwxrwxrwx   - flowagent hadoop          0 2021-03-29 11:18 /emr-flow
drwxr-x---   - has       hadoop          0 2021-03-29 11:16 /emr-sparksql-udf
drwxrwxrwt   - hadoop    hadoop          0 2021-03-29 11:17 /spark-history
drwxr-x---   - hadoop    hadoop          0 2021-03-29 11:16 /tmp
drwxrwxrwt   - hadoop    hadoop          0 2021-03-29 11:17 /user
```

> **说明** 当您需要运行YARN作业时，请提前在集群中所有节点添加对应的Linux账号，详情请参见RAM认证。

通过Java代码访问HDFS

- 使用本地ticket cache

> **说明** 需要提前执行kinit获取ticket，且ticket过期后程序会访问异常。

```
public static void main(String[] args) throws IOException {
   Configuration conf = new Configuration();
   //加载hdfs的配置，需要从EMR集群上复制hdfs的配置。
   conf.addResource(new Path("/etc/ecm/hadoop-conf/hdfs-site.xml"));
   conf.addResource(new Path("/etc/ecm/hadoop-conf/core-site.xml"));
   //需要在Linux账号下，提前通过kinit获取ticket。
   UserGroupInformation.setConfiguration(conf);
   UserGroupInformation.loginUserFromSubject(null);
   FileSystem fs = FileSystem.get(conf);
   FileStatus[] fsStatus = fs.listStatus(new Path("/"));
   for(int i = 0; i < fsStatus.length; i++){
       System.out.println(fsStatus[i].getPath().toString());
   }
}
```

- 使用keytab文件（推荐）

> **说明** keytab长期有效，跟本地ticket无关。

```
public static void main(String[] args) throws IOException {
  String keytab = args[0];
  String principal = args[1];
  Configuration conf = new Configuration();
  //加载hdfs的配置，需要从EMR集群上复制hdfs的配置。
  conf.addResource(new Path("/etc/ecm/hadoop-conf/hdfs-site.xml"));
  conf.addResource(new Path("/etc/ecm/hadoop-conf/core-site.xml"));
  //直接使用keytab文件，该文件从EMR集群emr-header-1上执行相关命令获取。
  UserGroupInformation.setConfiguration(conf);
  UserGroupInformation.loginUserFromKeytab(principal, keytab);
  FileSystem fs = FileSystem.get(conf);
  FileStatus[] fsStatus = fs.listStatus(new Path("/"));
  for(int i = 0; i < fsStatus.length; i++){
      System.out.println(fsStatus[i].getPath().toString());
  }
  }
```

pom依赖如下所示。

```
<dependencies>
  <dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-common</artifactId>
    <version>x.x.x</version>
  </dependency>
  <dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-hdfs</artifactId>
    <version>x.x.x</version>
   </dependency>
</dependencies>
```

> **说明** `x.x.x` 为您集群的hadoop版本。

6.2.34.4. 跨域互信

E-MapReduce中的HAS Kerberos在配置跨域互信（Cross Realm）后支持跨域访问，即不同Kerberos集群之间可以互相访问。

使用限制

EMR-3.37.1及后续版本和EMR-5.3.1及后续版本的集群，HAS Kerberos才支持跨域互信。

操作流程

1. 步骤一：工作准备
2. 步骤二：添加跨域认证Principal
3. 步骤三：配置Cluster-A的krb5.conf

4. 步骤四：访问Cluster-B服务

步骤一：工作准备

本文以Cluster-A跨域去访问Cluster-B中的服务为例。配置完成后，Cluster-A在获取到本集群KDC授予的TGT（Ticket Granting Ticket）后，能够跨域访问Cluster-B中的服务。本文配置的跨域互信是单向的，即Cluster-B无法跨域访问Cluster-A上的服务，如果需要实现双向跨域互信，按照同样的方法交换配置即可。

在两个集群在emr-header-1节点上，执行 `hostname` 命令获取hostname。在emr-header-1节点的*/etc/krb5.conf*文件中获取realm。本文使用的两个集群信息示例如下：

- Cluster-A的相关信息：
  - hostname：emr-header-1.cluster-1234。
  - realm：EMR.1234.COM。
- Cluster-B的相关信息：
  - hostname：emr-header-1.cluster-6789。
  - realm：EMR.6789.COM。

步骤二：添加跨域认证Principal

1. 使用SSH方式登录到集群Cluster-A，详情请参见登录集群。
2. 使用root用户，在集群Cluster-A的emr-header-1节点执行以下命令。

   ```
   sh /usr/lib/has-current/bin/admin-local.sh /etc/ecm/has-conf -k /etc/ecm/has-conf/admin.keytab
   admin.local: addprinc -pw 123456 krbtgt/EMR.6789.COM@EMR.1234.COM
   ```

   上面命令涉及的参数如下：
   - `123456`：是初始密码，您可以自定义。
   - `EMR.1234.COM`：Cluster-A的realm。
   - `EMR.6789.COM`：Cluster-B的realm。
3. 在集群Cluster-B的emr-header-1节点，重复上述步骤~，添加跨域认证Principal。

步骤三：配置Cluster-A的krb5.conf

1. 执行以下命令，修改Cluster-A集群上*krb5.conf*的配置信息。

   ```
   vim /etc/krb5.conf
   ```

   配置 `[realms]`、`[domain_realm]` 和 `[capaths]`。示例如下。

   ```
   [libdefaults]
       kdc_realm = EMR.1234.COM
       default_realm = EMR.1234.COM
       udp_preference_limit = 4096
       kdc_tcp_port = 88
       kdc_udp_port = 88
       dns_lookup_kdc = false
   [realms]
       EMR.1234.COM = {
           kdc = 10.81.**.**:88
       }
       EMR.6789.COM = {
           kdc = 10.81.**.**:88
       }
   [domain_realm]
       .cluster-1234 = EMR.1234.COM
       .cluster-6789 = EMR.6789.COM
   [capaths]
       EMR.1234.COM = {
          EMR.6789.COM = .
       }
       EMR.6789.COM = {
          EMR.1234.COM = .
       }
   ```

   > **说明** kdc参数值中的 `10.81.**.**` 为HAS KDC所在节点的内网IP地址。

2. 同步修改好的*krb5.conf*配置信息至Cluster-A所有节点。

3. 拷贝Cluster-B集群节点*/etc/hosts*中的信息（只需要长域名 `emr-xxx-x.cluster-xxx` ）至Cluster-A集群所有节点的*/etc/hosts*文件中。

```
10.**.**.**  emr-worker-1.cluster-xxx
10.**.**.**  emr-worker-2.cluster-xxx
10.**.**.**  emr-header-1.cluster-xxx
```

> **说明**
> - 如果Cluster-A上需要运行作业访问Cluster-B，则需要先重启YARN。
> - 在Cluster-A的所有节点上，配置Cluster-B的host绑定信息。

步骤四：访问Cluster-B服务

在Cluster-A上，您可以使用Cluster-A的Kerberos keytab文件，访问Cluster-B的服务。

例如，访问Cluster-B的HDFS服务。创建测试需要的Principal及导出keytab文件，详情请参见兼容MIT Kerberos认证。下面以test用户的keytab为例介绍。

```
kinit -kt test.keytab test@EMR.1234.COM
hadoop fs -ls hdfs://emr-header-1.cluster-6789:9000/
Found 6 items
drwxr-xr-x   - hadoop    hadoop          0 2021-08-27 10:10 hdfs://emr-header-1.cluster-6789:9000/apps
drwxrwxrwt   - hadoop    hadoop          0 2021-08-27 10:10 hdfs://emr-header-1.cluster-6789:9000/spark-history
drwxrwxrwt   - hadoop    hadoop          0 2021-08-27 10:11 hdfs://emr-header-1.cluster-6789:9000/tmp
drwxrwxrwt   - hadoop    hadoop          0 2021-08-27 10:11 hdfs://emr-header-1.cluster-6789:9000/user
```

6.2.34.5. RAM认证

E-MapReduce（简称EMR）集群中的Kerberos服务端除了支持第一种兼容MIT Kerberos的使用方式，也支持Kerberos客户端使用RAM用户作为身份信息进行身份认证。

RAM身份认证

RAM产品可以创建或管理RAM用户，通过RAM用户实现对云上各个资源的访问控制。

阿里云账号的管理员可以在RAM的用户管理界面创建一个RAM用户（RAM用户名称必须符合Linux用户的规范），然后将RAM用户的AccessKey下载下来提供给该RAM用户对应的开发人员，后续开发人员可以配置AccessKey，从而通过Kerberos认证访问集群服务。

下面以已经创建的RAM用户test在Gateway访问为例：

- EMR集群添加test账号。

  EMR的安全集群的Yarn使用了LinuxContainerExecutor，如果需要在集群上运行Yarn作业，则需要在集群所有节点上添加运行作业的用户账号，LinuxContainerExecutor执行程序过程中会根据用户账号进行相关的权限校验。

  EMR集群管理员在EMR集群的Master节点上执行如下命令。

  ```
  sudo su hadoop
  sh adduser.sh test 1 2
  ```

  *adduser.sh*代码如下：

  ```
  #添加的账户名称。
   user_name=$1
   #集群Master节点个数，例如HA集群有2个Master节点。
   master_cnt=$2
   #集群Worker节点个数。
   worker_cnt=$3
   for((i=1;i<=$master_cnt;i++))
   do
     ssh -o StrictHostKeyChecking=no emr-header-$i sudo useradd $user_name
   done
   for((i=1;i<=$worker_cnt;i++))
   do
     ssh -o StrictHostKeyChecking=no emr-worker-$i sudo useradd $user_name
  done
  ```

- 在Gateway机器上添加test用户。

  ```
  useradd test
  ```

- 配置Kerberos基础环境。

```
sudo su root
sh config_gateway_kerberos.sh 10.27.230.10 /pathto/emrheader1_pwd_file
 #确保Gateway上面/etc/ecm/hadoop-conf/core-site.xml中值为true。
 <property>
    <name>hadoop.security.authentication.use.has</name>
    <value>true</value>
 </property>
```

*config_gateway_kerberos.sh*代码如下：

```
#EMR集群的emr-header-1的IP地址。
 masterip=$1
 #保存了masterip对应的root登录密码文件。
 masterpwdfile=$2
 if ! type sshpass >/dev/null 2>&1; then
    yum install -y sshpass
fi
  ## Kerberos conf
 sshpass -f $masterpwdfile scp root@$masterip:/etc/krb5.conf /etc/
 mkdir /etc/has
 sshpass -f $masterpwdfile scp root@$masterip:/etc/has/has-client.conf /etc/has
 sshpass -f $masterpwdfile scp root@$masterip:/etc/has/truststore /etc/has/
 sshpass -f $masterpwdfile scp root@$masterip:/etc/has/ssl-client.conf /etc/has/
 #修改Kerberos客户端配置，将默认的auth_type从EMR改为RAM。
 #也可以手工修改该文件。
 sed -i 's/EMR/RAM/g' /etc/has/has-client.conf
```

- 使用test用户登录Gateway，配置AccessKey。

```
 #登录Gateway的test账号。
 sh add_accesskey.sh test
```

*add_accesskey.sh*代码如下：

```
user=$1
 if [[ `cat /home/$user/.bashrc | grep 'export AccessKey'` == "" ]];then
 echo "
 #修改为test用户的AccessKey ID和AccessKey Secret。
 export AccessKeyId=YOUR_AccessKeyId
 export AccessKeySecret=YOUR_AccessKeySecret
 " >>~/.bashrc
 else
    echo $user AccessKey has been added to .bashrc
 fi
```

- 使用test用户，执行如下命令：
  i. 执行HDFS命令。

  ```
  hadoop fs -ls /
  ```

  返回信息如下：

  ```
  17/11/19 12:32:15 INFO client.HasClient: The plugin type is: RAM
  Found 4 items
  drwxr-x---   - has    hadoop          0 2017-11-18 21:12 /apps
  drwxrwxrwt   - hadoop hadoop          0 2017-11-19 12:32 /spark-history
  drwxrwxrwt   - hadoop hadoop          0 2017-11-18 21:16 /tmp
  drwxrwxrwt   - hadoop hadoop          0 2017-11-18 21:16 /user
  ```

  ii. 运行Hadoop作业。

  ```
  hadoop jar /usr/lib/hadoop-current/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.2.jar pi 10 1
  ```

  iii. 运行Spark作业。

  ```
  spark-submit --conf spark.ui.view.acls=* --class org.apache.spark.examples.SparkPi --master yarn-client --driver-memo
  ry 512m --num-executors 1 --executor-memory 1g --executor-cores 2 /usr/lib/spark-current/examples/jars/spark-examples
  _2.11-2.1.1.jar 10
  ```

6.2.34.6. 数据开发认证

E-MapReduce集群支持数据开发认证。您可以通过阿里云账号授权RAM用户，进行数据开发的访问。

阿里云账号访问

在阿里云账号登录E-MapReduce控制台的情况下，在数据开发页面执行作业，提交作业至安全集群上执行，以 `hadoop` 用户访问作业中涉及的开源组件服务。

RAM用户访问

在RAM用户登录E-MapReduce控制台的情况下，在数据开发页面执行作业，提交作业至安全集群上执行，以RAM用户对应的用户名访问作业中涉及的开源组件服务。

示例

通过阿里云账号授权RAM用户的数据开发示例如下：

1. 阿里云账号管理员根据需求创建多个RAM用户（例如A、B和C）。

   在RAM控制台页面给RAM用户授予AliyunEMRFullAccess和AliyunEMRDevelopAccess的权限后，RAM用户就能正常登录并使用E-MapReduce控制台上的相关功能。授权详情请参见为RAM用户授权。

2. 阿里云账号管理员将RAM用户提供给相关开发人员。
3. 您已完成作业或工作流的创建，提交作业或工作流到集群运行，最终在集群上面以该RAM用户对应的用户名（例如A、B和C）去访问相关的组件服务。

   > **说明** 以 `hadoop` 账号周期调度工作流。

4. 组件服务使用RAM用户的用户名进行相关的权限控制。

6.2.35. Druid

6.2.35.1. 概述

Apache Druid是一个分布式内存实时分析系统，用于解决如何在大规模数据集下快速的、交互式的查询和分析问题。

基本特点

Apache Druid具有如下特点：

- 支持亚秒级的交互式查询。例如，多维过滤、Ad-hoc的属性分组和快速聚合数据。
- 支持实时的数据消费。
- 支持多租户同时在线查询。
- 支持PB级数据、千亿级事件快速处理，支持每秒数千查询并发。
- 支持高可用，并且滚动升级。

应用场景

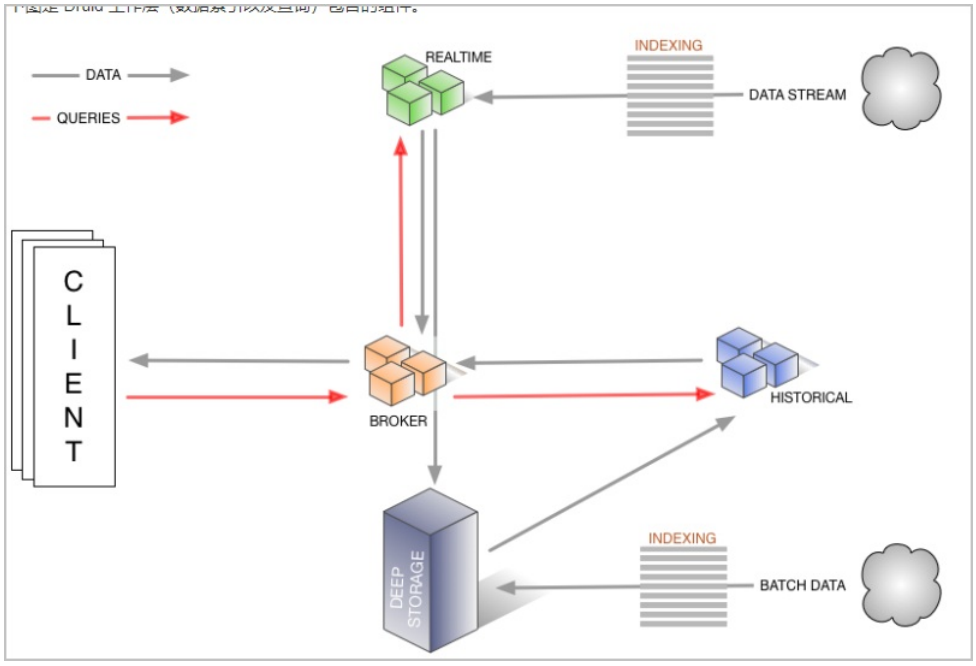
实时数据分析是Apache Druid最典型的使用场景。该场景涵盖的面很广，例如：

- 实时指标监控
- 推荐模型
- 广告平台
- 搜索模型

Apache Druid架构

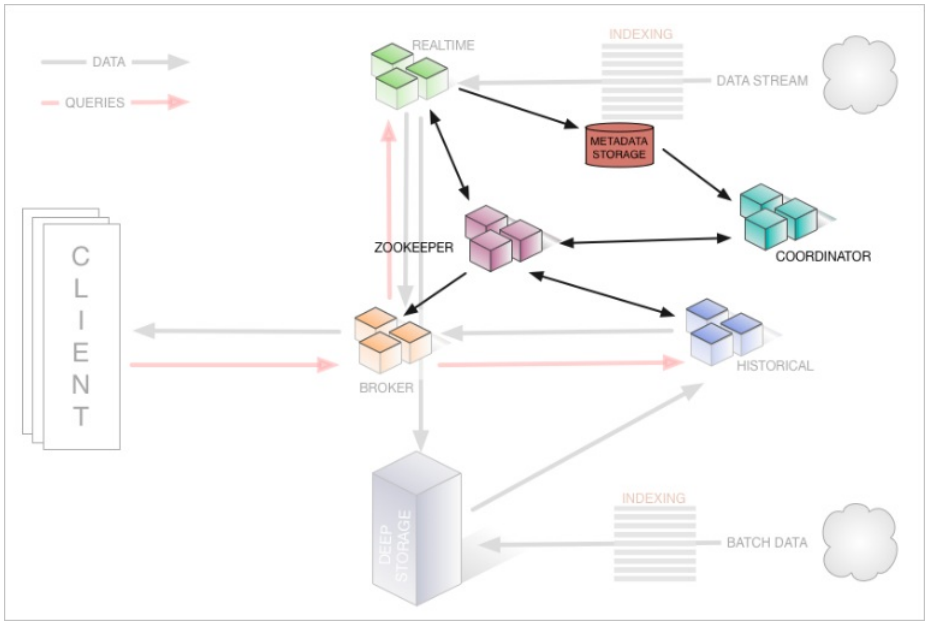
Apache Druid拥有优秀的架构设计，多个组件协同工作，共同完成数据从摄取到索引、存储和查询等一系列流程。

下图是Druid工作层（数据索引以及查询）包含的组件。



- Realtime组件负责数据的实时摄入。
- Broker组件负责查询任务的分发以及查询结果的汇总，并将结果返回给用户。
- Historical组件负责索引后的历史数据的存储，数据存储在 deep storage。Deep storage可以是本地，也可以是HDFS等分布式文件系统。
- Indexing service包含两个组件（图中未画出）。
  - Overlord组件负责索引任务的管理、分发。
  - MiddleManager负责索引任务的具体执行。

下图是Druid segments（Druid 索引文件）管理层所涉及的组件。



- Zookeeper 负责存储集群的状态以及作为服务发现组件，例如集群的拓扑信息、overlord leader的选举、indexing task的管理等。
- Coordinator负责segments的管理，如segments下载、删除以及如何在historical之间做均衡等等。
- Metadata storage负责存储segments 的元信息，以及管理集群各种各样的持久化或临时性数据，例如配置信息、审计信息等等。

E-MapReduce增强型Druid

E-MapReduce Druid基于Apache Druid做了大量的改进，包括与E-MapReduce和阿里云周边生态的集成、方便的监控与运维支持、易用的产品接口等，实现即买即用和7*24免运维。

E-MapReduce Druid目前支持的特性如下所示：

- 支持以OSS作为deep storage。
- 支持将OSS文件作为批量索引的数据来源。
- 支持从日志服务（Log Service）流式地索引数据（类似于Kafka），并提供高可靠保证和exactly-once语义。
- 支持将元数据存储到RDS。
- 集成了Superset工具。
- 方便地扩容和缩容（缩容针对Task节点）。
- 丰富的监控指标和告警规则。
- 故障迁移。
- 具有高安全性。
- 支持HA。

6.2.35.2. 使用Druid

EMR-3.11.0及其后续版本，E-MapReduce支持Druid作为单独的一种集群类型。

背景信息

E-MapReduce将Druid作为单独的集群类型，主要基于以下几方面的考虑：

- E-MapReduce Druid可以完全脱离Hadoop来使用。
- 大数据量情况下，E-MapReduce Druid对内存要求比较高，尤其是Broker和Historical节点。E-MapReduce Druid本身资源不受YARN管控，在多服务运行时容易发生资源抢夺。
- Hadoop作为基础设施，其规模通常较大，而E-MapReduce Druid集群较小，部署在同一集群上，由于规模不一致可能造成资源浪费，所以单独部署会更加灵活。

创建Druid集群

创建集群时选择Druid集群类型即可，详情请参见创建集群。

> **说明** E-MapReduce Druid集群自带的HDFS和YARN仅供测试使用。对于生产环境，建议您使用专门的Hadoop集群。

配置集群

- 配置HDFS作为E-MapReduce Druid的Deep Storage。

  对于独立的E-MapReduce Druid集群，如果您需要存放索引数据至一个Hadoop集群的HDFS，请设置两个集群的连通性（详情请参见与Hadoop集群交互）。

  在E-MapReduce Druid**配置**页面的**common.runtime**页签，配置如下参数。

参数	描述
druid.storage.type	设置为hdfs。
druid.storage.storageDirectory	HDFS目录，建议填写完整目录。例如*hdfs://emr-header-1.cluster-xxxxxxxx:9000/druid/segments*。

> **说明** 如果Hadoop集群为HA集群，emr-header-1.cluster-xxxxx:9000需要改成emr-cluster，或者把端口9000改成8020。

- 配置OSS作为E-MapReduce Druid的Deep Storage。

  在E-MapReduce Druid**配置**页面的**common.runtime**页签，配置如下参数。

参数	描述
druid.storage.type	设置为hdfs。
druid.storage.storageDirectory	OSS目录。例如*oss://emr-druid-cn-hangzhou/segments*。

- 配置RDS作为E-MapReduce Druid的元数据存储。

  默认情况下E-MapReduce Druid利用emr-header-1节点上的本地MySQL数据库作为元数据存储。您也可以配置使用阿里云RDS作为元数据存储。

  i. 本示例以RDS MySQL版为例介绍。在具体配置之前，请先确保：
     - 已创建RDS MySQL实例。
     - 为E-MapReduce Druid访问RDS MySQL创建了单独的账户（不推荐使用root）。例如账户名为druid，密码为druidpw。

- 为E-MapReduce Druid元数据创建单独的MySQL数据库。例如数据库名为druiddb。
- 确保账户druid有权限访问druiddb。

ii. 在E-MapReduce Druid**配置**页面的**common.runtime**页签，单击**自定义配置**，添加如下三个配置项。

参数	描述
druid.metadata.storage.connector.connectURI	设置为jdbc:mysql://rm-xxxxx.mysql.rds.aliyuncs.com:3306/druiddb。
druid.metadata.storage.connector.user	设置为druid。
druid.metadata.storage.connector.password	设置为druidpw。

iii. 依次单击右上角的**保存**、**部署客户端配置**和**重启All Components**，配置即可生效。

iv. 登录RDS管理控制台，查看druiddb创建表的情况。

- 配置组件内存。

  E-MapReduce Druid组件内存设置主要包括两方面：

  - 堆内存。
  - direct内存。

访问Druid web页面

E-MapReduce Druid自带三个Web页面：

- Overlord：*http://emr-header-1.cluster-1234:18090*，用于查看Task运行情况。
- Coordinator：*http://emr-header-1.cluster-1234:18081*，用于查看Segments存储情况，并设置Rule加载和丢弃Segments。
- Router（EMR-3.23.0及后续版本）：*http://emr-header-1.cluster-1234:18888*，也称之为Console，是新版Druid的统一入口。

访问E-MapReduce Druid的Web页面：

- 在集群管理页面，单击**访问链接与端口**，找到Druid Router UI链接，单击链接进入。

  > **说明** 您可以使用Knox账号访问Druid Web页面，Knox账号创建请参见管理用户，Knox使用请参见Knox使用说明。

- 通过建立SSH隧道。详情请参见通过SSH隧道方式访问开源组件Web UI。

批量索引

- 与Hadoop集群交互

  您在创建E-MapReduce Druid集群时如果勾选了YARN，则系统会自动为您配置好HDFS和YARN的交互，您无需额外操作。下面的介绍是E-MapReduce 配置独立Druid集群与独立Hadoop集群之间交互。例如，E-MapReduce Druid集群cluster id为1234，Hadoop集群cluster id为5678。

  > **说明** 请严格按照指导进行操作，如果操作不当，集群可能就不会按照预期工作。

  非安全独立Hadoop集群，请按照如下操作进行：

  i. 确保集群间能够通信（两个集群在一个安全组下，或两个集群在不同安全组，但两个安全组之间配置了访问规则）。

  ii. 在E-MapReduce Druid集群的每个节点的指定路径下，放置一份Hadoop集群中*/etc/ecm/hadoop-conf*路径下的*core-site.xml*、*hdfs-site.xml*、*yarn-site.xml*、*mapred-site.xml*文件。这些文件在E-MapReduce Druid集群节点上放置的路径与E-MapReduce集群的版本有关，详情说明如下：

  - EMR-3.23.0及后续版本：*/etc/ecm/druid-conf/druid/cluster/_common*
  - EMR-3.23.0之前版本：*/etc/ecm/druid-conf/druid/_common*

  > **说明** 如果创建集群时选了自带Hadoop，则在上述目录下会有几个软链接指向自带Hadoop的配置，请先移除这些软链接。

  iii. 将Hadoop集群的hosts写入到E-MapReduce Druid集群的hosts列表中，注意Hadoop集群的hostname应采用长名形式，如emr-header-1.cluster-xxxxxxxx，且最好将Hadoop的hosts放在本集群hosts之后，例如：

```
...
10.157.*.*    emr-as.cn-hangzhou.aliyuncs.com
10.157.*.*    eas.cn-hangzhou.emr.aliyuncs.com
192.168.*.*   emr-worker-1.cluster-1234 emr-worker-1 emr-header-2.cluster-1234 emr-header-2 iZbp1h9g7boqo9x23qb****
192.168.*.*   emr-worker-2.cluster-1234 emr-worker-2 emr-header-3.cluster-1234 emr-header-3 iZbp1eaa5819tkjx55y****
192.168.*.*   emr-header-1.cluster-1234 emr-header-1 iZbp1e3zwuvnmakmsje****
--以下为hadoop集群的hosts信息
192.168.*.*   emr-worker-1.cluster-5678 emr-header-2.cluster-5678 iZbp195rj7zvx8qar4f****
192.168.*.*   emr-worker-2.cluster-5678 emr-header-3.cluster-5678 iZbp15vy2rsxoegki4q****
192.168.*.*   emr-header-1.cluster-5678 iZbp10tx4egw3wfnh5o****
```

安全Hadoop集群，请按如下操作进行：

i. 确保集群间能够通信（两个集群在一个安全组下，或两个集群在不同安全组，但两个安全组之间配置了访问规则）。

ii. 在E-MapReduce Druid集群的每个节点的指定路径下，放置一份Hadoop集群*/etc/ecm/hadoop-conf*路径下的*core-site.xml*、*hdfs-site.xml*、*yarn-site.xml*、*mapred-site.xml*文件，并修改*core-site.xml*中hadoop.security.authentication.use.has为*false*。

其中，*core-site.xml*、*hdfs-site.xml*、*yarn-site.xml*、*mapred-site.xml*文件在E-MapReduce Druid集群节点上放置的路径与E-MapReduce集群的版本有关，详情说明如下：

- EMR-3.23.0 及以上版本：*/etc/ecm/druid-conf/druid/cluster/_common*
- EMR-3.23.0 以下版本：*/etc/ecm/druid-conf/druid/_common*

> **说明** 如果创建集群时选了自带Hadoop，则在上述目录下会有几个软链接指向自带Hadoop的配置，请先移除这些软链接。

其中，hadoop.security.authentication.use.has是客户端配置，目的是让用户能够使用AccessKey进行认证。如果使用Kerberos认证方式，则需要disable该配置。

iii. 将Hadoop集群的hosts写入到E-MapReduce Druid集群每个节点的hosts列表中。

Hadoop集群的hostname应采用长名形式，例如emr-header-1.cluster-xxxxxxxx，且最好将Hadoop的hosts放在本集群hosts之后。

iv. 设置两个集群间的Kerberos跨域互信，详情请参见跨域互信。

v. 在Hadoop集群的所有节点下都创建一个本地druid账户（`useradd -m -g hadoop druid`），或者设置druid.auth.authenticator.kerberos.authToLocal，创建Kerberos账户到本地账户的映射规则。

语法规则请参见Druid-Kerberos。

> **说明** 默认在安全Hadoop集群中，所有Hadoop命令必须运行在一个本地的账户中，该本地账户需要与Principal的name部分同名。YARN支持将一个Principal映射至本地一个账户。

vi. 重启Druid服务。

- 使用Hadoop对批量数据创建索引

E-MapReduce Druid自带了一个名为wikiticker的例子，在*${DRUID_HOME}/quickstart/tutorial*目录下（*${DRUID_HOME}*默认为*/usr/lib/druid-current*）。wikiticker文件（wikiticker-2015-09-12-sampled.json.gz）的每一行是一条记录，每条记录是个JSON对象。其格式如下所示。

```
```json
{
    "time": "2015-09-12T00:46:58.771Z",
    "channel": "#en.wikipedia",
    "cityName": null,
    "comment": "added project",
    "countryIsoCode": null,
    "countryName": null,
    "isAnonymous": false,
    "isMinor": false,
    "isNew": false,
    "isRobot": false,
    "isUnpatrolled": false,
    "metroCode": null,
    "namespace": "Talk",
    "page": "Talk:Oswald Tilghman",
    "regionIsoCode": null,
    "regionName": null,
    "user": "GELongstreet",
    "delta": 36,
    "added": 36,
    "deleted": 0
}
```
```

使用Hadoop对批量数据创建索引，请按照如下步骤进行操作：

i. 解压该压缩文件，并放置于HDFS的目录下（例如*hdfs://emr-header-1.cluster-5678:9000/druid*）。在Hadoop集群上执行如下命令。

```
### 如果是在独立Hadoop集群上操作，设置两个集群互信之后需要拷贝一个druid.keytab到Hadoop集群再kinit。
 kinit -kt /etc/ecm/druid-conf/druid.keytab druid
 ###
 hdfs dfs -mkdir hdfs://emr-header-1.cluster-5678:9000/druid
 hdfs dfs -put ${DRUID_HOME}/quickstart/tutorial/wikiticker-2015-09-12-sampled.json hdfs://emr-header-1.cluster-5678:
9000/druid
```

> **说明**
>
> - 对于安全集群执行HDFS命令前先修改*/etc/ecm/hadoop-conf/core-site.xml*中的hadoop.security.authentication.use.has为*false*。
> - 请确保已经在Hadoop集群每个节点上创建名为druid的Linux账户。

ii. 准备数据索引任务文件*${DRUID_HOME}/quickstart/tutorial/wikiticker-index.json*，示例如下。

```
{
     "type" : "index_hadoop",
     "spec" : {
         "ioConfig" : {
             "type" : "hadoop",
             "inputSpec" : {
                 "type" : "static",
                 "paths" : "hdfs://emr-header-1.cluster-5678:9000/druid/wikiticker-2015-09-12-sampled.json"
             }
         },
         "dataSchema" : {
             "dataSource" : "wikiticker",
             "granularitySpec" : {
                 "type" : "uniform",
                 "segmentGranularity" : "day",
                 "queryGranularity" : "none",
                 "intervals" : ["2015-09-12/2015-09-13"]
             },
             "parser" : {
                 "type" : "hadoopyString",
                 "parseSpec" : {
                     "format" : "json",
                     "dimensionsSpec" : {
                         "dimensions" : [
                             "channel",
                             "cityName",
                             "comment",
                             "countryIsoCode",
                             "countryName",
                             "isAnonymous",
                             "isMinor",
                             "isNew",
                             "isRobot",
                             "isUnpatrolled",
                             "metroCode",
                             "namespace",
                             "page",
                             "regionIsoCode",
                             "regionName",
                             "user"
                         ]
                     },
                     "timestampSpec" : {
                         "format" : "auto",
                         "column" : "time"
                     }
                 }
             },
             "metricsSpec" : [
                 {
                     "name" : "count",
                     "type" : "count"
                 },
```

```
                {
                    "name" : "added",
                    "type" : "longSum",
                    "fieldName" : "added"
                },
                {
                    "name" : "deleted",
                    "type" : "longSum",
                    "fieldName" : "deleted"
                },
                {
                    "name" : "delta",
                    "type" : "longSum",
                    "fieldName" : "delta"
                },
                {
                    "name" : "user_unique",
                    "type" : "hyperUnique",
                    "fieldName" : "user"
                }
            ]
        },
        "tuningConfig" : {
            "type" : "hadoop",
            "partitionsSpec" : {
                "type" : "hashed",
                "targetPartitionSize" : 5000000
            },
            "jobProperties" : {
                "mapreduce.job.classloader": "true"
            }
        }
    },
    "hadoopDependencyCoordinates": ["org.apache.hadoop:hadoop-client:2.8.5"]
}
```

参数	描述
spec.ioConfig.type	设置为hadoop。
spec.ioConfig.inputSpec.paths	文件路径。
tuningConfig.type	设置为hadoop。
tuningConfig.jobProperties	设置MapReduce Job的Classloader。
hadoopDependencyCoordinates	制定了Hadoop Client的版本。

iii. 在E-MapReduce Druid集群上运行批量索引命令。

```
cd ${DRUID_HOME}
 curl --negotiate -u:druid -b ~/cookies -c ~/cookies -XPOST -H 'Content-Type:application/json' -d @quickstart/tutoria
l/wikiticker-index.json http://emr-header-1.cluster-1234:18090/druid/indexer/v1/task
```

其中 `- -negotiate` 、`-u` 、`-b` 、`-c` 等选项是针对安全E-MapReduce Druid集群的。Overlord的端口默认为18090。

iv. 查看作业运行情况。

在浏览器访问*http://emr-header-1.cluster-1234:18090/console.html*，查看作业运行情况。

v. 根据Druid语法查询数据。

Druid有自己的查询语法。请准备一个JSON格式的查询文件。如下所示为wikiticker数据的top N查询文件（*${DRUID_HOME}/quickstart/tutorial/wikiticker-top-pages.json*）。

```
{
     "queryType" : "topN",
     "dataSource" : "wikiticker",
     "intervals" : ["2015-09-12/2015-09-13"],
     "granularity" : "all",
     "dimension" : "page",
     "metric" : "edits",
     "threshold" : 25,
     "aggregations" : [
         {
             "type" : "longSum",
             "name" : "edits",
             "fieldName" : "count"
         }
     ]
 }
```

在命令行界面运行下面的命令即可看到查询结果。

```
cd ${DRUID_HOME}
 curl --negotiate -u:druid -b ~/cookies -c ~/cookies -XPOST -H 'Content-Type:application/json' -d @quickstart/tutoria
l/wikiticker-top-pages.json 'http://emr-header-1.cluster-1234:18082/druid/v2/?pretty'
```

其中 `- -negotiate` 、`-u` 、`-b` 、`-c` 等选项是针对安全E-MapReduce Druid集群的。

实时索引

对于数据从Kafka集群实时到E-MapReduce Druid集群进行索引，推荐您使用Kafka Indexing Service扩展，提供高可靠保证和Exactly-Once语义。详情请参见Kafka Indexing Service。

如果您的数据实时到了阿里云日志服务（SLS），并想用E-MapReduce Druid实时索引这部分数据，您可以使用SLS Indexing Service扩展，提供高可靠保证和Exactly-Once语义。使用SLS Indexing Service避免了您额外建立并维护Kafka集群。详情请参见 SLS-Indexing-Service。

6.2.35.3. 数据格式描述文件

本文介绍索引数据的描述文件（Ingestion Spec文件）。

Ingestion Spec（数据格式描述）是Druid对要索引数据的格式以及如何索引该数据格式的一个统一描述，它是一个JSON文件，一般由三部分组成。

```
{
    "dataSchema" : {...},
    "ioConfig" : {...},
    "tuningConfig" : {...}
}
```

键	格式	描述	是否必须
dataSchema	JSON对象	待消费数据的schema信息。dataSchema是固定的，不随数据消费方式改变。	是
ioConfig	JSON对象	待消费数据的来源和消费去向。数据消费方式不同，ioConfig也不相同。	是
tuningConfig	JSON对象	调节数据消费时的参数。数据消费方式不同，可以调节的参数也不相同。	否

DataSchema

第一部分的dataSchema描述了数据的格式，如何解析该数据，典型结构如下。

```
{
    "dataSource": <name_of_dataSource>,
    "parser": {
        "type": <>,
        "parseSpec": {
            "format": <>,
            "timestampSpec": {},
            "dimensionsSpec": {}
        }
    },
    "metricsSpec": {},
    "granularitySpec": {}
}
```

键	格式	描述	是否必须
dataSource	字符串	数据源的名称。	是
parser	JSON对象	数据的解析方式。	是
metricsSpec	JSON对象数组	聚合器（aggregator）列表。	是
granularitySpec	JSON对象	数据聚合设置，如创建segments、聚合粒度等。	是

- parser

  parser部分决定了您的数据如何被正确地解析，metricsSpec定义了数据如何被聚集计算，granularitySpec定义了数据分片的粒度、查询的粒度。

  对于parser，type有两个选项：string和hadoopString，后者用于Hadoop索引的 job。parseSpec是数据格式解析的具体定义。

键	格式	描述	是否必须
type	字符串	数据格式，可以是 “json”、“jsonLowercase”、“csv”和“tsv”几种格式。	是
timestampSpec	JSON对象	时间戳和时间戳类型。	是
dimensionsSpec	JSON对象	数据的维度（包含哪些列）。	是

  对于不同的数据格式，可能还有额外的parseSpec选项。

  timestampSpec表的描述如下。

键	格式	描述	是否必须
column	字符串	时间戳对应的列。	是
format	字符串	时间戳类型，可选”iso”、“millis”、“posix”、“auto”和joda time支持的类型。	是

  dimensionsSpec表的描述如下。

键	格式	描述	是否必须
dimensions	JSON数组	数据包含的维度。每个维度可以只是个字符串，或者可以额外指明维度的属性。例如 “dimensions”:[“dimenssion1”,“dimenssion2”,“{“type”:“long”,“name”:“dimenssion3”}]，默认是STRING类型。	是
dimensionExclusions	JSON字符串数组	数据消费时要剔除的维度。	否
spatialDimensions	JSON对象数组	空间维度。	否

- metricsSpec

  metricsSpec是一个JSON对象数组，定义了一些聚合器（aggregators）。聚合器通常有如下的结构。

```
{
    "type": <type>,
    "name": <output_name>,
    "fieldName": <metric_name>
}
```

官方提供了以下常用的聚合器。

类型	type 可选
count	count
sum	longSum、doubleSum、floatSum
min/max	longMin/longMax、doubleMin/doubleMax、floatMin/floatMax
first/last	longFirst/longLast、doubleFirst/doubleLast、floatFirst/floatLast
javascript	javascript
cardinality	cardinality
hyperUnique	hyperUnique

> **说明** 后三个属于高级聚合器，详情请参见Apache Druid官方文档。

- granularitySpec

  聚合支持两种聚合方式：uniform和arbitrary，前者以一个固定的时间间隔聚合数据，后者尽量保证每个segments大小一致，时间间隔是不固定的。目前uniform是默认选项。

键	格式	描述	是否必须
segmentGranularity	字符串	segments粒度。uniform 方式使用。默认为DAY。	否
queryGranularity	字符串	可供查询的最小数据聚合粒度，默认值为true。	否
rollup	bool值	是否聚合。	否
intervals	字符串	数据消费时间间隔。	◦ batch：是 ◦ realtime：否

ioConfig

第二部分ioConfig描述了数据来源。以下是一个Hadoop索引的例子。

```
{
    "type": "hadoop",
    "inputSpec": {
        "type": "static",
        "paths": "hdfs://emr-header-1.cluster-6789:9000/druid/quickstart/wikiticker-2015-09-16-sampled.json"
    }
}
```

> **说明**
>
> 对于通过Tranquility处理的流式数据，这部分是不需要的。

Tunning Config

Tuning Config是指一些额外的设置。

Tunning Config的内容依赖于您的数据来源。例如，Hadoop对批量数据创建索引，您可以指定MapReduce参数。

6.2.35.4. Kafka Indexing Service

本文介绍如何在E-MapReduce中使用Apache Druid Kafka Indexing Service实时消费Kafka数据。

前提条件

已创建E-MapReduce的Druid集群和Kafka集群，详情请参见创建集群。

背景信息

Kafka Indexing Service是Apache Druid推出的使用Apache Druid的Indexing Service服务实时消费Kafka数据的插件。该插件会在Overlord中启动一个Supervisor，Supervisor启动后会在Middlemanager中启动indexing task，这些task会连接到Kafka集群消费topic数据，并完成索引创建。您只需要准备一个数据消费格式文件，通过REST API手动启动Supervisor。

配置Druid集群与Kafka集群交互

E-MapReduce Druid集群与Kafka集群交互的配置方式与Hadoop集群类似，均需要设置连通性和Hosts。

- 对于非安全Kafka集群，请按照以下步骤操作：
  i. 确保集群间能够通信（两个集群在一个安全组下，或两个集群在不同安全组，但两个安全组之间配置了访问规则）。
  ii. 将Kafka集群的Hosts写入到E-MapReduce Druid集群每一个节点的Hosts列表中。

  > 注意 Kafka集群的hostname应采用长名形式，例如emr-header-1.cluster-xxxxxxxx。

- 对于安全Kafka集群，您需要执行下列操作（前两步与非安全Kafka集群相同）：
  i. 确保集群间能够通信（两个集群在一个安全组下，或两个集群在不同安全组，但两个安全组之间配置了访问规则）。
  ii. 将Kafka集群的hosts写入到E-MapReduce Druid集群每一个节点的hosts列表中。

  > 注意 Kafka集群的hostname应采用长名形式，例如emr-header-1.cluster-xxxxxxxx。

  iii. 设置两个集群间的Kerberos跨域互信（详情请参见跨域互信），推荐做双向互信。
  iv. 准备一个客户端安全配置文件，文件内容格式如下。

  ```
  KafkaClient {
        com.sun.security.auth.module.Krb5LoginModule required
        useKeyTab=true
        storeKey=true
        keyTab="/etc/ecm/druid-conf/druid.keytab"
        principal="druid@EMR.1234.COM";
    };
  ```

  文件准备好后，将该配置文件同步到E-MapReduce Druid集群的所有节点上，放置于某一个目录下面（例如*/tmp/kafka/kafka_client_jaas.conf*）。
  v. 在E-MapReduce Druid配置页面的*overlord.jvm*中新增如下选项。

  ```
  -Djava.security.auth.login.config=/tmp/kafka/kafka_client_jaas.conf
  ```

  vi. 在E-MapReduce Druid配置页面的*middleManager.runtime*中配置 `druid.indexer.runner.javaOpts=-Djava.security.auth.login.confi=/tmp/kafka/kafka_client_jaas.conf` 和其他JVM启动参数。
  vii. 重启Druid服务。

使用Kafka Indexing Service实时消费Kafka数据

1. 在Kafka集群（或Gateway）上执行以下命令创建一个名称为*metrics*的topic。

   ```
   -- 如果开启了Kafka高安全。
   export KAFKA_OPTS="-Djava.security.auth.login.config=/etc/ecm/kafka-conf/kafka_client_jaas.conf"
   kafka-topics.sh --create --zookeeper emr-header-1:2181,emr-header-2:2181,emr-header-3:2181 --partitions 1 --replication-factor 1 --topic metrics
   ```

   实际创建topic时，您需要根据您的环境配置来替换上述命令中的各个参数。其中，--zookeeper参数中路径的获取方式是：登录阿里云 E-MapReduce 控制台> 进入Kafka集群的Kafka服务的配置页面，查看zookeeper.connect配置项的值。如果您的Kafka集群是自建集群，则您需要根据集群的实际配置来替换--zookeeper参数。
2. 定义数据源的数据格式描述文件（名称命名为*metrics-kafka.json*），并放置在当前目录下（或放置在其他您指定的目录上）。

```
{
     "type": "kafka",
     "dataSchema": {
         "dataSource": "metrics-kafka",
         "parser": {
             "type": "string",
             "parseSpec": {
                 "timestampSpec": {
                     "column": "time",
                     "format": "auto"
                 },
                 "dimensionsSpec": {
                     "dimensions": ["url", "user"]
                 },
                 "format": "json"
             }
         },
         "granularitySpec": {
             "type": "uniform",
             "segmentGranularity": "hour",
             "queryGranularity": "none"
         },
         "metricsSpec": [{
                 "type": "count",
                 "name": "views"
             },
             {
                 "name": "latencyMs",
                 "type": "doubleSum",
                 "fieldName": "latencyMs"
             }
         ]
     },
     "ioConfig": {
         "topic": "metrics",
         "consumerProperties": {
             "bootstrap.servers": "emr-worker-1.cluster-xxxxxxxx:9092(您 Kafka 集群的 bootstrap.servers)",
             "group.id": "kafka-indexing-service",
             "security.protocol": "SASL_PLAINTEXT",
             "sasl.mechanism": "GSSAPI"
         },
         "taskCount": 1,
         "replicas": 1,
         "taskDuration": "PT1H"
     },
     "tuningConfig": {
         "type": "kafka",
         "maxRowsInMemory": "100000"
     }
 }
```

说明 ioConfig.consumerProperties.security.protocol和ioConfig.consumerProperties.sasl.mechanism为安全相关选项（非安全Kafka集群不需要）。

3. 执行如下命令添加Kafka Supervisor。

```
curl --negotiate -u:druid -b ~/cookies -c ~/cookies -XPOST -H 'Content-Type: application/json' -d @metrics-kafka.json h
ttp://emr-header-1.cluster-1234:18090/druid/indexer/v1/supervisor
```

其中 `--negotiate` 、 `-u` 、 `-b` 和 `-c` 是针对安全E-MapReduce Druid集群的选项。

4. 在Kafka集群上开启一个Console Producer。

```
# 如果开启了Kafka高安全：
export KAFKA_OPTS="-Djava.security.auth.login.config=/etc/ecm/kafka-conf/kafka_client_jaas.conf"
echo -e "security.protocol=SASL_PLAINTEXT\nsasl.mechanism=GSSAPI" > /tmp/kafka-producer.conf
kafka-console-producer.sh --producer.config /tmp/kafka-producer.conf --broker-list emr-header-1:9092,emr-header-2:9092,
emr-header-3:9092 --topic metrics
```

其中，*--producer.config /tmp/kafka-producer.conf*是针对安全Kafka集群的选项。

5. 在*Kafka-console-producer.sh*的命令提示符下输入数据。

```
{"time": "2018-03-06T09:57:58Z", "url": "/foo/bar", "user": "alice", "latencyMs": 32}
{"time": "2018-03-06T09:57:59Z", "url": "/", "user": "bob", "latencyMs": 11}
{"time": "2018-03-06T09:58:00Z", "url": "/foo/bar", "user": "bob", "latencyMs": 45}
```

时间戳可用如下Python命令生成。

```
python -c 'import datetime; print(datetime.datetime.utcnow().strftime("%Y-%m-%dT%H:%M:%SZ"))'
```

6. 准备名为*metrics-search.json*的查询文件。

```
{
     "queryType" : "search",
     "dataSource" : "metrics-kafka",
     "intervals" : ["2018-03-02T00:00:00.000/2018-03-08T00:00:00.000"],
     "granularity" : "all",
     "searchDimensions": [
         "url",
         "user"
     ],
     "query": {
         "type": "insensitive_contains",
         "value": "bob"
     }
 }
```

7. 在E-MapReduce Druid集群的Master节点上执行如下命令。

```
curl --negotiate -u:Druid -b ~/cookies -c ~/cookies -XPOST -H 'Content-Type: application/json' -d @metrics-search.json
http://emr-header-1.cluster-1234:18082/druid/v2/?pretty
```

其中 `--negotiate` 、 `-u` 、 `-b` 和 `-c` 是针对安全 E-MapReduce Druid集群的选项。

返回结果示例如下。

```
[ {
   "timestamp" : "2018-03-06T09:00:00.000Z",
   "result" : [ {
     "dimension" : "user",
     "value" : "bob",
     "count" : 2
   } ]
 } ]
```

6.2.35.5. SLS Indexing Service

SLS Indexing Service是E-MapReduce推出的一个Druid插件，用于从日志服务（Log Service，简称SLS）消费数据。

背景介绍

SLS Indexing Service优点如下：

- 极为便捷的数据采集，可以利用SLS的多种数据采集方式实时将数据导入SLS。
- 无需额外维护一个Kafka集群，省去了数据流的一个环节。
- 支持Exactly-Once语义。

  因为SLS Indexing Service消费原理与Kafka Indexing Service类似，所以也支持Kafka Indexing Service一样的Exactly-Once语义。
- 消费作业高可靠保证，例如，作业失败重试，集群重启等。

准备工作

- 如果您还没有开通SLS服务，请先开通SLS服务，并配置好相应的Project和Logstore。
- 准备好以下配置项内容：
  - SLS服务的Endpoint（请使用内网服务入口）。
  - 阿里云账号的AccessKey ID和对应的AccessKey Secret。

使用SLS Indexing Service

1. 准备数据格式描述文件

   如果您熟悉 Kafka Indexing Service，那么 SLS Indexing Service 会非常简单。具体请参见Kafka Indexing Service的介绍，我们用同样的数据进行索引，那么数据源的数据格式描述文件如下（将其保存为 metrics-sls.json）：

```
{
    "type": "sls",
    "dataSchema": {
        "dataSource": "metrics-sls",
        "parser": {
            "type": "string",
            "parseSpec": {
                "timestampSpec": {
                    "column": "time",
                    "format": "auto"
                },
                "dimensionsSpec": {
                    "dimensions": ["url", "user"]
                },
                "format": "json"
            }
        },
        "granularitySpec": {
            "type": "uniform",
            "segmentGranularity": "hour",
            "queryGranularity": "none"
        },
        "metricsSpec": [{
                "type": "count",
                "name": "views"
            },
            {
                "name": "latencyMs",
                "type": "doubleSum",
                "fieldName": "latencyMs"
            }
        ]
    },
    "ioConfig": {
        "project": <your_project>,
        "logstore": <your_logstore>,
        "endpoint": "cn-hangzhou-intranet.log.aliyuncs.com", （以杭州为例，注意使用内网服务入口）
        "accessKeyId": <your_access_key_id>,
        "accessKeySec": <your_access_key_secret>,
        "collectMode": "simple"/"other"
        "taskCount": 1,
        "replicas": 1,
        "taskDuration": "PT1H"
    },
    "tuningConfig": {
        "type": "sls",
        "maxRowsInMemory": "100000"
    }
}
```

对比Kafka Indexing Service一节中的介绍，我们发现两者基本上是一样的。这里简要列一下需要注意的字段：

- type: sls。
- dataSchema.parser.parseSpec.format：与ioConfig.consumerProperties.logtail.collection-mode有关，也就是与SLS日志的收集模式有关。如果是极简模式（simple）收集，那么该处原本文件是什么格式，就填什么格式。如果是非极简模式（other）收集，那么此处取值为json。
- ioConfig.project：您要收集的日志的project。
- ioConfig.logstore：您要收集的日志的logstore。
- ioConfig.consumerProperties.endpoint：SLS内网服务地址，例如杭州对应 *cn-hangzhou-intranet.log.aliyuncs.com*。
- ioConfig.consumerProperties.access-key-id：阿里云账号的AccessKey ID。
- ioConfig.consumerProperties.access-key-secret：阿里云账号的AccessKeySecret。
- ioConfig.consumerProperties.logtail.collection-mode：SLS日志收集模式，极简模式填simple，其他情况填 other。

> 注意 上述配置文件中的*ioConfig*配置格式仅适用于EMR-3.20.0及之前版本。自EMR-3.21.0开始，*ioConfig*配置变更如下：
>
> ```
> "ioConfig": {
>         "project": <your_project>,
>         "logstore": <your_logstore>,
>         "endpoint": "cn-hangzhou-intranet.log.aliyuncs.com", （以杭州为例，注意使用内网服务入口）
>         "accessKeyId": <your_access_key_id>,
>         "accessKeySec": <your_access_key_secret>,
>         "collectMode": "simple"/"other"
>         "taskCount": 1,
>         "replicas": 1,
>         "taskDuration": "PT1H"
>     },
> ```
>
> 即，取消了 consumerProperties 层级、access-key-id、access-key-secret，logtail.collection-mode 变更为 accessKeyIdaccessKeySeccollectMode 。

2. 执行下述命令添加SLS supervisor。

```
curl --negotiate -u:druid -b ~/cookies -c ~/cookies -XPOST -H 'Content-Type: application/json' -d @metrics-sls.json http://emr-header-1.cluster-1234:18090/druid/indexer/v1/supervisor
```

> 注意 其中--negotiate、-u、-b、-c等选项是针对安全Druid集群。

3. 向SLS中导入数据。

   您可以采用多种方式向SLS中导入数据。具体请参见 SLS 文档。

4. 在Druid端进行相关查询。

6.2.35.6. 常见问题

本文介绍E-MapReduce Druid使用过程中遇到的一些常见问题以及解决方法。

索引失败问题分析思路

当发现索引失败时，一般遵循如下排错思路：

- 对于批量索引
  i. 如果curl直接返回错误，或者不返回，检查一下输入文件格式。或者curl加上-v参数，观察REST API的返回情况。
  ii. 在Overlord页面观察作业执行情况，如果失败，查看页面上的logs。
  iii. 在很多情况下并没有生成logs。如果是Hadoop作业，打开YARN页面查看是否有索引作业生成，并查看作业执行log。
  iv. 如果上述情况都没有定位到错误，需要登录到E-MapReduce Druid集群，查看overlord的执行日志（位于*/mnt/disk1/log/druid/overlord—emr-header-1.cluster-xxxx.log*），如果是HA集群，查看您提交作业的那个Overlord。
  v. 如果作业已经被提交到Middlemanager，但是从Middlemanager返回了失败，则需要从Overlord中查看作业提交到了哪个worker，并登录到相应的worker，查看Middlemanager的日志（位于*/mnt/disk1/log/druid/middleManager-emr-header-1.cluster-xxxx.log*）。
- 对于Kafka Indexing Service和SLS Indexing Service
  i. 首先查看Overlord的Web页面：*http://emr-header-1:18090*，查看Supervisor的运行状态，检查payload是否合理。
  ii. 查看失败task的log。
  iii. 如果不能从task log定位出失败原因，则需要从Overlord log排查问题。
- 对于Tranquility实时索引

  查看Tranquility log，查看消息是否被接收到了或者是否被丢弃（drop）掉了。

  其余的排查步骤同批量索引的2～5。

  错误多数情况为集群配置问题和作业问题。集群配置问题包括：内存参数是否合理、跨集群联通性是否正确、安全集群访问是否通过、principal是否正确等等，作业问题包括作业描述文件格式是否正确、输入数据是否能够正常被解析，以及一些其他的作业相关的配置（例如ioConfig）。

问题汇总

- 组件启动失败

  此类问题多数是由于组件JVM运行参数配置问题，例如机器可能没有很大的内存，而配置了较大的JVM内存或者较多的线程数量。

  解决方法：查看组件日志并调整相关参数即可解决。JVM内存涉及堆内存和直接内存。具体可参见Basic cluster tuning。

- 索引时YARN task执行失败，显示诸如 `Error: class com.fasterxml.jackson.datatype.guava.deser.HostAndPortDeserializer overrides final method deserialize.(Lcom/fasterxml/jackson/core/JsonParser;Lcom/fasterxml/jackson/databind/DeserializationContext;)Ljava/lang/Object;` 之类的 jar 包冲突错误。

  解决方法：在indexing的作业配置文件中加入如下配置。

```
"tuningConfig" : {
     ...
     "jobProperties" : {
         "mapreduce.job.classloader": "true"
         或者
         "mapreduce.job.user.classpath.first": "true"
     }
     ...
 }
```

  其中参数mapreduce.job.classloader 让MR job用独立的classloader，mapreduce.job.user.classpath.first是让MapReduce优先使用用户的JAR包，两个配置项配置一个即可。 请参见 Working with different versions of Apache Hadoop。

- indexing作业的日志中报reduce无法创建segments目录。

  解决方法：

  - 注意检查deep storage的设置，包括type和directory。当type为local时，注意directory的权限设置。当type为HDFS时，directory尽量用完整的HDFS路径写法，例如*hdfs://hdfs_master:9000/path*。hdfs_master最好用IP，如果用域名，要用完整的域名，例如emr-header-1.cluster-xxxxxxxx，而不是emr-header-1。
  - 用Hadoop批量索引时，要将segments的deep storage设置为hdfs，local的方式会导致MR作业处于UNDEFINED状态，这是因为远程的YARN集群无法在reduce task下创建local的segments目录。（此针对独立E-MapReduce Druid集群）。

- 错误提示 `Failed to create directory within 10000 attempts…`

  此问题一般为JVM配置文件中java.io.tmp设置的路径不存在的问题。设置该路径并确保E-MapReduce Druid账户有权限访问即可。

- 错误提示 `com.twitter.finagle.NoBrokersAvailableException: No hosts are available for disco!firehose:druid:overlord`

  此问题一般是ZooKeeper的连接问题。确保E-MapReduce Druid与Tranquility对于ZooKeeper有相同的连接字符串。E-MapReduce Druid默认的ZooKeeper路径为*/druid*，因此确保Tranquility设置中zookeeper.connect包含路径*/druid*。

  > **说明** Tranquility Kafka设置中有两个ZooKeeper的设置，一个为zookeeper.connect，连接E-MapReduce Druid集群的ZooKeeper，一个为kafka.zookeeper.connect，连接Kafka集群的ZooKeeper。

- 索引时MiddleManager提示找不到类 `com.hadoop.compression.lzo.LzoCodec` 。

  这是因为EMR的Hadoop集群配置了lzo压缩。

  解决方法：拷贝*EMR $HADOOP_HOME/lib*下的JAR包和*native*文件夹到E-MapReduce Druid的druid.extensions.hadoopDependenciesDir（默认为*$DRUID_HOME/hadoop-dependencies*）。

- 索引时提示如下错误：

```
2018-02-01T09:00:32,647 ERROR [task-runner-0-priority-0] com.hadoop.compression.lzo.GPLNativeCodeLoader - could not unpack the binaries
  java.io.IOException: No such file or directory
          at java.io.UnixFileSystem.createFileExclusively(Native Method) ~[?:1.8.0_151]
          at java.io.File.createTempFile(File.java:2024) ~[?:1.8.0_151]
          at java.io.File.createTempFile(File.java:2070) ~[?:1.8.0_151]
          at com.hadoop.compression.lzo.GPLNativeCodeLoader.unpackBinaries(GPLNativeCodeLoader.java:115) [hadoop-lzo-0.4.21-SNAPSHOT.jar:?]
```

  这个问题是因为*java.io.tmp*路径不存在的问题。设置该路径并确保E-MapReduce Druid账户有权限访问。

6.2.36. Kubeflow

6.2.36.1. 基于Kubeflow的Training示例

KubeFlow提供TFJob和PyTorchJob等CRD（CustomResourceDefinition），基于这些CRD，您可以在Kubernetes集群上运行分布式训练，无需过多关注分布式代码逻辑，也无需过多考虑集群的运维工作，可以将全部精力集中到模型开发当中，DataScience集群为您提供稳定的算力输出，以及Tensorflow和PyTorch等丰富的机器学习框架。

前提条件

- 已创建DataScience集群，并且选择了Kubeflow服务，详情请参见创建集群。
- 下载dsdemo代码：请已创建DataScience集群的用户，使用钉钉搜索钉钉群号32497587加入钉钉群以获取dsdemo代码。

背景信息

通过以下三种方式为您展示Training示例：

- estimator-API
- keras-API
- Pytorch训练

准备工作

1. 通过SSH方式连接集群，详情请参见登录集群。
2. 上传并解压缩下载的代码包至集群Master节点的*/dsdemo*目录下。
   本文示例是上传并解压缩在*root/dsdemo*目录下，您可以指定目录。

estimator-API

1. 执行以下命令，进入*estimator-API*目录。

   ```
   cd /root/dsdemo/kubeflow_samples/training/tf/estimator-API
   ```

2. 制作Train镜像。
   - 执行以下命令，制作Train镜像。

     ```
     make
     ```

   - 执行以下命令，制作镜像并将镜像push到镜像仓库。

     ```
     make push
     ```

3. 根据您的实际需求，修改*distributed_tfjob.yaml*文件内容。
   您需要修改代码中 `image` 的值，*distributed_tfjob.yaml*文件内容如下。

   ```
   apiVersion: "kubeflow.org/v1"
   kind: "TFJob"
   metadata:
     name: "distributed-training"
   spec:
     cleanPodPolicy: None
     tfReplicaSpecs:
       Worker:
         replicas: 3
         restartPolicy: Never
         template:
           metadata:
                 annotations:
                   scheduling.k8s.io/group-name: "distributed-training"
           spec:
             containers:
               - name: tensorflow
                 image: datascience-registry.cn-beijing.cr.aliyuncs.com/kubeflow-examples/distributed_worker:0.1.4
                 volumeMounts:
                   - mountPath: /train
                     name: training
             volumes:
               - name: training
                 persistentVolumeClaim:
                   claimName: strategy-estimator-volume
   ```

4. 执行以下命令，使修改的配置生效。

   ```
   kubectl apply -f distributed_tfjob.yaml
   ```

5. 根据您的实际需求，修改*pvc.yaml*配置文件。
   *pvc.yaml*文件内容如下。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: strategy-estimator-volume
  labels:
    app: strategy-estimator-volume
spec:
  storageClassName: "nfs-client"
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
```

> **说明** 您可以修改 `storage` 的值，调大存储空间，例如50 Gi。

6. 执行以下命令，使修改的配置生效。

```
kubectl apply -f pvc.yaml
```

7. 查看训练状态。
   i. 执行以下命令，获取Pod信息。

   ```
   kubectl get pods
   ```

   返回如下类似信息。

   ```
   NAME                                  READY   STATUS      RESTARTS   AGE
   distributed-training-worker-0         0/1     Completed   0          22h
   distributed-training-worker-1         0/1     Completed   0          22h
   distributed-training-worker-2         0/1     Completed   0          22h
   nfs-client-provisioner-5cb8b7cf76-k2z4d   1/1     Running     0          25h
   ```

   > **说明** 当Pod的状态为Completed时，表示训练结束。

   ii. 执行以下命令，查看Log信息。

   ```
   kubectl logs distributed-training-worker-0
   ```

8. 查看Checkpoint模型导出文件。
   i. 执行以下命令，进入*default_storage_class*目录。

   ```
   cd /mnt/disk1/k8s_pv/default_storage_class
   ```

   ii. 执行 `ll` 命令，获取模型目录。

   ```
   total 28
   drwxrwxrwx 2 root    root     4096 Jul  8 16:56 anonymous-workspace-ds-notebook2-pvc-09789bbd-aa40-4381-af63-ebcab57c584f
   drwxrwxrwx 7 root    root     4096 Jul  8 18:09 anonymous-workspace-ds-notebook-pvc-ae4db112-9491-4449-854a-7b0e67264703
   drwxrwxrwx 5 root    root     4096 Jul  8 16:27 default-strategy-estimator-volume-pvc-6cd2979e-f925-45b7-8c25-ce67e12e2f03
   drwxrwxrwx 3 root    root     4096 Jul  8 13:47 kubeflow-katib-mysql-pvc-9a3ebbfe-2952-4eaa-8a83-36e42d620cac
   drwxrwxrwx 3 root    root     4096 Jul  8 13:47 kubeflow-metadata-mysql-pvc-5778e165-8a08-4536-8e36-9185144d3d6d
   drwxrwxrwx 3 root    root     4096 Jul  8 13:47 kubeflow-minio-pvc-pvc-10015211-fc40-460e-b723-a5c220112d08
   drwxrwxrwx 6 polkitd ssh_keys 4096 Jul  8 13:54 kubeflow-mysql-pv-claim-pvc-5136b864-c868-4951-9b34-f6c5f06c2d6f
   ```

   iii. 执行以下命令，进入模型目录。

   ```
   cd default-strategy-estimator-volume-pvc-6cd2979e-f925-45b7-8c25-ce67e12e2f03
   ```

   > **说明** `default-strategy-estimator-volume-pvc-6cd2979e-f925-45b7-8c25-ce67e12e2f03` 需要替换为您实际生成的目录。

   iv. 执行以下命令，进入*master*目录。

   ```
   cd master
   ```

v. 执行下 `ll` 命令，查看目录信息。

```
total 4
drwxr-xr-x 3 root root 4096 Jul  8 16:27 1625732821
```

vi. 执行以下命令，进入*1625732821*目录。

```
cd 1625732821
```

> **说明** `1625732821` 需要替换为您实际生成的目录。

vii. 执行下 `ll` 命令，查看目录信息。

```
total 1788
-rw-r--r-- 1 root root    130 Jul  8 16:27 checkpoint
-rw-r--r-- 1 root root 838550 Jul  8 16:27 events.out.tfevents.1625732823.distributed-training-worker-0
-rw-r--r-- 1 root root 523304 Jul  8 16:27 graph.pbtxt
drwxr-xr-x 2 root root   4096 Jul  8 16:27 keras
-rw-r--r-- 1 root root    780 Jul  8 16:27 model.ckpt-0.data-00000-of-00001
-rw-r--r-- 1 root root    249 Jul  8 16:27 model.ckpt-0.index
-rw-r--r-- 1 root root 218354 Jul  8 16:27 model.ckpt-0.meta
-rw-r--r-- 1 root root    780 Jul  8 16:27 model.ckpt-3200.data-00000-of-00001
-rw-r--r-- 1 root root    249 Jul  8 16:27 model.ckpt-3200.index
-rw-r--r-- 1 root root 218354 Jul  8 16:27 model.ckpt-3200.meta
```

keras-API

> **注意** 此示例需要GPU环境，因此创建DataScience集群时需要选择GPU的配置，并且制作镜像时也需要GPU。

1. 执行以下命令，进入*keras-API*目录。

```
cd /root/dsdemo/kubeflow_samples/training/tf/keras-API
```

2. 制作Train镜像。
   - 执行以下命令，制作Train镜像。

```
make
```

   - 执行以下命令，制作镜像并将镜像push到镜像仓库。

```
make push
```

3. 根据您的实际需求，修改目录下训练配置文件的镜像名称。

   *multi_worker_tfjob.yaml*文件内容如下。

```
apiVersion: kubeflow.org/v1
kind: TFJob
metadata:
  name: multi-worker
spec:
  cleanPodPolicy: None
  tfReplicaSpecs:
    Worker:
      replicas: 3
      restartPolicy: Never
      template:
        spec:
          containers:
            - name: tensorflow
              image: datascience-registry.cn-beijing.cr.aliyuncs.com/kubeflow-examples/multi_worker_strategy:0.1.1
              volumeMounts:
                - mountPath: /train
                  name: training
              resources:
                limits:
                  nvidia.com/gpu: 1
          volumes:
            - name: training
              persistentVolumeClaim:
                claimName: strategy-volume
```

4. 执行以下命令，使修改的配置生效。

```
kubectl apply -f multi_worker_tfjob.yaml
```

5. 修改pvc配置文件。

*pvc.yam*l文件内容如下。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: strategy-volume
  labels:
    app: strategy-volume
spec:
  storageClassName: "nfs-client"
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
```

> **说明** 您可以修改 `storage` 的值，调大存储空间，例如50 Gi。

6. 执行以下命令，使修改的配置生效。

```
kubectl apply -f pvc.yaml
```

7. 查看训练状态。

i. 执行以下命令，获取Pod信息。

```
kubectl get pods
```

返回如下类似信息。

```
NAME                                     READY   STATUS      RESTARTS   AGE
multi-worker-worker-0                    0/1     Pending     0          5m33s
multi-worker-worker-1                    0/1     Pending     0          5m33s
multi-worker-worker-2                    0/1     Pending     0          5m33s
nfs-client-provisioner-5cb8b7cf76-k2z4d  1/1     Running     0          25h
```

> **说明** 当Pod的状态为Completed时，表示训练结束。

ii. 执行以下命令，查看Log信息。

```
kubectl logs distributed-training-worker-0
```

8. 进入如下目录，查看Checkpoint模型导出文件。

```
ll /mnt/disk1/k8s_pv/default_storage_class/default-strategy-estimator-volume-pvc-aa16c081-cd94-4565-bfde-daee90ee25a4/master/1625553118/
```

> **说明** `default-strategy-estimator-volume-pvc-aa16c081-cd94-4565-bfde-daee90ee25a4` 和 `1625553118` 需要替换为您实际的目录。

```
total 1788
-rw-r--r-- 1 root root    130 Jul  6 14:32 checkpoint
-rw-r--r-- 1 root root 838550 Jul  6 14:32 events.out.tfevents.1625553120.distributed-training-worker-0
-rw-r--r-- 1 root root 523405 Jul  6 14:32 graph.pbtxt
drwxr-xr-x 2 root root   4096 Jul  6 14:31 keras
-rw-r--r-- 1 root root    780 Jul  6 14:32 model.ckpt-0.data-00000-of-00001
-rw-r--r-- 1 root root    249 Jul  6 14:32 model.ckpt-0.index
-rw-r--r-- 1 root root 218354 Jul  6 14:32 model.ckpt-0.meta
-rw-r--r-- 1 root root    780 Jul  6 14:32 model.ckpt-3200.data-00000-of-00001
-rw-r--r-- 1 root root    249 Jul  6 14:32 model.ckpt-3200.index
-rw-r--r-- 1 root root 218354 Jul  6 14:32 model.ckpt-3200.meta
```

Pytorch训练

1. 执行以下命令，进入*mnist*目录。

```
cd /root/dsdemo/kubeflow_samples/training/pytorch/mnist
```

2. 制作Train镜像。
   - 执行以下命令，制作Train镜像。

     ```
     make
     ```

   - 执行以下命令，制作镜像并将镜像push到镜像仓库。

     ```
     make push
     ```

3. 根据您的实际需求，修改训练配置文件的镜像名称。

```
apiVersion: "kubeflow.org/v1"
kind: "PyTorchJob"
metadata:
  name: "pytorch-dist-mnist-gloo"
spec:
  pytorchReplicaSpecs:
    Master:
      replicas: 1
      restartPolicy: OnFailure
      template:
        metadata:
          annotations:
            sidecar.istio.io/inject: "false"
        spec:
          containers:
            - name: pytorch
              image: datascience-registry.cn-beijing.cr.aliyuncs.com/kubeflow-examples/pytorch-dist-mnist-test:0.1.5
              args: ["--backend", "gloo"]
              volumeMounts:
                - mountPath: /train
                  name: training
              # Comment out the below resources to use the CPU.
              resources:
                limits:
                  nvidia.com/gpu: 1
          volumes:
            - name: training
              persistentVolumeClaim:
                claimName: strategy-pytorch-volume
    Worker:
      replicas: 1
      restartPolicy: OnFailure
      template:
        metadata:
          annotations:
            sidecar.istio.io/inject: "false"
        spec:
          containers:
            - name: pytorch
              image: datascience-registry.cn-beijing.cr.aliyuncs.com/kubeflow-examples/pytorch-dist-mnist-test:0.1.5
              args: ["--backend", "gloo"]
              volumeMounts:
                - mountPath: /train
                  name: training
              # Comment out the below resources to use the CPU.
              resources:
                limits:
                  nvidia.com/gpu: 1
          volumes:
            - name: training
              persistentVolumeClaim:
                claimName: strategy-pytorch-volume
```

4. 修改pvc配置文件。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: strategy-pytorch-volume
  labels:
    app: strategy-pytorch-volume
spec:
  storageClassName: "nfs-client"
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
```

> **说明** 您可以修改 `storage` 的值，调大存储空间，例如50 Gi。

5. 申请PVC并提交训练。

```
kubectl apply -f ./v1/pytorch_job_mnist_gloo.yaml
```

6. 查看训练状态。
   i. 执行以下命令，获取Pod信息。

   ```
   kubectl get pods
   ```

   返回如下类似信息。

   ```
   NAME                                READY   STATUS      RESTARTS   AGE
   pytorch-dist-mnist-gloo-worker-0    0/1     Pending     0          11m
   ```

   > **说明** 当Pod的状态为Completed时，表示训练结束。

   ii. 执行以下命令，查看Log信息。

   ```
   kubectl logs pytorch-dist-mnist-gloo-master-0
   ```

7. 进入如下目录，查看Checkpoint模型导出文件。

```
cd /mnt/disk1/k8s_pv/default_storage_class/default-strategy-pytorch-volume-pvc-633b12ab-5f63-4e63-97b9-24c2b3de733c/
```

> **说明** `default-strategy-pytorch-volume-pvc-633b12ab-5f63-4e63-97b9-24c2b3de733c` 需要替换为您实际的目录。

返回信息类似如下所示。

```
total 1688
-rw-r--r-- 1 root root 1725813 Jul  6 15:35 mnist_cnn.pt
```

问题反馈

如果您在使用DataScience集群过程中有任何疑问或问题，请提交工单或联系我们的技术人员协助处理，同时也欢迎您使用钉钉搜索钉钉群号32497587加入钉钉群进行反馈或交流。

6.2.36.2. 基于Kubeflow或Seldon的在线服务

DataScience集群的KubeFlow服务内置了SeldonCore组件， 可以为模型提供在线服务，基于Kubernetes，您无需关心在线服务的运维工作。您可以根据提供的dsdemo代码，将Tensorflow，Pytorch和Python等模型部署到Seldon中。

前提条件

- 已创建DataScience集群，并且选择了Kubeflow服务，详情请参见创建集群。
- 已进行keras-API训练，详情请参见基于Kubeflow的Training示例。
- 下载dsdemo代码：请已创建DataScience集群的用户，使用钉钉搜索钉钉群号32497587加入钉钉群以获取dsdemo代码。

操作步骤

> **注意** 本示例使用的是keras-api训练的导出模型，详情请参见基于Kubeflow的Training示例。

1. 通过SSH方式连接集群，详情请参见登录集群。

2. 执行以下命令，进入*mnist_from_pvcmodel*目录。

```
cd dsdemo/kubeflow_samples/serving/seldon/tf/mnist_from_pvcmodel/
```

3. 执行以下命令，安装*seldon_core*。

```
pip3.7 install seldon_core
```

4. 根据实际需求，配置*mnist_grpc.yaml*文件。

```
apiVersion: machinelearning.seldon.io/v1alpha2
kind: SeldonDeployment
metadata:
  name: tfserving
spec:
  name: mnist
  predictors:
  - graph:
      children: []
      implementation: TENSORFLOW_SERVER
      modelUri: "pvc://strategy-volume/saved_model/master/"
      name: mnist-model
      parameters:
        - name: signature_name
          type: STRING
          value: serving_default
        - name: model_name
          type: STRING
          value: mnist-model
        - name: model_input
          type: STRING
          value: images
        - name: model_output
          type: STRING
          value: scores
    name: default
    replicas: 1
```

5. 执行以下命令，获取istio-gateway的IP地址。

```
kubectl get svc istio-ingressgateway -n istio-system
```

返回如下类似信息。

```
# kubectl get svc istio-ingressgateway -n istio-system
NAME                   TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
istio-ingressgateway   NodePort   10.104.**.**   <none>        15020:31872/TCP,80:31380/TCP,443:31390/TCP,31400:31400/T
CP,15029:30016/TCP,15030:30264/TCP,15031:31961/TCP,15032:31309/TCP,15443:31254/TCP   86m
```

> **说明** `CLUSTER-IP`对应的值即为istio-gateway的IP地址。

6. 模型预测GRPC协议或REST协议。

> **说明** 请根据实际需求修改*predict_rest.py*或*predict_grpc.py*中*minikube_ambassador_endpoint*地址，修改为步骤5中获取到的istio-gateway的IP地址。

- 模型预测GRPC协议。

```
python3.7 predict_grpc.py
```

- 模型预测REST协议。

```
python3.7 predict_rest.py
```

返回如下类似信息。

```
Response:
{'data': {'names': ['t:0', 't:1', 't:2', 't:3', 't:4', 't:5', 't:6', 't:7', 't:8', 't:9'], 'tensor': {'shape': [1, 10],
'values': [0.0191311873, 9.02173269e-09, 0.000745186175, 0.000349402311, 3.86572756e-05, 0.000118751872, 0.00138262415,
1.31301803e-07, 0.978211641, 2.23898933e-05]}}, 'meta': {}}
```

按照如上方式，即可将TensorFlow、PyTorch和Python等服务部署在Seldon中，实现在线服务。

问题反馈

如果您在使用DataScience集群过程中有任何疑问或问题，请提交工单或联系我们的技术人员协助处理，同时也欢迎您使用钉钉搜索钉钉群号32497587加入钉钉群进行反馈或交流。

6.2.36.3. Kubeflow Easyrec Pipeline示例

本文通过示例为您介绍如何使用Easyrec算法库进行模型训练、部署在线服务，并形成例行化Pipeline工作流。

前提条件

- 已创建DataScience集群，并且选择了Kubeflow服务，详情请参见创建集群。
- 本地安装了PuTTY和文件传输工具（SSH Secure File Transfer Client）。
- 下载dsdemo代码：请已创建DataScience集群的用户，使用钉钉搜索钉钉群号32497587加入钉钉群以获取dsdemo代码。

操作流程

1. 步骤一：准备工作
2. 步骤二：提交任务
3. （可选）步骤三：制作Hive CLI、Spark CLI、dscontroller、Hue、notebook或httpd镜像
4. 步骤四：编译Pipeline
5. 步骤五：上传Pipeline文件
6. 步骤六：创建并运行Experiments
7. （可选）步骤七：查看Pipeline状态
8. 步骤八：模型预测
9. 步骤九：通过PairecEngine部署在线服务

步骤一：准备工作

1. （可选）安装软件包。
   i. 通过SSH方式连接集群，详情请参见登录集群。
   ii. 在Header节点执行以下命令，安装seldon_core kfp。

   ```
   pip3.7 install seldon_core kfp configobj
   ```

   > **说明** 如果已经安装seldon_core kfp，可以跳过该步骤。

2. 登录容器镜像服务控制台，开通个人免费版ACR，并创建命名空间。

   创建命名空间详情，请参见命名空间的基本操作。

   > **说明** 如果您选用的是ACR企业版，则可以设置对VPC开放，以提升安全性。

3. 修改*config*文件的REGISTRY地址和experiment命名空间名称，并登录ACR。
   i. 执行以下命令，进入*ml_on_ds*目录。

   ```
   cd /root/dsdemo/ml_on_ds
   ```

   ii. 执行以下命令，查看*config*文件中的REGISTRY地址。

   ```
   cat config
   ```

   返回如下类似信息。

   ```
   # cat config
   # !!! Extremely Important !!!
   ```

```
# !!! You must use A NEW EXP different from others !!!
EXP=exp1
#ACR
REGISTRY=registry.cn-beijing.aliyuncs.com
NAMESPACE=dsexperiment
#PREFIX, could be a magic code.
PREFIX=prefix
#k8s namespace
KUBERNETES_NAMESPACE=default
#sc
NFSPATH=/mnt/disk1/k8s_pv/default_storage_class/
#NFSPATH=/mnt/disk1/nfs/ifs/kubernetes/
# region
REGIONID=cn-default
# emr-datascience clusterid
CLUSTERID="C-DEFAULT"
#HDFSADDR, train/test dir should be exist under $HDFSADDR, like
#user
#└── easy_rec
#    ├── 20210917
#    │   ├── test
#    │   │   ├── test0.csv
#    │   │   └── _SUCCESS
#    │   └── train
#    │       ├── train0.csv
#    │       └── _SUCCESS
#    └── 20210918
#        ├── test
#        │   ├── test0.csv
#        │   └── _SUCCESS
#        └── train
#            ├── train0.csv
#            └── _SUCCESS
HDFSADDR=hdfs://emr-header-1:9000/user/easy_rec/dssm
MODELDIR=hdfs://emr-header-1:9000/user/easy_rec/dssm
REGEX="*.csv"
SUCCESSFILE=_SUCCESS
# for allinone.sh development based on supposed TODAY & WHEN & YESTERDAY
TODAY=20211128
WHEN=20211128190001
YESTERDAY=20211127
# for daytoday.sh & multidays training, use HDFSADDR, MODELDIR
START_DATE=20211114
END_DATE=20211130
#DSSM post processing, file or directory, and filed seperator
USERFEATURE=taobao_user_feature_data.csv
ITEMFEATURE=taobao_item_feature_data.csv
SEP=','
# faiss_mysql: mysql as user_embedding storage, faiss as itemembedding index.
# holo_holo: holo as user & item embedding , along with indexing.
VEC_ENGINE=faiss_mysql
MYSQL_HOST=mysql.bitnami
MYSQL_PORT=3306
MYSQL_USER=root
MYSQL_PASSWORD=emr-datascience
#wait before pod finished after easyrec's python process end.
#example: 30s  10m 1h
WAITBEFOREFINISHED=10s
#train
#PS_NUMBER take effect only on training.
#WORKER_NUMBER take effect on training and predict.
TRAINING_REPOSITORY=tf-easyrec-training
TRAINING_VERSION=latest
PS_NUMBER=2
WORKER_NUMBER=3
#hivecli
HIVE_REPOSITORY=ds_hivecli
HIVE_VERSION=latest
#sparkcli
SPARK_REPOSITORY=ds_sparkcli
SPARK_VERSION=latest
#ds-controller
```

```
DSCONTROLLER_REPOSITORY=ds_controller
DSCONTROLLER_VERSION=latest
#notebook
NOTEBOOK_REPOSITORY=ds_notebook
NOTEBOOK_VERSION=latest
#hue
HUE_REPOSITORY=ds_hue
HUE_VERSION=latest
#httpd
HTTPD_REPOSITORY=ds_httpd
HTTPD_VERSION=latest
#customize
CUSTOMIZE_REPOSITORY=ds_customize
CUSTOMIZE_VERSION=latest
# ak/sk for cluster resize
EMR_AKID=AAAAAAAA
EMR_AKSECRET=BBBBBBBB
HOSTGROUPTYPE=TASK
INSTANCETYPE=ecs.g6.4xlarge
NODECOUNT=1
SYSDISKCAPACITY=120
SYSDISKTYPE=CLOUD_SSD
DISKCAPACITY=480
DISKTYPE=CLOUD_SSD
DISKCOUNT=4
# model is export into pvc
PVC_NAME='easyrec-volume'
SAVEDMODELS_RESERVE_DAYS=7
HIVEDB='jdbc:hive2://192.168.**.**:10000/zqkd'
# for spark, run `cat /etc/hosts|grep header-` for multiple master.
HOST1='192.168.**.**    emr-header-1.cluster-61030 emr-header-1 iZhp34gkdhy70wlgtw6****'
HOST2='192.168.**.**    emr-header-2.cluster-61030 emr-header-2 iZhp34gkdhy70wlgtw6****'
HOST3='192.168.**.**    emr-header-3.cluster-61030 emr-header-3 iZhp34gkdhy70wlgtw6****'
#auc threshold
THRESHOLD=0.3
# sms alert
SMS_AKID=AAAAAAAA
SMS_AKSECRET=BBBBBBBB
SMS_TEMPLATEDCODE=SMS_220default
SMS_PHONENUMBERS="186212XXXXX,186211YYYYY"
SMS_SIGNATURE="mysignature"
EAS_AKID=AAAAAAAA
EAS_AKSECRET=BBBBBBBB
EAS_ENDPOINT=pai-eas.cn-beijing.aliyuncs.com
EAS_SERVICENAME=eas_test_service
# ak/sk for access oss
OSS_AKID=AAAAAAAA
OSS_AKSECRET=BBBBBBBB
OSS_ENDPOINT=oss-cn-huhehaote-internal.aliyuncs.com
OSS_BUCKETNAME=zixiaotest-huhehaote
# !!! Do not change !!!
OSS_OBJECTNAME=%%EXP%%_faissserver/item_embedding.faiss.svm
# ak/sk for access holo
HOLO_AKID=AAAAAAAA
HOLO_AKSECRET=BBBBBBBB
HOLO_ENDPOINT=hgprecn-cn-default-cn-beijing-vpc.hologres.aliyuncs.com
#tensorboard
TENSORBOARDPORT=6006
```

iii. 执行以下命令，登录您的ACR，以便后续push镜像。

```
docker login --username=<用户名> <your_REGISTRY>-registry.cn-beijing.cr.aliyuncs.com
```

> **说明** ACR需要开启匿名访问并开通公开访问权限，方便pull镜像。代码中的 `<用户名>` 为您在容器镜像服务ACR控制台上配置的访问凭证，配置详情请参见配置访问凭证。 `<your_REGISTRY>` 为您前一步中查看到的REGISTRY地址。

4. 挂载NAT网关访问ACR，详情请参见创建和管理公网NAT网关实例。
5. 准备测试数据。

> 注意 您可以将测试数据写到DataScience集群的HDFS中，也可以按需写到您自己的HDFS中，但需要保证网络畅通。

```
sh allinlone.sh
```

根据返回信息提示，选择 `ppd) Prepare data` 。

步骤二：提交任务

> 注意 需要修改*config*的REPOSITORY地址为您自己的ACR仓库地址、VERSION版本号以及experiment命名空间名称。

1. 执行*allinone.sh*文件。

```
sh allinone.sh
```

返回信息如下。

```
***Welcome to dsdemo***
0)         Exit                                                    k8s: default
ppd)       Prepare data            ppk)      Prepare k8s config
1|build)   build & push training image to ACR
buildall) build & push all images to ACR(slow)
2)         applytraining           3)        deletetraining          dck)   deletecheckpoint
4)         applyeval               5)        deleteeval              ser)   showevalresult
6)         applyexport             7)        deleteexport
8)         applyserving            9)        deleteserving
10)        applypredict            11)       deletepredict
mt)        multidaystraining       dmt)      deletemultidaystraining
me)        multidayseval           dme)      deletemultidayseval
cnt)       createnotebook          dnt)      deletenotebook          snt)   shownotebooklink
cft)       createsftp              dft)      deletesftp              sft)   showsftplink
che)       createhue               dhe)      deletehue               she)   showhuelink
chd)       createhttpd             dhd)      deletehttpd             shd)   showhttpdlink
a)         kubectl get tfjobs      b|sdep)   kubectl get sdep
mp|mpl)    compile mlpipeline      bp|bpl)   compile bgpipeline      bu)    bdupload
tb)        tensorboard             vc)       verifyconfigfile        spl)   showpaireclink
c)         kubectl top pods        log|logs) show pod logs           setnl) set k8s node label
e|clean)   make clean              cleanall) make cleanall           sml)   showmilvuslink
99)        kubectl get pods
>
输入相应数字,点击回车
```

2. 输入相应的数字，单击回车。

   按照顺序，分别输入 `2` 、 `4` 、 `6` 和 `8` 。其中， `2` 对应提交Training训练任务， `4` 对应提交Evaluate评估任务， `6` 对应提交Export导出任务， `8` 对应提交Serving任务。

   您可以通过Tensorboard查看训练过程中的auc曲线：

   i. 执行以下命令，进入*ml_on_ds*目录。

   ```
   cd /root/dsdemo/ml_on_ds
   ```

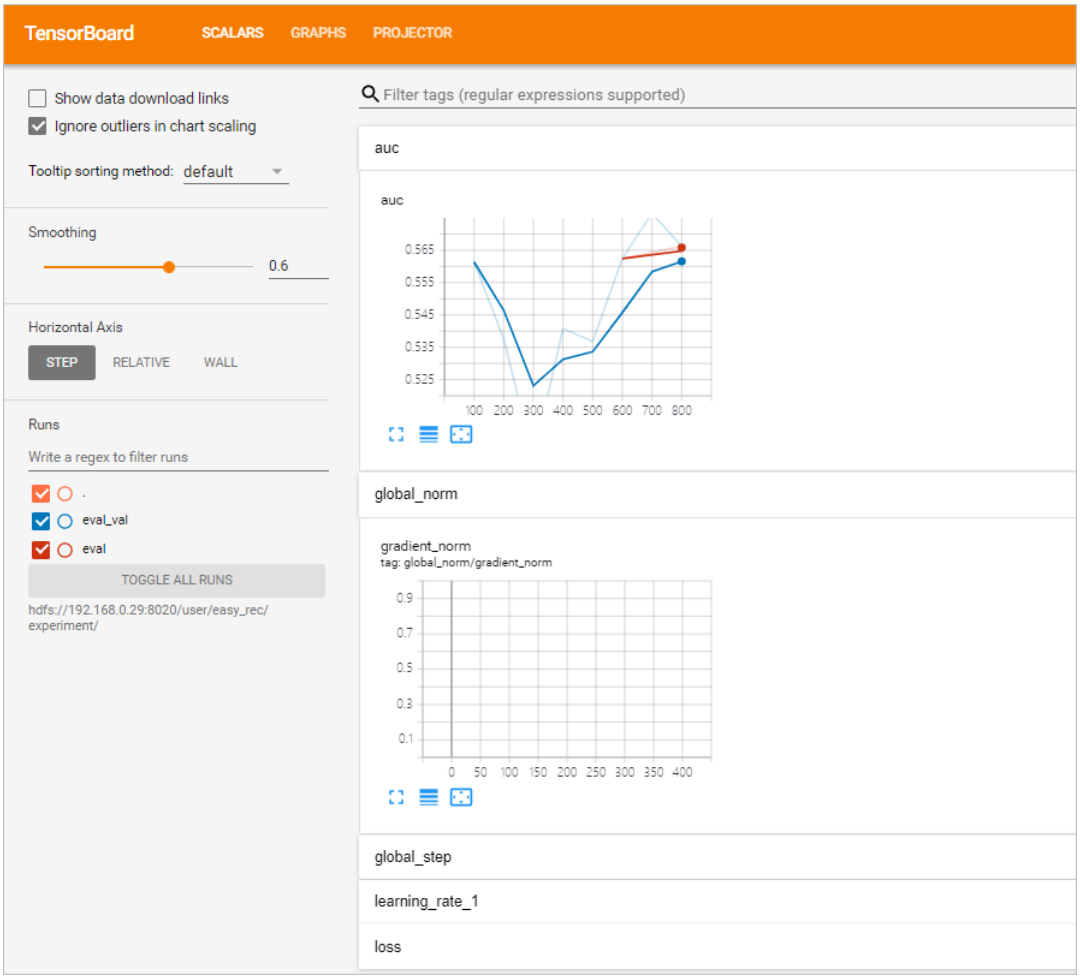
   ii. 执行以下命令，运行Tensorboard。

   ```
   sh run_tensorboard.sh
   ```

   选择 `tb` ，会显示当前实验的ckpt的Tensorboard信息，或者执行 `sh run_tensorboard.sh 20211209` 命令，查看20211209训练ckpt的Tensorboard信息。

   > 说明
   > - 默认使用config里TODAY_MODELDIR的modeldir。您也可以指定日期的modeldir，例如 `sh run_tensorboard.sh hdfs://192.168.**.**:9000/user/easy_rec/20210923/` 。
   > - 您可以自行修改*run_tensorboard.sh*脚本内容，调整相应的参数。

iii. 您可以在浏览器访问*http://<yourPublicIPAddress>:6006*，查看auc曲线。



（可选）步骤三：制作Hive CLI、Spark CLI、dscontroller、Hue、notebook或httpd镜像

> **说明**
> - 制作Hive CLI或Spark CLI镜像的目的是提交Hive或Spark任务进行大数据处理，生成待训练的数据，如果您已经自行准备好数据，可以跳过本步骤。如果是Spark任务，则会直接使用DataScience集群自带的Spark集群，如果是Hive任务，需要使用单独的Hadoop或Hive集群。
> - dscontroller镜像用来进行动态扩缩容。

- Hive CLI

  进入Hive CLI目录并制作镜像。

  ```
  cd hivecli && make
  ```

- Spark CLI

  进入Spark CLI目录并制作镜像。

  ```
  cd sparkcli && make
  ```

- dscontroller

  进入dscontroller目录并制作镜像。

  ```
  cd dscontroller && make
  ```

- Hue

  进入Hue目录并制作镜像。

  ```
  cd hue && make
  ```

- notebook

  进入notebook目录并制作镜像。

  ```
  cd notebook && make
  ```

- httpd

  进入httpd目录并制作镜像。

  ```
  cd httpd && make
  ```

步骤四：编译Pipeline

1. 执行以下命令，进入*/ml_on_ds*目录。

   ```
   cd /root/dsdemo/ml_on_ds
   ```

2. 执行以下命令，编译Pipeline。

   ```
   make pipeline
   ```

   > **说明** 您也可以执行命令 `sh allinone.sh` ，选择 `mpl` 来编译Pipeline。

   编译成功后生成*datascience_mlpipeline.tar.gz*文件。您可以使用文件传输工具将编译出来的*datascience_mlpipeline.tar.gz*文件，下载到本地PC，便于后续上传。

步骤五：上传Pipeline文件

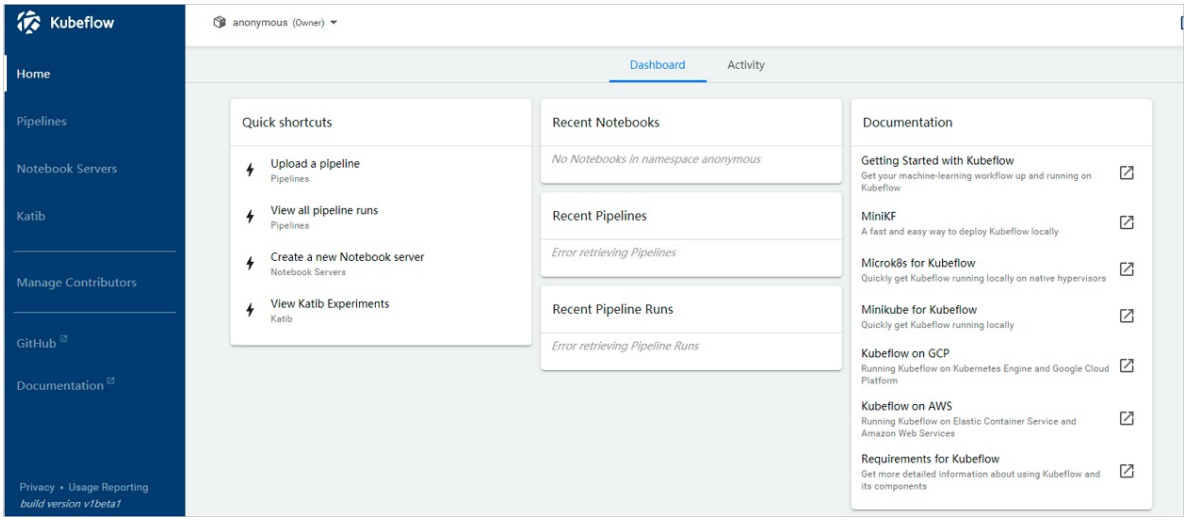
1. 进入集群详情页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
2. 在**集群基础信息**页面的**主机信息**区域，查看公网IP地址。

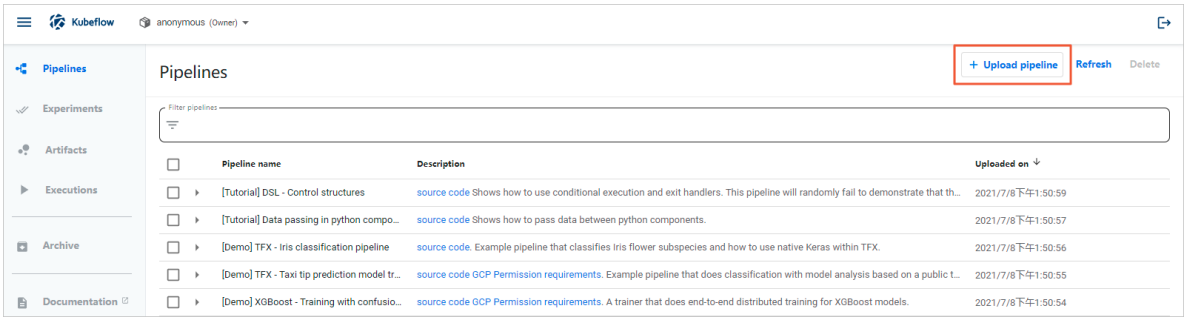
3. 在地址栏中，输入*http:///<yourPublicIPAddress>:31380*，按回车键。

   > **说明** <yourPublicIPAddress>为您前一步骤中，获取的公网IP地址。

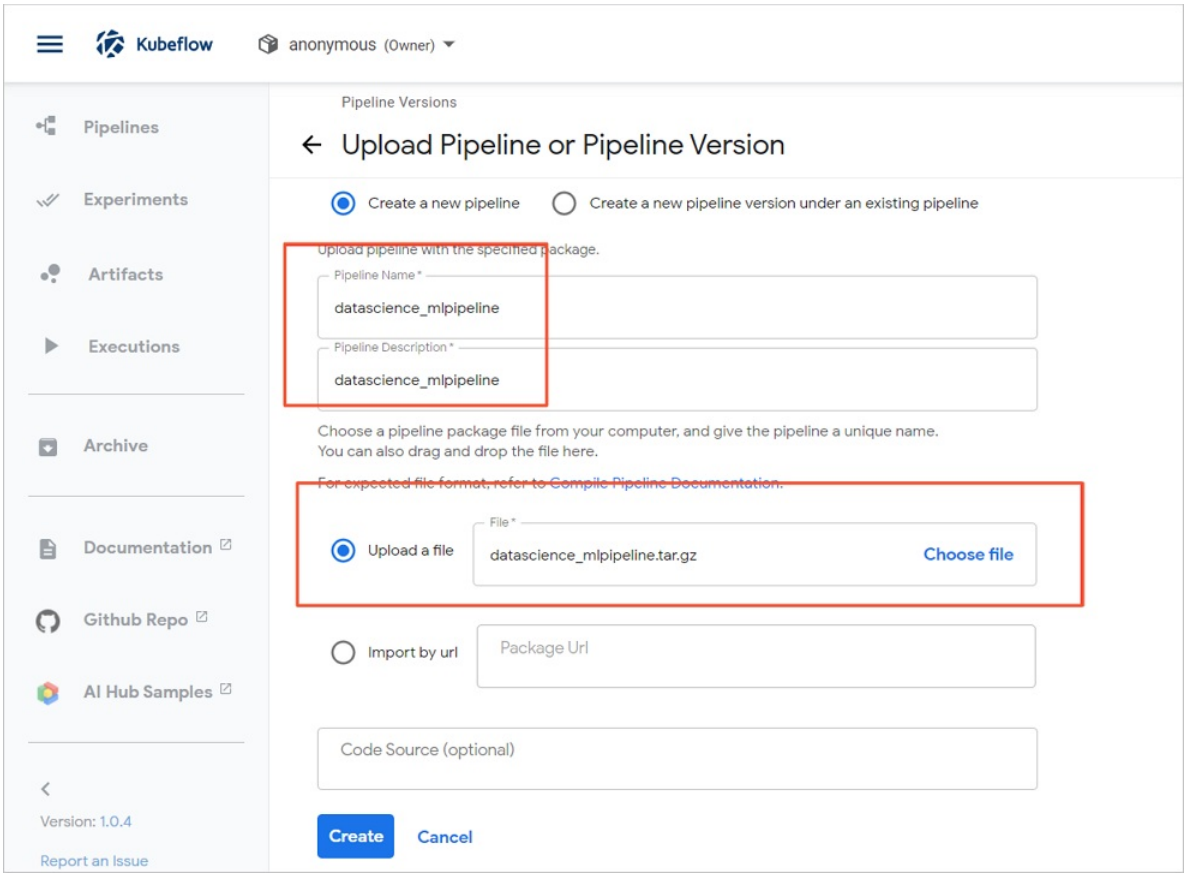
   使用默认的anonymous空间即可。进入后，默认页面如下。



4. 在左侧导航栏，单击**Pipelines**。
5. 在**Pipelines**页面，单击**Upload pipeline**。



6. 在**Upload Pipeline or Pipeline Version**，输入Pipeline Name，选择编译出的*datascience_mlpipeline.tar.gz*。

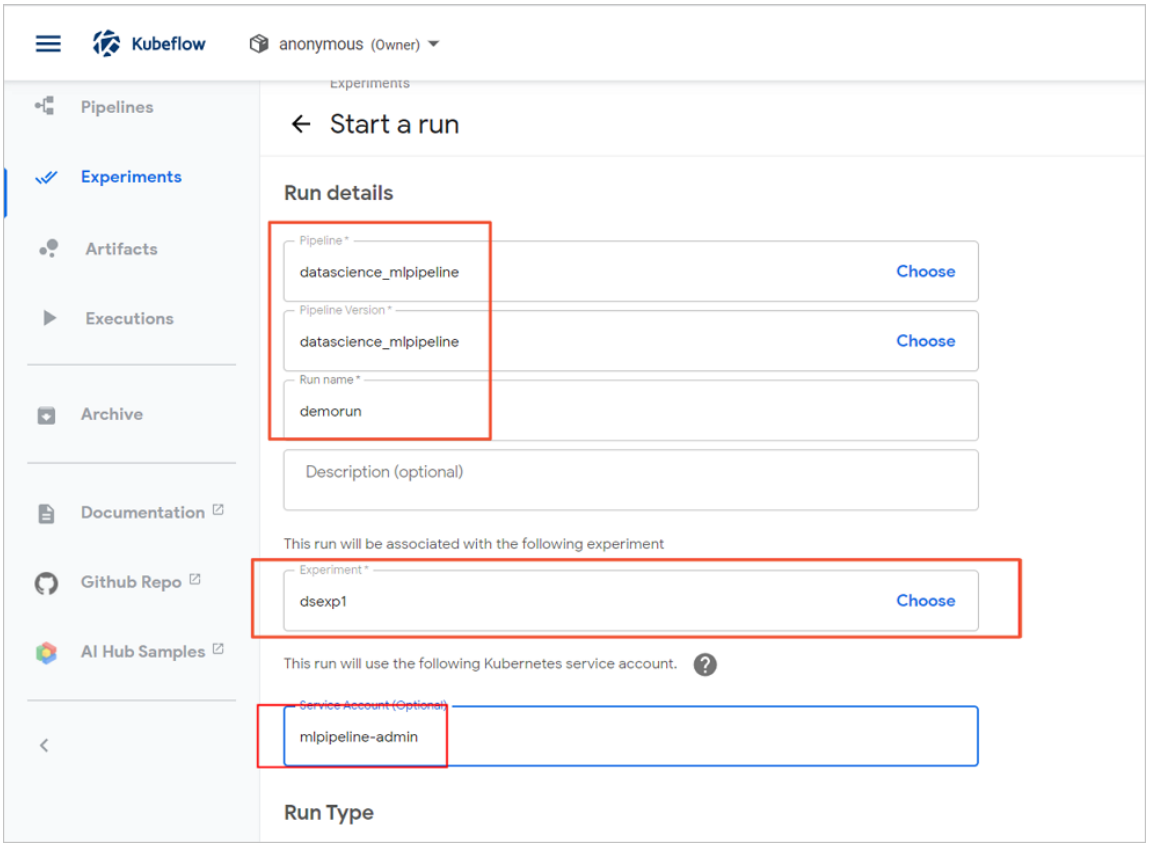


7. 单击Create。

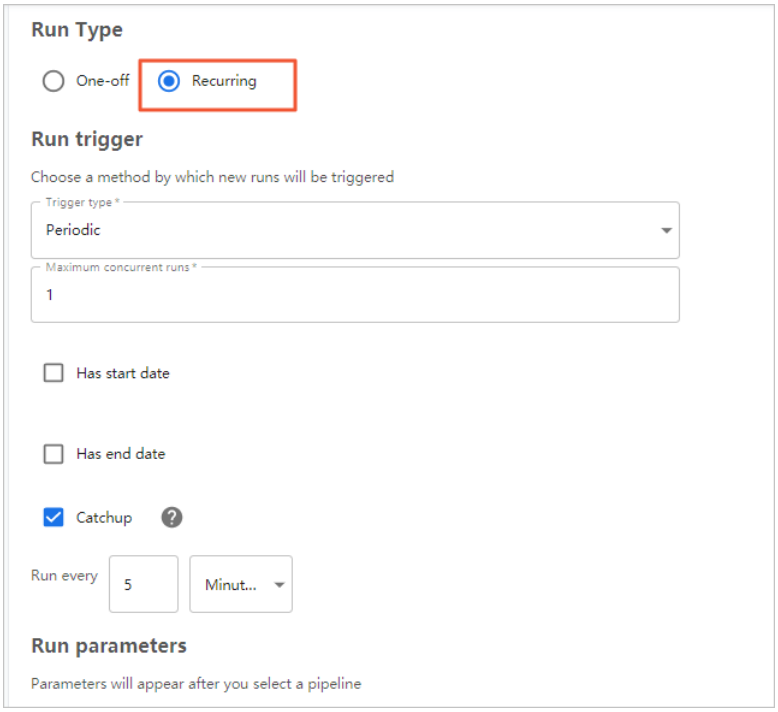
步骤六：创建并运行Experiments

1. 在Kubeflow的左侧导航栏，单击Experiments。
2. 单击上方的Create experiment。
3. 在New experiment页面，输入Experiment name。
4. 单击Next。
5. 在Start a run页面，配置参数。

i. 选择步骤四：编译Pipeline下载到本地的*datascience_mlpipeline.tar.gz*文件。



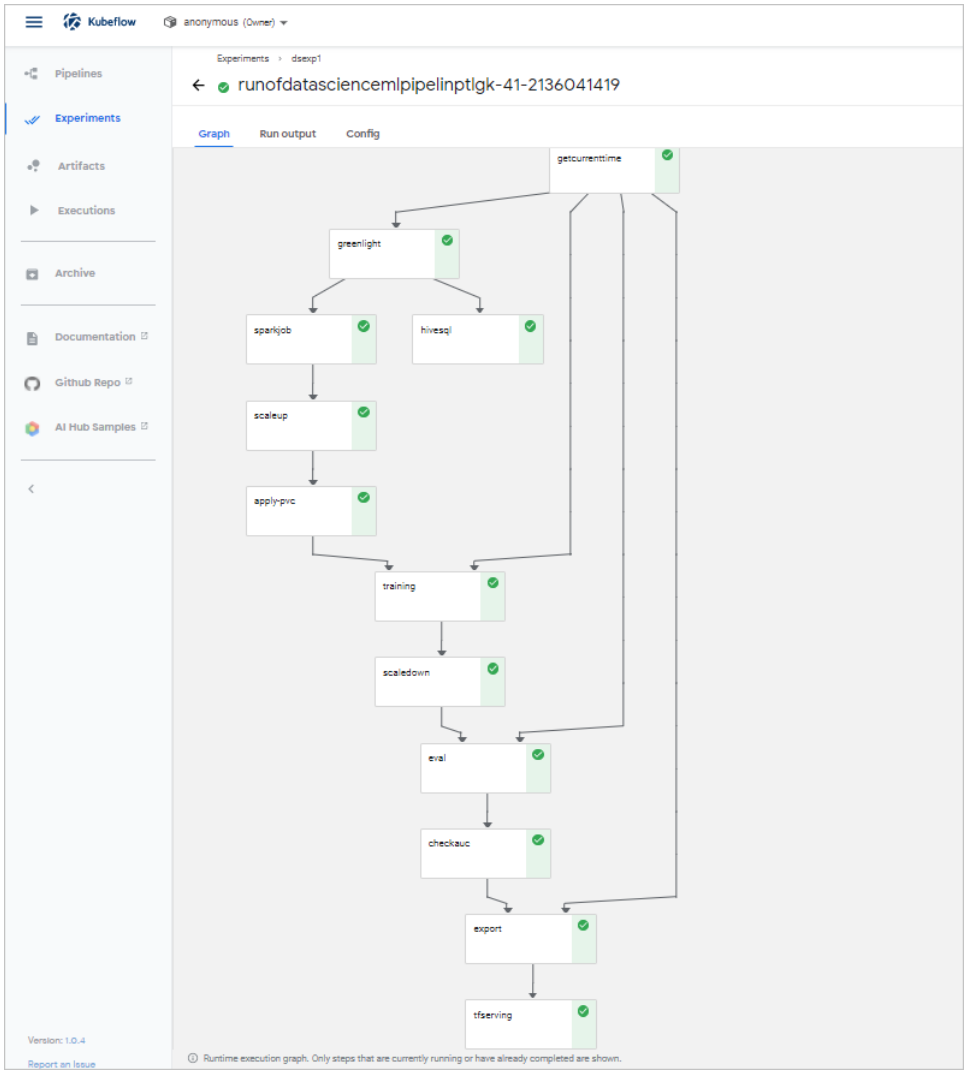
ii. 单击Recurring。



6. 单击Start。

（可选）步骤七：查看Pipeline状态

您可以在Experiments中查看Pipeline状态，模型示例展示如下。



步骤八：模型预测

- （推荐）使用HTTP请求方式

  此方式支持任何开发语言。预测代码请参见*predict_rest.sh*文件。

  您可以执行以下命令进行模型预测。

  > 注意 代码中的 `default` 是default命令空间， `easyrec-tfserving` 为部署Serving用的默认名称，请您按需调整。

```
!/bin/sh
curl -X POST http://127.0.0.1:31380/seldon/default/easyrec-tfserving/api/v1.0/predictions -H 'Content-Type: application/j
son' -d '
{
"jsonData": {
    "inputs": {
        "app_category":["10","10"],
        "app_domain":["1005","1005"],
        "app_id":["0","0"],
        "banner_pos":["85f751fd","4bf5bbe2"],
        "c1":["c4e18dd6","6b560cc1"],
        "c14":["50e219e0","28905ebd"],
        "c15":["0e8e4642","ecad2386"],
        "c16":["b408d42a","7801e8d9"],
        "c17":["09481d60","07d7df22"],
        "c18":["a99f214a","a99f214a"],
        "c19":["5deb445a","447d4613"],
        "c20":["f4fffcd0","cdf6ea96"],
        "c21":["1","1"],
        "device_conn_type":["0","0"],
        "device_id":["2098","2373"],
        "device_ip":["32","32"],
        "device_model":["5","5"],
        "device_type":["238","272"],
        "hour":["0","3"],
        "site_category":["56","5"],
        "site_domain":["0","0"],
        "site_id":["5","3"]
    }
}
}'
```

返回结果如下。

```
{"jsonData":{"outputs":{"logits":[-7.20718098,-4.15874624],"probs":[0.000740694755,0.0153866885]}},"meta":{}}
```

- 使用Seldon库方式

  执行以下命令，模型预测REST协议。

  ```
  python3.7 predict_rest.py
  ```

  返回信息如下。

  ```
  Response:
  {'jsonData': {'outputs': {'logits': [-2.66068792, 0.691401482], 'probs': [0.0653333142, 0.66627866]}}, 'meta': {}}
  ```

  > **说明** 预测代码请参见*predict_rest.py*文件。

步骤九：通过PairecEngine部署在线服务

详细信息，请参见PAI-Rec使用示例。

问题反馈

如果您在使用DataScience集群过程中有任何疑问或问题，请提交工单或联系我们的技术人员协助处理，同时也欢迎您使用钉钉搜索钉钉群号32497587加入钉钉群进行反馈或交流。

6.2.36.4. 转换自定义DAG为Pipeline

DataScience支持您将自定义DAG转换为Pipeline，并在KubeFlow上运行。本文通过示例为您介绍如何将自定义DAG转为Pipeline。

前提条件

- 已创建DataScience集群，并且选择了Kubeflow服务，详情请参见创建集群。
- 已通过SSH方式连接DataScience集群，详情请参见登录集群。

操作步骤

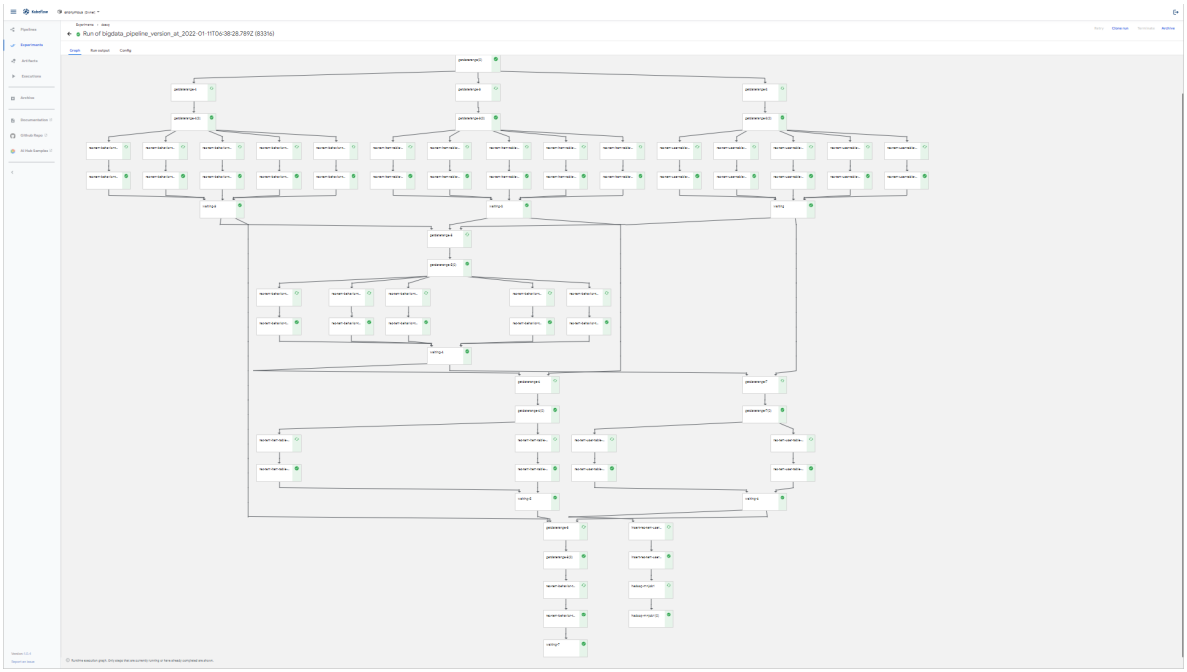
1. 准备数据。

   例如，将以下JSON表述的DAG运行在KubeFlow中，借助dag2pipeline，您可以自定义非常复杂的spark-sql进行特征工程，并设置好例行化

选项。

```
{
    "name": "DataScience",
    "link": "https://emr.console.aliyun.com/#/cn-beijing/cluster/create",
    "nodes": [{
            "name": "rec_tem_behavior_table_test_preprocess",
            "database": "xy_rec_sln_test",
            "type": "SPARKSQL_PARALLEL",
            "relative_start_date": -4,
            "relative_end_date": 0,
            "dependencies": [],
            "sqlfile": "test/feature/rec_tem_behavior_table_test_preprocess.sql",
            "args": [
                "--num-executors",
                "1"
            ],
            "comment": "sparksql parallel"
        },
        {
            "name": "rec_tem_user_table_test_preprocess",
            "database": "xy_rec_sln_test",
            "type": "SPARKSQL_PARALLEL",
            "relative_start_date": -4,
            "relative_end_date": 0,
            "dependencies": [],
            "sqlfile": "test/feature/rec_tem_user_table_test_preprocess.sql",
            "args": [
                "--num-executors",
                "1"
            ],
            "comment": "sparksql parallel"
        },
        {
            "name": "rec_tem_item_table_test_preprocess",
            "database": "xy_rec_sln_test",
            "type": "SPARKSQL_PARALLEL",
            "relative_start_date": -4,
            "relative_end_date": 0,
            "dependencies": [],
            "sqlfile": "test/feature/rec_tem_item_table_test_preprocess.sql",
            "args": [
                "--num-executors",
                "1"
            ],
            "comment": "sparksql parallel"
        },
        {
            "name": "rec_tem_behavior_table_test_preprocess_wide",
            "database": "xy_rec_sln_test",
            "type": "SPARKSQL_PARALLEL",
            "relative_start_date": -4,
            "relative_end_date": 0,
            "dependencies": [
                "rec_tem_item_table_test_preprocess",
                "rec_tem_user_table_test_preprocess",
                "rec_tem_behavior_table_test_preprocess"
            ],
            "sqlfile": "test/feature/rec_tem_behavior_table_test_preprocess_wide.sql",
            "args": [
                "--num-executors",
                "1"
            ],
            "comment": "sparksql parallel"
        },
        {
            "name": "rec_tem_user_table_test_preprocess_all_feature",
            "database": "xy_rec_sln_test",
            "type": "SPARKSQL_PARALLEL",
            "relative_start_date": -1,
            "relative_end_date": 0,
            "dependencies": [
                "rec_tem_behavior_table_test_preprocess_wide"
```

```
                "rec_tem_behavior_table_test_preprocess_wide",
                "rec_tem_user_table_test_preprocess"
            ],
            "sqlfile": "test/feature/rec_tem_user_table_test_preprocess_all_feature.sql",
            "args": [
                "--num-executors",
                "1"
            ],
            "comment": "sparksql parallel"
        },
        {
            "name": "rec_tem_item_table_test_preprocess_all_feature",
            "database": "xy_rec_sln_test",
            "type": "SPARKSQL_PARALLEL",
            "relative_start_date": -1,
            "relative_end_date": 0,
            "dependencies": [
                "rec_tem_behavior_table_test_preprocess_wide",
                "rec_tem_item_table_test_preprocess"
            ],
            "sqlfile": "test/feature/rec_tem_item_table_test_preprocess_all_feature.sql",
            "args": [
                "--num-executors",
                "1"
            ],
            "comment": "sparksql parallel"
        },
        {
            "name": "rec_tem_behavior_table_test_preprocess_rank_sample",
            "database": "xy_rec_sln_test",
            "type": "SPARKSQL_PARALLEL",
            "relative_start_date": 0,
            "relative_end_date": 0,
            "dependencies": [
                "rec_tem_item_table_test_preprocess_all_feature",
                "rec_tem_behavior_table_test_preprocess",
                "rec_tem_user_table_test_preprocess_all_feature"
            ],
            "sqlfile": "test/rank/rec_tem_behavior_table_test_preprocess_rank_sample.sql",
            "args": [
                "--num-executors",
                "1"
            ],
            "comment": "sparksql parallel"
        },
        {
            "name": "insert_rec_tem_user_table_test_preprocess_all_feature_holo",
            "type": "DI",
            "dependencies": [
                "rec_tem_user_table_test_preprocess_all_feature"
            ],
            "dataxjson": "test/feature/1.json",
            "comment": "DATAX, for more detail please access https://github.com/alibaba/datax"
        },
        {
            "name": "hadoop_mr_job1",
            "type": "HADOOP_MR",
            "dependencies": [
                "insert_rec_tem_user_table_test_preprocess_all_feature_holo"
            ],
            "jar": "hadoop-mapreduce-examples-2.8.5.jar",
            "classname": "wordcount",
            "args": [
                "/wordcount/input/",
                "/wordcount/output%%DATE%%"
            ],
            "comment": "jar file should be placed in bigdata directoray."
        }
    ]
}
```



上述代码示例中，您可以根据节点类型，将包放置到相应的目录。

- SPARKSQL_PARALLEL

  可以并行执行sparksql节点。代码详细信息如下所示。

```
{
    "name": "rec_tem_behavior_table_test_preprocess",
    "database": "xy_rec_sln_test",
    "type": "SPARKSQL_PARALLEL",
    "relative_start_date": -4,
    "relative_end_date": 0,
    "dependencies": [],
    "sqlfile": "test/feature/rec_tem_behavior_table_test_preprocess.sql",
    "args": [
        "--num-executors",
        "1"
    ],
    "comment": "sparksql parallel"
}
```

  代码示例中将test目录置于*bigdata/*目录下。

  其中，涉及参数描述如下：

  - relative_start_date和relative_end_date：并行执行*rec_tem_behavior_table_test_preprocess.sql*的起止日期。
  - args：spark-sql参数。
  - database：spark-sql运行的数据库名称。

- HADOOP_MR

  代码详细信息如下所示。

```
{
    "name": "hadoop_mr_job1",
    "type": "HADOOP_MR",
    "dependencies": [
        "insert_rec_tem_user_table_test_preprocess_all_feature_holo"
    ],
    "jar": "hadoop-mapreduce-examples-2.8.5.jar",
    "classname": "wordcount",
    "args": [
        "/wordcount/input/",
        "/wordcount/output%%DATE%%"
    ],
    "comment": "jar file should be placed in bigdata directoray."
}
```

代码示例中将*hadoop-mapreduce-examples-2.8.5.jar*置于*bigdata/*目录下。

其中，涉及参数描述如下：

- args：Hadoop Job的参数。
- jar：Hadoop Job的JAR包。
- classname：JAR包内的ClassName。

- DI/DATAX

  代码详细信息如下所示。

```
{
    "name": "insert_rec_tem_user_table_test_preprocess_all_feature_holo",
    "type": "DI",
    "dependencies": [
        "rec_tem_user_table_test_preprocess_all_feature"
    ],
    "dataxjson": "test/feature/1.json",
    "comment": "DATAX, for more detail please access https://github.com/alibaba/datax"
}
```

  代码示例中将*test*目录置于*bigdata/*下。

  其中，参数dataxjson表示DataX Job的参数，详细信息请参见DataX。

2. 执行以下命令，编译DAG生成bigdata_pipeline.tar.gz。

```
python3.7 tools/convertdag2pipeline.py bigdata/test.json
```

3. 上传和运行Pipeline。

   详细信息，请参见Kubeflow Easyrec Pipeline示例。

6.2.36.5. 配置钉钉机器人接收Kubeflow报警

设置钉钉机器人报警后，您可以通过指定钉钉群接收报警通知。本文为您介绍如何配置钉钉机器人接收Kubeflow报警。

前提条件

- 已创建DataScience集群，且选择了Kubeflow服务，详情请参见创建集群。
- 下载dsdemo代码：请已创建DataScience集群的用户，使用钉钉搜索钉钉群号32497587加入钉钉群以获取dsdemo代码。

操作步骤

1. 配置钉钉机器人并获取Webhook地址。

   在您想要添加报警机器人的钉钉群的右上角找到群机器人，然后添加一个自定义通过Webhook接入的机器人并进行安全设置，同时获取Webhook地址。

   具体操作请参见安全设置和获取自定义机器人Webhook。

   安全设置

   注意 安全设置勾选**自定义关键词**，使用alert_warn作为关键词。

   获取Webhook地址

注意 请保管好此Webhook地址，以备后用。Webhook地址格式为 `https://oapi.dingtalk.com/robot/send?access_token=*****` 。

2. 配置Kubeflow报警。
   i. 上传获取到的*dsdemo*.zip*至DataScience集群的header节点。
   ii. 通过SSH方式连接DataScience集群，详情请参见登录集群。
   iii. 解压*dsdemo*.zip*。
   iv. 修改*ml_on_ds*目录下的*config*文件。
   在*config*文件中新增ACCESS_TOKEN参数，参数值为上步骤中获取到的Webhook地址中的 `access_token` 。

```
# sms alert
SMS_AKID=AAAAAAAA
SMS_AKSECRET=BBBBBBBB
SMS_TEMPLATEDCODE=SMS_220default
SMS_PHONENUMBERS="186212XXXXX,186211YYYYY"
SMS_SIGNATURE="mysignature"

# dingtalk alert
# if ACCESS_TOKEN=ZZZZZZZZ choose sms alert else choose dingtalk alert
ACCESS_TOKEN=ZZZZZZZZ


EAS_AKID=AAAAAAAA
EAS_AKSECRET=BBBBBBBB
EAS_ENDPOINT=pai-eas.cn-beijing.aliyuncs.com
EAS_SERVICENAME=eas_test_service
```

6.2.37. PairecEngine

6.2.37.1. PAI-Rec使用示例

PAI-Rec是一款基于Go的在线推荐服务引擎的框架，您可以基于此框架快速搭建在线推荐服务，也可以定制化进行二次开发。本文为您介绍如何在DataScience集群上使用PAI-Rec搭建一个简单的在线推荐服务，实现U2I召回和特征加载。

背景信息

PAI-Rec框架内容如下：

- 集成Go http Server，提供路由注册功能，方便开发Restful API。
- 包含推荐引擎的Pipeline流程，里面预定义了多种召回、过滤、排序策略，内置访问阿里云PAI-EAS服务。
- 包含多种数据源的加载，支持Hologres、MySQL、Redis、Tablestore（OTS）和Kafka等。
- 基于灵活的配置描述推荐流程。
- 集成轻量级A/B Test实验平台。
- 支持简单易用的扩展点，方便自定义操作。

更多框架信息，请参见PAI-Rec文档。

前提条件

- 已创建DataScience集群，并且选择了PairecEngine和PostgreSQL服务，详情请参见创建集群。
- 通过SSH方式连接集群，详情请参见登录集群。

操作流程

1. 步骤一：数据准备
2. 步骤二：加载配置
3. 步骤三：推荐测试

步骤一：数据准备

因为要实现U2I召回和特征加载，所以需要准备相应的召回表和特征表。

1. 单击preparedata_mysql.sql，下载示例文件。

   该文件会帮您创建*db_pairecdemo*数据库及以下数据表：

   - 召回表：*t_recall*，包含字段user_id，表示用户ID，字段item_ids表示对应要召回的物品ID。
   - user特征表：*t_userfeature*，包含字段user_id和若干特征字段。
   - item特征表：*t_itemfeature*，包含字段item_id和若干特征字段。

   > **注意** 示例中的库、表和字段信息会在之后的推荐引擎配置中用到。其中召回表的结构定义是约定好的，不能随意更改配置。更多配置详情请参见配置总览。

2. 执行以下命令，获取MySQL服务的IP地址和端口。

   ```
   kubectl -n bitnami get svc mysql
   ```

3. 使用MySQL客户端连接服务，执行*preparedata_mysql.sql*文件。

   > **说明** 数据库root账号的默认密码为emr-datascience。

步骤二：加载配置

PAI-Rec推荐引擎的具体Pipeline流程由配置定义，目前配置支持启动时加载config配置文件和通过配置中心热加载两种方式，本示例为通过配置中心热加载的方式。

配置中心也一同部署在DataScience上。

1. 执行以下命令，获取端口。

   ```
   kubectl -n pairec get svc pairec-experiment -o jsonpath='{.spec.ports[0].nodePort}'
   ```

2. 访问*http://<header1节点的IP地址>:<端口>*链接，登录配置中心。

   默认用户名和密码均为admin。

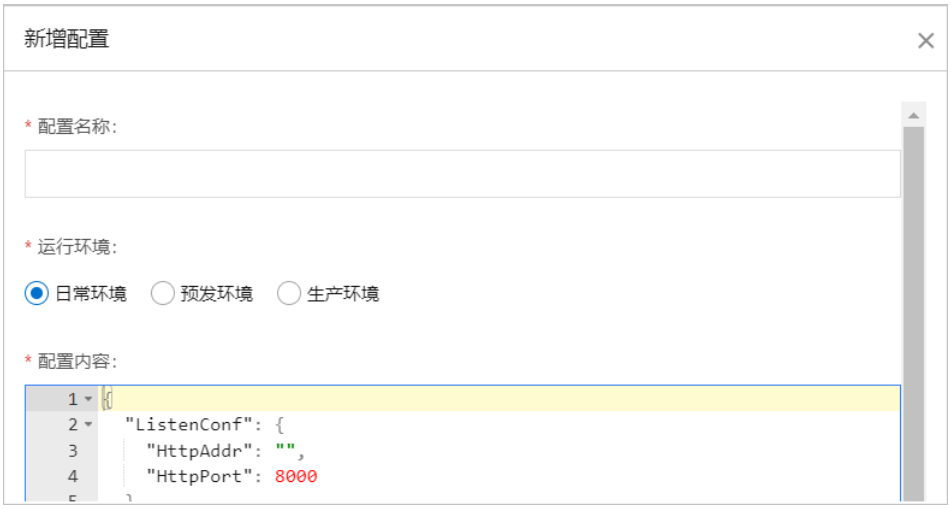
   > **说明** <端口>为步骤1获取到的。

3. 新增配置。

   i. 在左侧导航栏，选择**服务配置 > 推荐引擎配置**。

   ii. 在**推荐引擎配置**页面，单击**新增**。

   iii. 在**新增配置**对话框中，填写配置名称和配置内容，选择运行环境。



参数	描述
配置名称	填写为pairec_config。 DataScience上部署的PAI-Rec推荐引擎，默认使用日常环境下名称为pairec_config的配置。因此这里的配置名称设为pairec_config，运行环境设为日常环境。
运行环境	选择日常环境。
配置内容	详细示例请参见config.json。

配置内容分为以下几部分：

- 数据源配置

```
{
  // ...
  "MysqlConfs": {  // mysql 配置
      "pairec-mysql": { // 自定义名称
        "DSN": "root:emr-datascience@tcp(mysql.bitnami)/db_pairecdemo?parseTime=true&loc=Asia%2FShanghai"
      }
    },
  // ...
}
```

如果默认配置中没有MysqlConfs，新增即可。名称您可以自定义，会在之后用到。数据库设为之前数据准备中创建的db_pairecdemo。

- 召回配置

```
{
  // ...
    "RecallConfs": [
        {
            "Name": "user2item_recall",
            "RecallType": "UserCustomRecall",
            "ItemType": "video",
            "RecallCount": 500,
            "DaoConf": {
                "AdapterType": "mysql",
                "MysqlName": "pairec-mysql",
                "MysqlTable": "t_recall"
            }
        }
    ],
  // ...
  "SceneConfs": {
    "default_scene": {
      "default": {
        "RecallNames": [
          "user2item_recall"
        ]
      }
    }
  },
  // ...
}
```

需要在场景配置中启用配置好的user2item_recall。其中，涉及的参数如下表所示。

参数	描述
RecallType	填写为UserCustomRecall ，表示使用U2I的召回方式。
RecallCount	填写为500，表示最多召回500个item。
AdapterType	填写为mysql，表示使用MySQL数据源。
MysqlName	填写为pairec-mysql，表示使用上文中配置的数据源- pairec-mysql。
MysqlTable	填写为t_recall，表示召回表名称为t_recall。

- 特征配置

```
{
  // ...
  "FeatureConfs": {
    "default_scene": {
      "AsynLoadFeature": true,
      "FeatureLoadConfs": [
        {
          "FeatureDaoConf": {
            "AdapterType": "mysql",
            "MysqlName": "pairec-mysql",
            "FeatureKey": "user:uid",
            "UserFeatureKeyName": "user_id",
            "MysqlTable": "t_userfeature",
            "UserSelectFields": "c1,c14,c15,c16,c17,c18,c19,c20,c21",
            "FeatureStore": "user"
          },
          "Features": [
          ]
        },
        {
          "FeatureDaoConf": {
            "AdapterType": "mysql",
            "MysqlName": "pairec-mysql",
            "ItemFeatureKeyName": "item_id",
            "FeatureKey": "item:id",
            "MysqlTable": "t_itemfeature",
            "ItemSelectFields": "item_id,app_category,app_domain,app_id,banner_pos,device_conn_type,device_id,dev
ice_ip,device_model,device_type,hour,site_category,site_domain,site_id",
            "FeatureStore": "item"
          },
          "Features": [
          ]
        }
      ]
    }
  }
  // ...
}
```

其中，涉及的参数如下表所示。

参数	描述
AdapterType	填写为mysql，表示使用MySQL数据源。
MysqlName	填写为pairec-mysql，表示使用上文中配置的数据源。
MysqlTable	填写为t_recall，表示召回表名称为 t_recall。
UserFeatureKeyName	用于指定特征表查询条件中的字段名。
ItemFeatureKeyName	
FeatureKey	用于指定从哪里获取特征表查询条件中的字段值。
UserSelectFields	用于指定查询哪些特征。
ItemSelectFields	
FeatureStore	用于指定将查询得到的特征存到哪里。

4. 完成配置后，单击**确定**。
5. 保存配置后，单击配置操作列的**发布**。

   发布配置，引擎会自动加载设置好的配置。

步骤三：推荐测试

PAI-Rec引擎推荐接口为/api/rec/feed，访问此接口即可进行推荐测试（需要获取引擎服务地址）。

参数	描述	类型	是否必选	示例
uid	用户ID。	String	是	10****
size	获取item数量。	Integer	是	10
scene_id	场景ID。	String	是	feed
category	类目。	String	否	无
features	上下文特征。	JSON Map	否	{"age":20,"sex":"male"}
debug	打印更多的日志。	BOOL	否	true

您可以通过以下命令获取引擎端口，然后*http://<header1节点的IP地址>:<端口>*即为服务地址。

```
kubectl -n pairec get svc pairec-engine -o jsonpath='{.spec.ports[0].nodePort}'
```

假设服务地址为*http://127.0.0.1:32204*，则可以使用curl工具进行访问和测试。

```
curl http://127.0.0.1:32204/api/rec/feed -d '{"uid":"10****", "size":2, "debug":true}'
```

> **说明** 因为召回和特征都配置在default_scene，即默认场景下，所以此处无需指定scene_id。

当返回如下信息时，表示流程执行成功。如果推荐结果有问题或有其他需求，可以通过日志分析排查推荐流程，详情信息请参见日志分析。

```
{
  "code": 200,
  "msg": "success",
  "request_id": "16dda535-9d7f-4bbb-aaa9-dc7199f9f6e8",
  "size": 2,
  "items": [
    {
      "item_id": "200001",
      "score": 0,
      "retrieve_id": "user2item_recall"
    },
    {
      "item_id": "200002",
      "score": 0,
      "retrieve_id": "user2item_recall"
    }
  ]
}
```

日志分析

如果推荐结果有问题或有其他需求，可以通过日志分析排查推荐流程。

1. 执行以下命令，获取部署的PAI-Rec引擎的pod名称。

   ```
   kubectl get po -n pairec
   ```

   返回信息如下所示。

   ```
   # NAME                                READY   STATUS    RESTARTS   AGE
   # pairec-engine-9669c98fd-w9828       1/1     Running   0          2d4h
   ```

2. 执行以下命令，打印pod日志。

```
kubectl logs pairec-engine-9669c98fd-w9828 -n pairec
```

一次推荐流程的日志示例如下所示。

```
I1230 08:36:04.103712       6 log.go:13] [INFO] requestId=16dda535-9d7f-4bbb-aaa9-dc7199f9f6e8  event=begin     uri=/ap
i/rec/feed       address=10.244.0.0:39088        body={"uid":"10****", "size":2, "debug":true}
I1230 08:36:04.103814       6 logger.go:18] [ERROR]     scene:default_scene, not found the scene info
I1230 08:36:04.103828       6 log.go:13] [INFO] requestId=16dda535-9d7f-4bbb-aaa9-dc7199f9f6e8,uid=100001,scene_name=de
fault_scene,exp_id=
I1230 08:36:04.106789       6 log.go:13] [INFO] requestId=16dda535-9d7f-4bbb-aaa9-dc7199f9f6e8  module=UserCustomRecall
name=user2item_recall   count=3 cost=2
I1230 08:36:04.106805       6 log.go:13] [INFO] requestId=16dda535-9d7f-4bbb-aaa9-dc7199f9f6e8  module=recall    cost=2
I1230 08:36:04.106815       6 log.go:13] [INFO] requestId=16dda535-9d7f-4bbb-aaa9-dc7199f9f6e8  module=Filter    cost=0
I1230 08:36:04.109738       6 log.go:13] [INFO] requestId=16dda535-9d7f-4bbb-aaa9-dc7199f9f6e8  module=LoadFeatures
cost=2
&{100001  map[c1:1005 c14:2035 c15:30 c16:25 c17:234 c18:2 c19:4 c20:1002 c21:3 uid:10****] {{0 0} 0 0 0 0}}
I1230 08:36:04.109810       6 log.go:13] [INFO] requestId=16dda535-9d7f-4bbb-aaa9-dc7199f9f6e8  module=rank      cost=0
&{200001  0 user2item_recall video [] {{0 0} 0 0 0 0} map[app_category:07d7df22 app_domain:7801e8d9 app_id:ecad2386 ban
ner_pos:0 device_conn_type:0 device_id:a99f214a device_ip:9d046c7f device_model:9a45a8e8 device_type:1 hour:21 recall_n
ame:user2item_recall recall_score:0 site_category:3e814130 site_domain:7687a86e site_id:5b08c53b user2item_recall:0] ma
p[]}
&{200002  0 user2item_recall video [] {{0 0} 0 0 0 0} map[app_category:07d7df22 app_domain:7801e8d9 app_id:ecad2386 ban
ner_pos:0 device_conn_type:66 device_id:a99f214a device_ip:9d046c7f device_model:9a45a8e8 device_type:23 hour:21 recall
_name:user2item_recall recall_score:0 site_category:3e814130 site_domain:7687a86e site_id:5b08c53b user2item_recall:0]
map[]}
I1230 08:36:04.109973       6 log.go:13] [INFO] requestId=16dda535-9d7f-4bbb-aaa9-dc7199f9f6e8  event=end       uri=/ap
i/rec/feed       cost=6
```

展示了Pipeline各个模块的执行情况。其中，count代表结果数量， cost代表花费时间。加载的特征也会打印出来。

6.2.38. EasyRec

6.2.38.1. 使用EasyRec读取Hive表

EasyRec支持csv和Parquet两种Hive文件存储格式。本文通过示例为您介绍，如何基于Hive在Data Science集群进行EasyRec模型训练、评估和预测。

前提条件

- 已创建Hadoop集群，详情请参见创建集群。
- 已创建DataScience集群，且选择了EasyRec和TensorFlow服务，详情请参见创建集群。
- 下载dsdemo代码：请已创建DataScience集群的用户，使用钉钉搜索钉钉群号32497587加入钉钉群以获取dsdemo代码。

操作步骤

1. 开放安全组。

   将DataScience集群的所有公网IP地址，添加至Hadoop集群的安全组中，端口为10000和9000，详情请参见添加安全组规则。

2. 修改*ml_on_ds*目录下的文件。

   i. 上传获取到的*dsdemo*.zip*至DataScience集群的header节点。

   ii. 通过SSH方式连接DataScience集群，详情请参见登录集群。

   iii. 解压*dsdemo*.zip*。

iv. 修改*ml_on_ds*目录下的*easyrec_model.config*文件。

使用*./ml_on_ds/testdata/dssm_hiveinput/dssm_hive_csv_input.config*或*dssm_hive_parquet_input.config*作为*easyrec_model.config*。

```
yes | cp ./testdata/dssm_hiveinput/dssm_hive_csv_input.config ./easyrec_model.config
```

*easyrec_model.config*的代码片段如下，根据集群信息修改host和username。

```
hive_train_input {
    host: "*.*.*.*"
    username: "admin"
    port:10000
}
hive_eval_input {
    host: "*.*.*.*"
    username: "admin"
    port:10000
}
train_config {
   ...
}
eval_config {
    ...
}
data_config {
  input_type: HiveInput
  eval_batch_size: 1024
  ...
}
```

参数	描述
host	Hadoop集群Header节点的外网IP地址。
username	Hadoop集群LDAP的用户名。默认为admin。
input_type	输入类型。EasyRec支持HiveInput和HiveParquetInput两种关于Hive的输入类型。

3. 生成镜像文件。

修改*ml_on_ds*目录下的*config*文件，设置 `DATABASE` 、 `TRAIN_TABLE_NAME` 、 `EVAL_TABLE_NAME` 、 `PREDICT_TABLE_NAME` 、 `PREDICT_OUTPUT_TABLE_NAME` 和 `PARTITION_NAME` 。设置好容器服务的地址，保证容器服务是私有的且为企业镜像。配置好之后执行 `make build push` 命令打包镜像。

4. 准备数据。

i. 在*ml_on_ds*目录下运行以下命令，以启动Hive。

```
spark-sql
```

ii. 执行以下命令，准备测试数据。

```
create database pai_online_projects location 'hdfs://192.168.*.*:9000/pai_online_projects.db';
----------------------csv数据----------------------
CREATE TABLE pai_online_projects.easyrec_demo_taobao_train_data(`clk` BIGINT, `buy` BIGINT, `pid` STRING, `adgroup_
id` STRING, `cate_id` STRING, `campaign_id` STRING, `customer` STRING, `brand` STRING, `user_id` STRING, `cms_segid
` STRING, `cms_group_id` STRING, `final_gender_code` STRING, `age_level` STRING, `pvalue_level` STRING, `shopping_l
evel` STRING, `occupation` STRING, `new_user_class_level` STRING, `tag_category_list` STRING, `tag_brand_list` STRI
NG, `price` BIGINT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY','
partitioned by(dt string)
LOCATION 'hdfs://192.168.*.*:9000/pai_online_projects.db/easyrec_demo_taobao_train_data';
CREATE TABLE pai_online_projects.easyrec_demo_taobao_test_data(`clk` BIGINT, `buy` BIGINT, `pid` STRING, `adgroup_i
d` STRING, `cate_id` STRING, `campaign_id` STRING, `customer` STRING, `brand` STRING, `user_id` STRING, `cms_segid`
STRING, `cms_group_id` STRING, `final_gender_code` STRING, `age_level` STRING, `pvalue_level` STRING, `shopping_lev
el` STRING, `occupation` STRING, `new_user_class_level` STRING, `tag_category_list` STRING, `tag_brand_list` STRING
, `price` BIGINT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY','
partitioned by(dt string)
LOCATION 'hdfs://192.168.*.*:9000/pai_online_projects.db/easyrec_demo_taobao_test_data';
load data local inpath './testdata/taobao/19700101/train/easyrec_demo_taobao_train_data_10000.csv' into table pai_o
nline_projects.easyrec_demo_taobao_train_data partition(dt = 'yyyymmdd');
load data local inpath './testdata/taobao/19700101/test/easyrec_demo_taobao_test_data_1000.csv' into table pai_onli
ne_projects.easyrec_demo_taobao_test_data partition(dt = 'yyyymmdd');
--------------------parquet数据--------------------
CREATE TABLE pai_online_projects.easyrec_demo_taobao_train_data_parquet(`clk` BIGINT, `buy` BIGINT, `pid` STRING, `
adgroup_id` STRING, `cate_id` STRING, `campaign_id` STRING, `customer` STRING, `brand` STRING, `user_id` STRING, `c
ms_segid` STRING, `cms_group_id` STRING, `final_gender_code` STRING, `age_level` STRING, `pvalue_level` STRING, `sh
opping_level` STRING, `occupation` STRING, `new_user_class_level` STRING, `tag_category_list` STRING, `tag_brand_li
st` STRING, `price` BIGINT)
STORED AS PARQUET
partitioned by(dt string)
LOCATION 'hdfs://192.168.*.*:9000/pai_online_projects.db/easyrec_demo_taobao_train_data_parquet';
CREATE TABLE pai_online_projects.easyrec_demo_taobao_test_data_parquet(`clk` BIGINT, `buy` BIGINT, `pid` STRING, `a
dgroup_id` STRING, `cate_id` STRING, `campaign_id` STRING, `customer` STRING, `brand` STRING, `user_id` STRING, `cm
s_segid` STRING, `cms_group_id` STRING, `final_gender_code` STRING, `age_level` STRING, `pvalue_level` STRING, `sho
pping_level` STRING, `occupation` STRING, `new_user_class_level` STRING, `tag_category_list` STRING, `tag_brand_lis
t` STRING, `price` BIGINT)
STORED AS PARQUET
partitioned by(dt string)
LOCATION 'hdfs://192.168.*.*:9000/pai_online_projects.db/easyrec_demo_taobao_test_data_parquet';
insert into pai_online_projects.easyrec_demo_taobao_train_data_parquet partition(dt='yyyymmdd')
select clk , buy , pid , adgroup_id , cate_id , campaign_id , customer , brand , user_id , cms_segid , cms_group_id
, final_gender_code , age_level , pvalue_level , shopping_level , occupation , new_user_class_level , tag_category_
list , tag_brand_list , price  from pai_online_projects.easyrec_demo_taobao_train_data where dt = yyyymmdd;
insert into pai_online_projects.easyrec_demo_taobao_test_data_parquet partition(dt='yyyymmdd')
select clk , buy , pid , adgroup_id , cate_id , campaign_id , customer , brand , user_id , cms_segid , cms_group_id
, final_gender_code , age_level , pvalue_level , shopping_level , occupation , new_user_class_level , tag_category_
list , tag_brand_list , price  from pai_online_projects.easyrec_demo_taobao_test_data where dt = yyyymmdd;
```

5. 执行以下命令，进行模型训练。

```
kubectl apply -f tfjob_easyrec_training_hive.yaml
```

6. 执行以下命令，进行模型评估。

```
kubectl apply -f tfjob_easyrec_eval_hive.yaml
```

7. 执行以下命令，进行模型预测。

```
kubectl apply -f tfjob_easyrec_predict_hive.yaml
```

6.2.38.2. 读取MaxCompute训练EasyRec模型

本文为您介绍如何在Data Science集群读取MaxCompute的数据，进行EasyRec模型训练。

前提条件

- 已创建DataScience集群，且选择了EasyRec和TensorFlow服务，详情请参见创建集群。
- 已创建MaxCompute项目，详情请参见创建MaxCompute项目。
- 下载dsdemo代码：请已创建DataScience集群的用户，使用钉钉搜索钉钉群号32497587加入钉钉群以获取dsdemo代码。

操作步骤

1. 连接容器服务。
   i. 登录容器镜像服务控制台，创建企业版实例详情，详情请参见创建企业版实例。
   ii. 在**仓库管理 > 命名空间**页面，创建命名空间。

   iii. 在**实例管理 > 访问凭证**页面，设置固定密码。

   iv. 在DataScience集群的header节点，通过 `docker login` 命令连接容器服务，详情请参见docker login。
2. 上传dsdemo代码至DataScience集群的header节点，并解压缩。
3. 通过SSH方式连接DataScience集群，详情请参见登录集群。
4. 修改文件。
   i. 修改*ml_on_ds/tools/*下的*odps_config.in*文件，添加访问MaxCompute的AccessKey和Endpoint。
   Endpoint详情，请参见Endpoint。
   ii. 修改*ml_on_ds*目录下的*config*文件，根据前面的容器服务路径、命名空间和区域信息修改相应的配置。
   iii. 修改模型的*easyrec_model.config*文件，将input_type的参数值修改为OdpsInputV3。

> **说明** OdpsInputV3是专门定制在DataScience集群读取MaxCompute表的Class。

5. 根据Python版本，选择对应common_io。
   - Python 2.7版本

```
pip install --user -U https://tfsmoke1.oss-cn-zhangjiakou.aliyuncs.com/tunnel_paiio/common_io/py2/common_io-0.1.0-cp27-cp27mu-linux_x86_64.whl
```

   - Python 3.6版本

```
pip3 install --user -U http://tfsmoke1.cn-hangzhou.oss.aliyun-inc.com/tunnel_paiio/common_io/py3/common_io-0.3.0-cp36-cp36m-linux_x86_64.whl
```

   - Python 3.7版本

```
pip3 install --user -U http://tfsmoke1.cn-hangzhou.oss.aliyun-inc.com/tunnel_paiio/common_io/py3/common_io-0.2.0-cp37
-cp37m-linux_x86_64.whl
```

6. 修改*ml_on_ds*目录下的*Dockerflie*文件，添加以下信息。

   请根据您Python版本，执行相应命令。

```
ADD ./common_io-0.3.0-cp36-cp36m-linux_x86_64.whl /tmp/
RUN pip3 install --user -U http://tfsmoke1.cn-hangzhou.oss.aliyun-inc.com/tunnel_paiio/common_io/py3/common_io-0.3.0-cp
36-cp36m-linux_x86_64.whl -i http://mirrors.cloud.aliyuncs.com/pypi/simple --trusted-host mirrors.cloud.aliyuncs.com
COPY ./odps_config.ini   /root/.odps_config.ini
```

7. 执行以下命令，打包镜像。

```
make build push
```

8. 修改*ml_on_ds*目录下的*tfjob_easyrec_training.yam*的数据输入。

```
- "--train_input_path"
- "odps://<pai_online_project>/tables/<easyrec_demo_taobao_train_data>"
- "--eval_input_path"
- "odps://<pai_online_project>/tables/<easyrec_demo_taobao_test_data>"
```

   说明 *<pai_online_project>*需要替换为您创建的MaxCompute项目名。*<easyrec_demo_taobao_train_data>*和*<easyrec_demo_taobao_test_data>*需要替换为您创建的MaxCompute表名。

9. 执行以下命令，进行模型训练。

```
kubectl apply -f tfjob_easyrec_training.yaml
```

6.3. ClickHouse

6.3.1. ClickHouse概述

开源大数据平台E-MapReduce（简称EMR）的ClickHouse提供了开源OLAP分析引擎ClickHouse的云上托管服务。EMR ClickHouse完全兼容开源版本的产品特性，同时提供集群快速部署、集群管理、扩容、缩容和监控告警等云上产品功能，并且在开源的基础上优化了ClickHouse的读写性能，提升了ClickHouse与EMR其他组件快速集成的能力。

特性

特性	描述
列式存储	相较于行式存储，列式存储在查询性能上更优。同时列式存储的数据压缩比更高，更加节省存储空间。
MPP架构	每个节点只访问本地内存和存储，节点信息交互和节点本身是并行处理的。查询性能好，易于扩展。 向量化引擎：为了高效的使用CPU，数据不仅仅按列存储，同时还按向量（列的一部分）进行处理，这样可以更加高效地使用CPU。
支持SQL	ClickHouse支持一种基于SQL的声明式查询语言，它在许多情况下与ANSI SQL标准相同。支持GROUP BY、ORDER BY、FROM、JOIN和IN查询以及非相关子查询。
实时的数据更新	ClickHouse支持在表中定义主键。为了使查询能够快速在主键中进行范围查找，数据总是以增量的方式有序的存储在MergeTree中。 近实时数据更新， Clickhouse支持近实时的数据插入、指标聚合以及索引创建。
支持索引	按照主键对数据进行排序，ClickHouse可以在几十毫秒以内完成对数据特定值或范围的查找。

典型应用场景

场景	描述
用户行为分析	行为分析系统的表可以制作成一张大的宽表，每个表包含大量的列，可以超过一千列。JOIN的形式相对少一点，可以实现路径分析、漏斗分析和路径转化等功能。
流量和监控	可以将系统和应用监控指标通过流式计算引擎Flink或Spark streaming将监控数据清洗处理以后，实时写入ClickHouse，然后结合Grafana进行可视化展示。

场景	描述
用户画像	可以将各种用户特征进行数据加工，制作成包含全部用户的一张或多张用户特征表，提供灵活的用户画像分析、支撑广告和圈人等业务需求。
实时BI报表	根据业务需求，可以实时制作一些及时产出的查询灵活的BI报表，实现秒级查询，绝大多数查询能够实时反馈。BI报表包括订单分析、营销效果分析和大促活动分析。

> **说明** 不合适的场景：
> - 没有完整的事务支持。
> - 缺少高频率、低延迟的修改或删除已存在数据的能力。
> - 仅能用于批量删除或修改数据。

6.3.2. 快速入门

6.3.2.1. 创建集群

本文为您介绍创建ClickHouse集群的详细操作步骤和相关配置。

前提条件

已在目标地域创建一个专有网络和交换机，详情请参见创建和管理专有网络和创建和管理交换机。

背景信息

机型、内存和磁盘的设置，请参见Usage Recommendations。

操作步骤

1. 进入创建集群页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
      - 地域：创建的集群将会在对应的地域内，一旦创建不能修改。
      - 资源组：默认显示账号全部资源。
   iii. 单击**创建集群**，进行创建。
2. 配置集群信息。

   创建集群时，您需要对集群进行软件配置、硬件配置和基础配置。

   > **注意** 集群创建完成后，除了集群名称以外，其他配置均无法修改，所以在创建时请仔细确认各项配置。

   i. 软件配置。

配置项	说明
集群类型	选择**ClickHouse**。
产品版本	默认最新的软件版本。
必选服务	默认的服务组件，后期可以在管理页面中启停服务。
高级设置	**软件自定义配置**：可指定JSON文件对集群中的基础软件（例如Hadoop、Spark和Hive等）进行配置，详细使用方法请参见软件配置。默认不开启。

ii. 硬件配置。

区域	配置项	说明
付费类型	**付费类型**	默认包年包月。当前支持的付费类型如下： ■ **按量付费**：一种后付费模式，即先使用再付费。按量付费是根据实际使用的小时数来支付费用，每小时计费一次，适合短期的测试任务或是灵活的动态任务。 ■ **包年包月**：一种预付费模式，即先付费再使用。 说明 建议测试场景下使用**按量付费**，测试正常后再新建一个**包年包月**的生产集群正式使用。
网络配置	**可用区**	可用区为在同一地域下的不同物理区域，可用区之间内网互通。通常使用默认的可用区即可。
	网络类型	默认专有网络。
	VPC	选择在该地域的VPC。如果没有可用的VPC，单击**创建VPC/子网（交换机）**前往新建。
	交换机	选择在对应VPC下可用区的交换机，如果在这个可用区没有可用的交换机，则需要新创建一个。
	安全组名称	默认选择已有的安全组。安全组详情请参见安全组概述。 您也可以单击**新建安全组**，然后直接输入安全组名称来新建一个安全组。 注意 禁止使用ECS上创建的企业安全组。
实例	**选型配置**	您可以根据需要选择实例规格，详情请参见实例规格族。 ■ **系统盘配置**：根据需要选择ESSD云盘、SSD云盘或者高效云盘。 ■ **系统盘大小**：根据需要调整磁盘容量，推荐至少120 GB。取值范围为60 ~ 500 GB。 ■ **数据盘配置**：根据需要选择ESSD云盘、SSD云盘或者高效云盘。 ■ **数据盘大小**：根据需要调整磁盘容量，推荐至少80 GB。取值范围为40 ~ 32768 GB。 ■ **ClickHouse数量**：默认4台。

iii. 基础配置。

在**基础信息**区域，配置如下参数。

> **注意** 暂不支持高级配置区域的参数，因此请勿设置。

区域	配置项	说明
基础信息	**集群名称**	集群的名字，长度限制为1~64个字符，仅可使用中文、字母、数字、中划线（-）和下划线（_）。
	Shard数量	分片的数量。不支持修改。 **说明** 创建集群时，Shard数量会被自动计算，`Shard数量 = ClickHouse数量 ÷ Replica数量`。请保证ClickHouse数量可以被Replica数量整除，否则无法创建集群。
	Replica数量	副本的数量。默认为2。 **说明** 如果需要保证ClickHouse集群的高可用，Replica数量至少为2。
	挂载公网	集群是否挂载弹性公网IP地址，默认不开启。 **说明** 不开启**挂载公网**，将无法使用EMR控制台访问链接与端口功能查看开源组件的Web UI。
	密钥对	关于密钥对的使用详情请参见SSH密钥对。
	登录密码	设置Master节点的登录密码，密码规则：8~30个字符，且必须同时包含大写字母、小写字母、数字和特殊字符。 特殊字符包括：感叹号（!）、at（@）、井号（#）、美元符号（$）、百分号（%）、乘方（^）、and（&）和星号（*）。
高级设置	**添加用户**	添加访问开源大数据软件Web UI的账号。
	权限设置	通过RAM角色为在集群上运行的应用程序提供调用其他阿里云服务所需的必要权限，无需调整，使用默认即可。 ■ **服务角色**：用户将权限授予EMR服务，允许EMR代表用户调用其他阿里云的服务，例如ECS和OSS。 ■ **ECS应用角色**：当用户的程序在EMR计算节点上运行时，可不填写阿里云AccessKey来访问相关的云服务（例如OSS），EMR会自动申请一个临时AccessKey来授权本次访问。**ECS应用角色**用于控制这个AccessKey的权限。
	数据盘加密	默认不开启。 打开**加密开关**，即启动对集群节点ECS中所有属性为云盘的数据盘进行加密的功能。默认使用服务密钥为用户的数据进行加密，也支持使用用户自选密钥为用户的数据进行加密。 **注意** 不支持加密本地盘。
	引导操作	可选配置，您可以在集群启动Hadoop前执行您自定义的脚本，详情请参见引导操作。
	标签	可选配置，您可以在创建集群时绑定标签，也可以在集群创建完成后，在集群详情页绑定标签，详情请参见设置标签。
	资源组	可选配置。详情请参见使用资源组。

> **说明** 页面右边会显示您所创建集群的配置清单以及集群费用。根据不同的付费类型，展示不同的价格信息。

3. 选中**E-MapReduce服务条款**复选框。
4. 单击**创建**。

创建集群后可以通过刷新页面来查看进度，当集群**状态**显示为**空闲**时，集群创建成功。

6.3.2.2. 快速使用ClickHouse

本文通过示例为您介绍，如何快速将数据随机写入ClickHouse集群各个节点的本地表。

前提条件

已创建ClickHouse集群，详情请参见创建集群。

操作步骤

1. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。
2. 执行以下命令，下载官方样例数据集。

```
curl  https://datasets.clickhouse.com/hits/tsv/hits_v1.tsv.xz  | unxz --threads=`nproc` > hits_v1.tsv
```

3. 执行如下命令，启动ClickHouse客户端。

```
clickhouse-client -m
```

4. 执行如下命令，创建数据库。

   可以使用on CLUSTER参数在集群的所有节点创建数据库， 默认集群标识为cluster_emr。

```
CREATE DATABASE IF NOT EXISTS demo on CLUSTER cluster_emr;
```

   返回信息如下所示。

```
emr-header-1.cluster-2578 :) CREATE DATABASE IF NOT EXISTS demo on CLUSTER cluster_emr;

CREATE DATABASE IF NOT EXISTS demo ON CLUSTER cluster_emr

┌─host──────────────────────────┬─port─┬─status─┬─error─┬─num_hosts_remaining─┬─num_hosts_active─┐
│ emr-worker-1.cluster-257      │ 9000 │      0 │       │                   3 │                0 │
│ emr-worker-3.cluster-257      │ 9000 │      0 │       │                   2 │                0 │
│ emr-header-1.cluster-257      │ 9000 │      0 │       │                   1 │                0 │
│ emr-worker-2.cluster-257      │ 9000 │      0 │       │                   0 │                0 │
└───────────────────────────────┴──────┴────────┴───────┴─────────────────────┴──────────────────┘

4 rows in set. Elapsed: 0.109 sec.
```

5. 在集群上的所有节点创建一张复制表（Replicated表）。

   复制表（Replicated表）会根据副本的个数，实现数据的多副本，并实现数据的最终一致性。

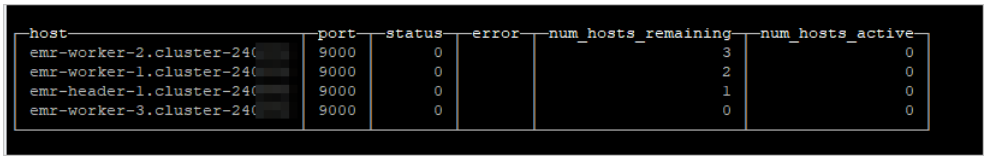
```
CREATE TABLE demo.hits_local ON CLUSTER cluster_emr
(
    `WatchID` UInt64,
    `JavaEnable` UInt8,
    `Title` String,
    `GoodEvent` Int16,
    `EventTime` DateTime,
    `EventDate` Date,
    `CounterID` UInt32,
    `ClientIP` UInt32,
    `ClientIP6` FixedString(16),
    `RegionID` UInt32,
    `UserID` UInt64,
    `CounterClass` Int8,
    `OS` UInt8,
    `UserAgent` UInt8,
    `URL` String,
    `Referer` String,
    `URLDomain` String,
    `RefererDomain` String,
    `Refresh` UInt8,
    `IsRobot` UInt8,
    `RefererCategories` Array(UInt16),
    `URLCategories` Array(UInt16),
    `URLRegions` Array(UInt32),
    `RefererRegions` Array(UInt32),
    `ResolutionWidth` UInt16,
    `ResolutionHeight` UInt16,
    `ResolutionDepth` UInt8,
    `FlashMajor` UInt8,
    `FlashMinor` UInt8,
    `FlashMinor2` String,
    `NetMajor` UInt8,
```

```
    `NetMinor` UInt8,
    `UserAgentMajor` UInt16,
    `UserAgentMinor` FixedString(2),
    `CookieEnable` UInt8,
    `JavascriptEnable` UInt8,
    `IsMobile` UInt8,
    `MobilePhone` UInt8,
    `MobilePhoneModel` String,
    `Params` String,
    `IPNetworkID` UInt32,
    `TraficSourceID` Int8,
    `SearchEngineID` UInt16,
    `SearchPhrase` String,
    `AdvEngineID` UInt8,
    `IsArtifical` UInt8,
    `WindowClientWidth` UInt16,
    `WindowClientHeight` UInt16,
    `ClientTimeZone` Int16,
    `ClientEventTime` DateTime,
    `SilverlightVersion1` UInt8,
    `SilverlightVersion2` UInt8,
    `SilverlightVersion3` UInt32,
    `SilverlightVersion4` UInt16,
    `PageCharset` String,
    `CodeVersion` UInt32,
    `IsLink` UInt8,
    `IsDownload` UInt8,
    `IsNotBounce` UInt8,
    `FUniqID` UInt64,
    `HID` UInt32,
    `IsOldCounter` UInt8,
    `IsEvent` UInt8,
    `IsParameter` UInt8,
    `DontCountHits` UInt8,
    `WithHash` UInt8,
    `HitColor` FixedString(1),
    `UTCEventTime` DateTime,
    `Age` UInt8,
    `Sex` UInt8,
    `Income` UInt8,
    `Interests` UInt16,
    `Robotness` UInt8,
    `GeneralInterests` Array(UInt16),
    `RemoteIP` UInt32,
    `RemoteIP6` FixedString(16),
    `WindowName` Int32,
    `OpenerName` Int32,
    `HistoryLength` Int16,
    `BrowserLanguage` FixedString(2),
    `BrowserCountry` FixedString(2),
    `SocialNetwork` String,
    `SocialAction` String,
    `HTTPError` UInt16,
    `SendTiming` Int32,
    `DNSTiming` Int32,
    `ConnectTiming` Int32,
    `ResponseStartTiming` Int32,
    `ResponseEndTiming` Int32,
    `FetchTiming` Int32,
    `RedirectTiming` Int32,
    `DOMInteractiveTiming` Int32,
    `DOMContentLoadedTiming` Int32,
    `DOMCompleteTiming` Int32,
    `LoadEventStartTiming` Int32,
    `LoadEventEndTiming` Int32,
    `NSToDOMContentLoadedTiming` Int32,
    `FirstPaintTiming` Int32,
    `RedirectCount` Int8,
    `SocialSourceNetworkID` UInt8,
    `SocialSourcePage` String,
    `ParamPrice` Int64,
    `ParamOrderID` String,
    `ParamCurrency` FixedString(3)
```

```
    `ParamCurrencyID` UInt16,
    `GoalsReached` Array(UInt32),
    `OpenstatServiceName` String,
    `OpenstatCampaignID` String,
    `OpenstatAdID` String,
    `OpenstatSourceID` String,
    `UTMSource` String,
    `UTMMedium` String,
    `UTMCampaign` String,
    `UTMContent` String,
    `UTMTerm` String,
    `FromTag` String,
    `HasGCLID` UInt8,
    `RefererHash` UInt64,
    `URLHash` UInt64,
    `CLID` UInt32,
    `YCLID` UInt64,
    `ShareService` String,
    `ShareURL` String,
    `ShareTitle` String,
    `ParsedParams` Nested(Key1 String,Key2 String,Key3 String,Key4 String,Key5 String,ValueDouble Float64),
    `IslandID` FixedString(16),
    `RequestNum` UInt32,
    `RequestTry` UInt8
)
ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}/{database}/hits_local', '{replica}')
PARTITION BY toYYYYMM(EventDate)
ORDER BY (CounterID, EventDate, intHash32(UserID))
SAMPLE BY intHash32(UserID);
```

> **说明** {shard}和{replica}是阿里云EMR为ClickHouse集群自动生成的宏定义，可以直接使用。

返回信息如下图所示。



host	port	status	error	num_hosts_remaining	num_hosts_active
emr-worker-2.cluster-24[redacted]	9000	0		3	0
emr-worker-1.cluster-24[redacted]	9000	0		2	0
emr-header-1.cluster-24[redacted]	9000	0		1	0
emr-worker-3.cluster-24[redacted]	9000	0		0	0

6. 执行以下命令，创建分布式（Distributed）表。

   分布式表不存储数据，仅仅是底层表的一个View，但可以在多个服务器上进行分布式查询。本例中使用随机函数rand()，表示数据会随机写入各个节点的本地表。

   ```
   CREATE TABLE demo.hits_all on CLUSTER cluster_emr AS demo.hits_local
   ENGINE = Distributed(cluster_emr, demo, hits_local, rand());
   ```

7. 退出ClickHouse客户端，在样例数据的目录下执行以下命令，导入数据。

   ```
   clickhouse-client --query "INSERT INTO demo.hits_all FORMAT TSV" --max_insert_block_size=100000 < hits_v1.tsv;
   ```

8. 重新启动ClickHouse客户端，查看数据。

   因为数据是随机写入的，各节点数据量可能不同。

   - 查看emr-header-1节点*demo.hits_all*的数据量。

     ```
     select count(*) from demo.hits_all;
     ```

     返回信息如下。

     ```
     ┌─count() ─┐
     │  8873898   │
     └──────┘
     ```

   - 查看emr-header-1节点*demo.hits_local*的数据量。

     ```
     select count(*) from demo.hits_local;
     ```

     返回信息如下。

```
┌─count() ─┐
│  4434571  │
└──────────┘
```

- 查看emr-worker-1节点*demo.hits_local*的数据量。

  > **说明** 其余节点，您也可以按照以下步骤来查看*demo.hits_local*的数据量。

  a. 执行以下命令，在Master节点切换到hadoop账号。

  ```
  su hadoop
  ```

  b. 执行以下命令，切换到emr-worker-1节点。

  ```
  ssh emr-worker-1;
  ```

  c. 在ClickHouse客户端，执行以下命令，查看*demo.hits_local*的数据量。

  ```
  select count(*) from demo.hits_local;
  ```

  返回信息如下。

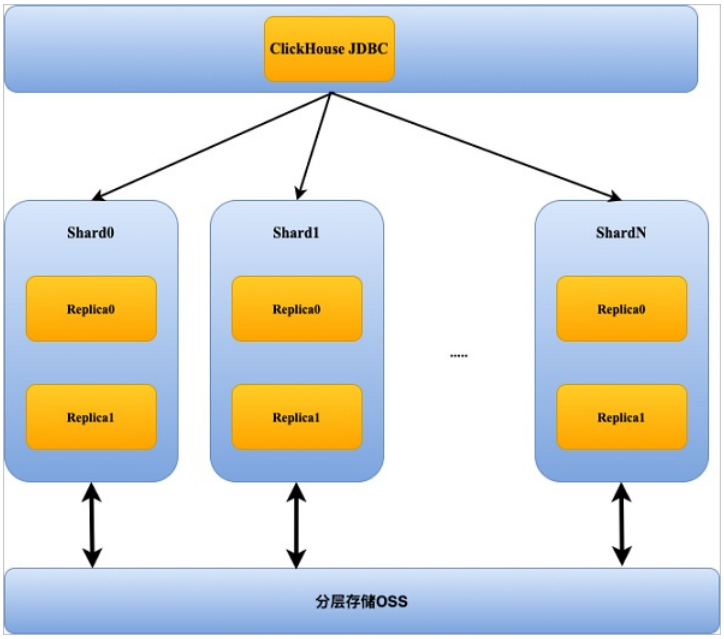
  ```
  ┌─count() ─┐
  │  4437327  │
  └──────────┘
  ```

6.3.2.3. 访问模式

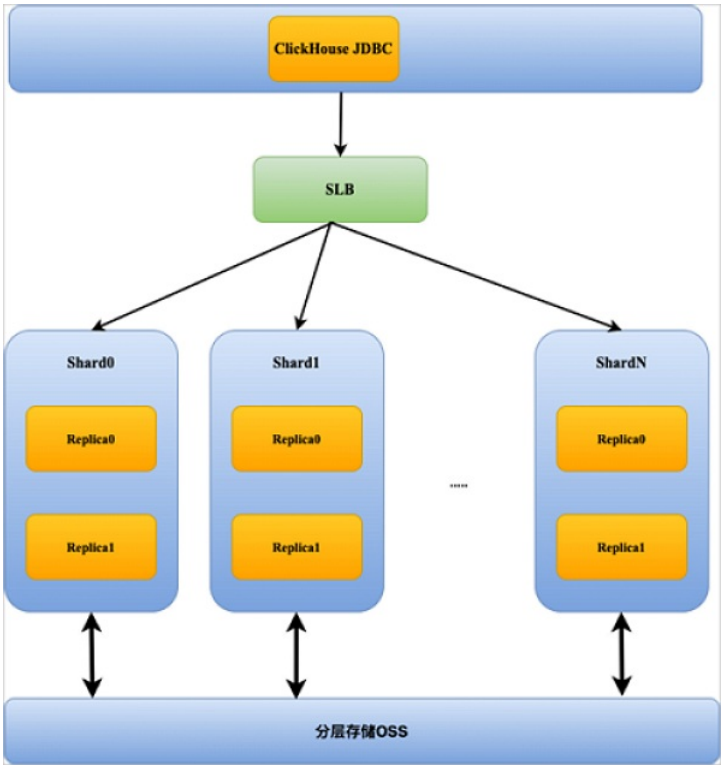
访问E-MapReduce（简称EMR）上的ClickHouse集群支持通过原生JDBC访问和通过负载均衡SLB访问两种方式。本文为您介绍如何通过这两种方式访问ClickHouse集群。

背景信息

- 通过原生JDBC访问ClickHouse集群的架构图如下。



- 通过负载均衡器SLB访问ClickHouse集群的架构图如下。



前提条件

- 已创建E-MapReduce的ClickHouse集群，详情请参见创建集群。
- 已创建SLB服务，详情请参见创建和管理CLB实例。

> **注意** 如果是想通过负载均衡器SLB访问ClickHouse集群，则需要创建SLB服务。并且在创建SLB服务时，如果**实例类型**选择的是**私网**，则在选择专有网络时，必须选择与EMR ClickHouse集群相同的VPC。

通过原生JDBC访问ClickHouse集群

1. 获取主机的IP地址。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。

v. 在左侧导航栏，单击**主机列表**。

在此页面您可以查看ClickHouse集群的IP地址。

2. 配置JDBC以访问ClickHouse集群，详情请参见ClickHouse JDBC driver。

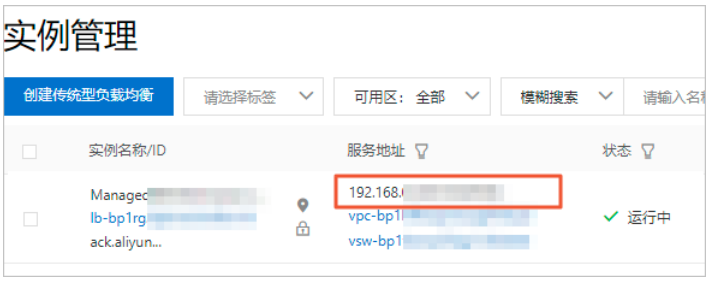
通过负载均衡器SLB访问ClickHouse集群

1. 配置SLB服务，详情请参见配置实例。

   通常情况下，ClickHouse使用SLB服务仅需要配置HTTP及TCP两种协议的监听，如果您有需要，也可以配置HTTPS的监听。配置监听详情，请参见添加TCP监听、添加HTTP监听和添加HTTPS监听。

   > 注意
   > - TCP监听所使用的虚拟服务器组，其端口应为ClickHouse通过TCP连接到服务器的端口，默认为9000。您可以在EMR控制台ClickHouse服务的**配置**页面，在搜索区域搜索**tcp_port**参数，参数值即为TCP端口。
   > - HTTP监听所使用的虚拟服务器组，其端口应为ClickHouse通过HTTP连接到服务器的端口，默认为8123。您可以在EMR控制台ClickHouse服务的**配置**页面，在搜索区域搜索**http_port**参数，参数值即为HTTP端口。

2. 在**实例管理**页面，获取SLB的服务地址。



3. 配置JDBC以访问ClickHouse集群，详情请参见ClickHouse JDBC driver。

6.3.3. ClickHouse运维

6.3.3.1. 日志配置说明

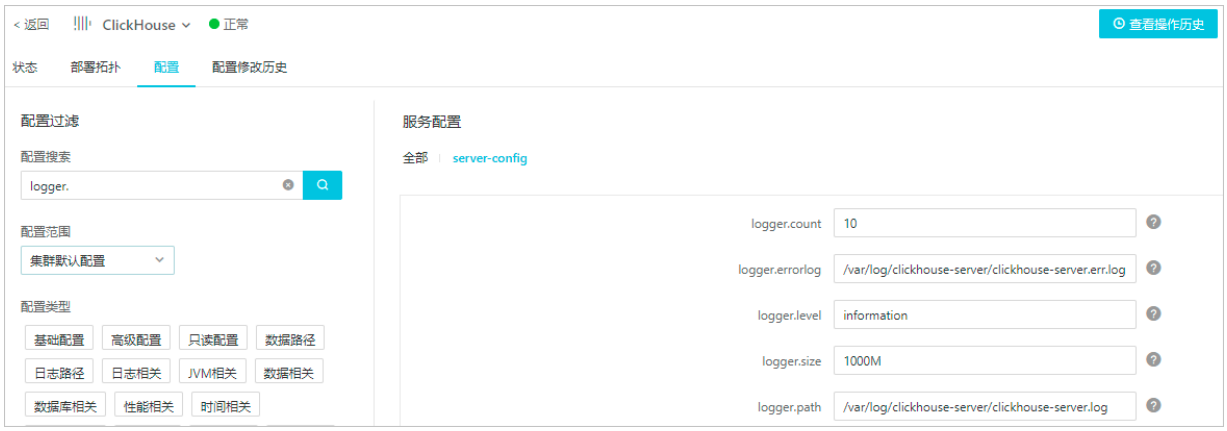
E-MapReduce（简称EMR）支持在控制台查看或配置日志参数，也支持在命令行中设置参数。本文为您介绍ClickHouse服务的日志配置。

前提条件

已创建ClickHouse集群，详情请参见创建集群。

Clickhouse控制台日志配置

您可以在ClickHouse服务**配置**页面的**服务配置**区域，在*server-config*页签中查看或修改配置，或者在ClickHouse服务的**配置**页面，在搜索区域搜索**logger.**，即可查看或修改所有的日志配置项。



参数	描述
logger.level	日志的等级，默认等级为information。可以配置的等级从严格到宽松依次为 • none：关闭日志。 • fatal：致命信息。 • critical：危险信息。 • error：错误信息。 • warning：警告信息。 • notice：普通但需要注意的信息。 • information（默认值）：重要或者您感兴趣的信息。 • debug：调试信息。 • trace：程序执行路径跟踪信息。
logger.path	ClickHouse Server正常输出的日志文件，默认为*/var/log/clickhouse-server/clickhouse-server.log*，会输出符合*logger.level*所指定的日志等级的日志。
logger.errorlog	ClickHouse Server中错误日志的输出路径。默认值为*/var/log/clickhouse-server/clickhouse-server.err.log*。
logger.size	日志文件的大小。当文件达到该参数设置的值时，ClickHouse会将其存档并重命名，并创建一个新的日志文件。默认值为**1000M**。
logger.count	存档的ClickHouse日志文件个数。当存档的日志文件个数达到该参数设置的值时，ClickHouse会将最早的存档删除。默认值为**10**。

ClickHouse客户端日志配置

您可以通过配置客户端日志，来接收来自服务端的日志，默认接收fatal级别的日志。

1. 通过SSH方式登录集群，详情请参见登录集群。
2. 基本操作示例。
   - 查看每次执行的日志。
     a. 执行以下命令，进入ClickHouse客户端。

     ```
     clickhouse-client -m
     ```

     b. 您可以执行以下命令，设置参数send_logs_level查看每次执行的日志。

     ```
     set send_logs_level='debug';
     ```

     返回信息如下所示。

     ```
     SET send_logs_level = 'debug'
     Ok.
     0 rows in set. Elapsed: 0.002 sec.
     ```

   - 在启动ClickHouse客户端时，您可以执行以下命令，将日志保存到指定的文件中。

     ```
     clickhouse-client -m --send_logs_level=trace --log-level=trace --server_logs_file='/tmp/query.log'
     ```

6.3.3.2. 系统表说明

系统表存储于System数据库中，仅提供数据读取功能，不能被删除或更改，但可以对其进行分离（detach）操作。大多数系统表将其数据存储在RAM中，一个ClickHouse服务在刚启动时便会创建此类系统表。本文为您介绍E-MapReduce（简称EMR）中常用的系统表。

背景信息

常用系统表如下：

- system.clusters
- system.query_log
- system.zookeeper
- system.replicas
- system.storage_policies
- system.disks

system.clusters

该表包含了配置文件中可用的集群及其服务器的信息。

参数	数据类型	描述
cluster	String	集群名。
shard_num	UInt32	集群中的分片数，从1开始。
shard_weight	UInt32	写数据时该分片的相对权重。
replica_num	UInt32	分片的副本数量，从1开始。
host_name	String	配置中指定的主机名。
host_address	String	从DNS获取的主机IP地址。
port	UInt16	连接到服务器的端口。
user	String	连接到服务器的用户名。
errors_count	UInt32	此主机无法访问副本的次数。
slowdowns_count	UInt32	在与对端请求建立连接时导致副本更改的slowdown的次数。
estimated_recovery_time	UInt32	在复制副本错误计数归零并被视为恢复正常之前剩余的秒数。

示例：您可以执行以下命令，查看表信息。

```
SELECT * FROM system.clusters LIMIT 2 FORMAT Vertical;
```

返回信息如下。

```
Row 1:
──────
cluster:                 cluster_emr
shard_num:               1
shard_weight:            1
replica_num:             1
host_name:               emr-header-1.cluster-24****
host_address:            192.168.**.**
port:                    9000
is_local:                1
user:                    default
default_database:
errors_count:            0
estimated_recovery_time: 0
Row 2:
──────
cluster:                 cluster_emr
shard_num:               1
shard_weight:            1
replica_num:             2
host_name:               emr-worker-1.cluster-24****
host_address:            192.168.**.**
port:                    9000
is_local:                0
user:                    default
default_database:
errors_count:            0
estimated_recovery_time: 0
```

system.query_log

该表包含了已执行查询的相关信息。例如，开始时间、处理持续时间和错误消息。

system.query_log表中记录了两种查询：

- 客户端直接运行的初始查询。
- 由其它查询启动的子查询（用于分布式查询执行）。 对于这些类型的查询，有关父查询的信息显示在 `initial_*` 列。

根据查询的状态（请参见 `type` 列），每个查询在查询日志表中创建一行或两行记录：

- 如果查询执行成功，则会创建 `type` 为 `QueryStart` 和 `QueryFinish` 的两行记录信息。
- 如果在查询处理期间发生错误，则会创建 `type` 为 `QueryStart` 和 `ExceptionWhileProcessing` 的两行记录信息。
- 如果在启动查询之前发生错误，则会创建 `type` 为 `ExceptionBeforeStart` 的一行记录信息。

参数	数据类型	描述
type	Enum8	执行查询时的事件类型。取值如下： • `'QueryStart' = 1` ：查询成功启动。 • `'QueryFinish' = 2` ：查询成功完成。 • `'ExceptionBeforeStart' = 3` ：查询执行前有异常。 • `'ExceptionWhileProcessing' = 4` ：查询执行期间有异常。
event_date	Date	查询开始日期。
event_time	DateTime	查询开始时间。
event_time_microseconds	DateTime64	以微秒精度查询开始时间。
query_start_time	DateTime	查询执行的开始时间。
query_start_time_microseconds	DateTime64	以微秒精度查询执行的开始时间。
query_duration_ms	UInt64	查询消耗的时间。单位为毫秒。
read_rows	UInt64	从参与了查询的所有表和表函数读取的总行数。包括常用的子查询，IN和JOIN的子查询。对于分布式查询 `read_rows` 包括在所有副本上读取的行总数。每个副本发送它的 `read_rows` 值，并且查询的发起方将所有接收到的和本地的值汇总。 缓存卷不会影响此值。

参数	数据类型	描述
read_bytes	UInt64	从参与了查询的所有表和表函数读取的总字节数。包括常用的子查询，IN和JOIN的子查询。对于分布式查询 `read_bytes` 包括在所有副本上读取的字节总数。 每个副本发送它的 `read_bytes` 值，并且查询的发起方将所有接收到的值和本地的值汇总。 缓存卷不会影响此值。
written_rows	UInt64	对于INSERT查询，为写入的行数。 对于其它查询，值为0。
written_bytes	UInt64	对于INSERT查询时，为写入的字节数。 对于其它查询，值为0。
result_rows	UInt64	SELECT查询结果的行数，或INSERT的行数。
result_bytes	UInt64	存储查询结果的RAM量。
memory_usage	UInt64	查询使用的内存。
query	String	查询语句。
exception	String	异常信息。
exception_code	Int32	异常码。
stack_trace	String	如果查询成功完成，则为空字符串。
is_initial_query	UInt8	查询类型。取值如下： • 0：由另一个查询发起的，作为分布式查询的一部分。 • 1：客户端发起的查询。
user	String	发起查询的用户。
query_id	String	查询ID。
address	Ipv6	发起查询的客户端IP地址。
port	UInt16	发起查询的客户端端口。
initial_user	String	初始查询的用户名（用于分布式查询执行）。
initial_query_id	String	初始查询的ID（用于分布式查询执行）。
initial_address	Ipv6	运行父查询的IP地址。
initial_port	UInt16	进行父查询的客户端端口。
interface	UInt8	发起查询的接口。取值如下： • 1：TCP • 2：HTTP
os_user	String	运行 `clickhouse-client` 的操作系统的用户名。
client_hostname	String	运行 `clickhouse-client` 或其他TCP客户端的机器的主机名。
client_name	String	`clickhouse-client` 或其他TCP客户端的名称。
client_revision	UInt32	`clickhouse-client` 或其他TCP客户端的Revision。
client_version_major	UInt32	`clickhouse-client` 或其他TCP客户端的Major Version。
client_version_minor	UInt32	`clickhouse-client` 或其他TCP客户端的Minor Version。
client_version_patch	UInt32	`clickhouse-client` 或其他TCP客户端的Patch component。
http_method	UInt8	发起查询的HTTP方法。取值如下： • 0：TCP接口的查询 • 1：GET • 2：POST
http_user_agent	String	HTTP查询中传递的HTTP请求头UserAgent。

参数	数据类型	描述
quota_key	String	在quotas配置里设置的quota key. 详细信息可以参见配额。
revision	UInt32	ClickHouse revision。
ProfileEvents	Map（String，UInt64）	其它事件的指标，可以在表system.events中找到相关的描述。
Settings	Map（String，String）	客户端运行查询时更改的设置。要启用对设置的日志记录更改，请将 `log_query_settings` 参数设置为1。
thread_ids	Array（UInt64）	参与查询的线程数。
Settings.Names	Array（String）	客户端运行查询时更改的设置的名称。 要启用对设置的日志记录更改，请将 `log_query_settings` 参数设置为1。
Settings.Values	Array（String）	Settings.Names列中列出的设置的值。

示例：您可以执行以下命令，查看表信息。

```
SELECT * FROM system.query_log LIMIT 1 FORMAT Vertical;
```

返回信息如下。

```
Row 1:
──────
type:                 QueryStart
event_date:           2021-08-12
event_time:           2021-08-12 14:11:58
query_start_time:     2021-08-12 14:11:58
query_duration_ms:    0
read_rows:            0
read_bytes:           0
written_rows:         0
written_bytes:        0
result_rows:          0
result_bytes:         0
memory_usage:         0
current_database:     default
query:                SELECT * FROM system.query_log LIMIT 1 FORMAT Vertical;
exception_code:       0
exception:
stack_trace:
is_initial_query:     1
user:                 default
query_id:             08e1336c-696f-4fad-b01b-255b77e56b1f
address:              ::ffff:127.0.0.1
port:                 60500
initial_user:         default
initial_query_id:     08e1336c-696f-4fad-b01b-255b77e56b1f
initial_address:      ::ffff:127.0.0.1
initial_port:         60500
interface:            1
os_user:              root
client_hostname:      emr-header-1.cluster-235053
client_name:          ClickHouse
client_revision:      54438
client_version_major: 20
client_version_minor: 8
client_version_patch: 12
http_method:          0
http_user_agent:
quota_key:
revision:             54438
thread_ids:           []
ProfileEvents.Names:  []
ProfileEvents.Values: []
Settings.Names:       ['use_uncompressed_cache','load_balancing','max_memory_usage']
Settings.Values:      ['0','random','10000000000']
```

system.zookeeper

该表可以查到ZooKeeper中的节点信息。

如果未配置ZooKeeper，则该表不存在。 允许从配置中定义的ZooKeeper集群读取数据。 查询必须具有 `path=` 条件，或使用WHERE子句设置了path IN条件，这对应于ZooKeeper中要获取数据的子对象的路径。

查询语句 `SELECT * FROM system.zookeeper WHERE path = '/clickhouse'` ，输出 `/clickhouse` 节点的对所有子路径的数据。如果需要输出所有根节点的数据，请写入路径为 `'/'` 。如果 `path` 中指定的路径不存在，则将提示异常。

查询语句 `SELECT * FROM system.zookeeper WHERE path IN ('/', '/clickhouse')` ，输出 `/` 和 `/clickhouse` 节点上所有子节点的数据。如果 `path` 中指定的路径不存在，则将提示异常。它可以用于一批ZooKeeper路径的查询。

参数	数据类型	描述
name	String	节点的名字。
path	String	节点的路径。
value	String	节点的值。
dataLength	Int32	节点的值长度。
numChildren	Int32	子节点的个数。
czxid	Int64	创建该节点的事务ID。
mzxid	Int64	最后修改该节点的事务ID。
pzxid	Int64	最后删除或者增加子节点的事务ID。
ctime	DateTime	节点的创建时间。
mtime	DateTime	节点的最后修改时间。
version	Int32	节点版本和节点被修改的次数。
cversion	Int32	最后删除或者增加子节点的事务ID。
aversion	Int32	ACL的修改次数。
ephemeralOwner	Int64	针对临时节点，拥有该节点的事务ID。

示例：您可以执行以下命令，查看表信息。

```
SELECT *
FROM system.zookeeper
WHERE path = '/clickhouse/tables/01-08/visits/replicas'
FORMAT Vertical
```

返回信息如下。

```
Row 1:
──────
name:           example01-08-1.yandex.ru
value:
czxid:          932998691229
mzxid:          932998691229
ctime:          2015-03-27 16:49:51
mtime:          2015-03-27 16:49:51
version:        0
cversion:       47
aversion:       0
ephemeralOwner: 0
dataLength:     0
numChildren:    7
pzxid:          987021031383
path:           /clickhouse/tables/01-08/visits/replicas
Row 2:
──────
name:           example01-08-2.yandex.ru
value:
czxid:          933002738135
mzxid:          933002738135
ctime:          2015-03-27 16:57:01
mtime:          2015-03-27 16:57:01
version:        0
cversion:       37
aversion:       0
ephemeralOwner: 0
dataLength:     0
numChildren:    7
pzxid:          987021252247
path:           /clickhouse/tables/01-08/visits/replicas
```

system.replicas

该表包含本地服务所有复制表的信息和状态，可以用于监控。

参数	数据类型	描述
database	String	数据库名称。
table	String	表名。
engine	String	表引擎名称。
is_leader	UInt8	副本是否是领导者。 一次只有一个副本可以成为领导者。 领导者负责选择要执行的后台合并。 注意 可以对任何可用且在Zookeeper中具有会话的副本执行写操作，不管该副本是否为leader。
can_become_leader	UInt8	副本是否可以当选为领导者。
is_readonly	UInt8	副本是否处于只读模式。 存在以下情形时，开启此配置： • 配置中缺省了zookeeper的部分。 • 在zookeeper重新加载会话时发生未知错误。 • 在会话期间重新初始化了zookeeper。
is_session_expired	UInt8	与ZooKeeper的会话已经过期。用法基本上与 `is_readonly` 相同。
future_parts	UInt32	由于尚未完成的插入或合并而显示的数据部分的数量。
parts_to_check	UInt32	队列中用于验证的part的数量。 如果怀疑part可能损坏了，则将其放入验证队列。
zookeeper_path	String	在ZooKeeper中的表数据路径。

参数	数据类型	描述
replica_name	String	ZooKeeper中的副本名称。同一表的不同副本具有不同的名称。
replica_path	String	ZooKeeper中副本数据的路径。 与*zookeeper_path/replicas/replica_path*下的内容相同。
columns_version	Int32	表结构的版本号。 表示执行ALTER的次数。 如果副本有不同的版本，则意味着部分副本还没有进行所有的更改。
queue_size	UInt32	等待执行的操作的队列大小。 操作包括插入数据块、合并和某些其它操作。 它通常与 `future_parts` 一致。
inserts_in_queue	UInt32	需要插入的数据块的数量。 数据的插入通常很快。 如果该数值很大，则说明有问题。
merges_in_queue	UInt32	等待进行合并的数量。 有时合并时间很长，因此此值可能长时间大于零。
part_mutations_in_queue	UInt32	等待进行的突变的数量。
queue_oldest_time	DateTime	如果 `queue_size` 大于0，则显示何时将最早的操作添加到队列。
inserts_oldest_time	DateTime	
merges_oldest_time	DateTime	
part_mutations_oldest_time	DateTime	
log_max_index	UInt64	一般活动日志中的最大条目数。 注意 存在与ZooKeeper的活动会话时才具有非零值。
log_pointer	UInt64	副本复制到其执行队列的常规活动日志中的最大条目数，再加一。 如果log_pointer比log_max_index小，则说明有问题。 注意 存在与ZooKeeper的活动会话时才具有非零值。
last_queue_update	DateTime	上次更新队列的时间。 注意 存在与ZooKeeper的活动会话时才具有非零值。
absolute_delay	UInt64	当前副本最大延迟时间。单位为秒。 注意 存在与ZooKeeper的活动会话时才具有非零值。
total_replicas	UInt8	此表的已知副本总数。
active_replicas	UInt8	在ZooKeeper中具有会话的此表的副本的数量（即正常运行的副本的数量）。

示例：您可以执行以下命令，查看表信息。

```
SELECT * FROM system.replicas WHERE table = 'visits' FORMAT Vertical
```

返回信息如下。

```
Row 1:
──────
database:                   cltest
table:                      flat_xl_customer_local
engine:                     ReplicatedMergeTree
is_leader:                  1
can_become_leader:          1
is_readonly:                0
is_session_expired:         0
future_parts:               0
parts_to_check:             0
zookeeper_path:             /cltest/cluster_emr-1/flat_xl_customer_local
replica_name:               emr-header-1.cluster-235053
replica_path:               /cltest/cluster_emr-1/flat_xl_customer_local/replicas/emr-header-1.cluster-235053
columns_version:            -1
queue_size:                 0
inserts_in_queue:           0
merges_in_queue:            0
part_mutations_in_queue:    0
queue_oldest_time:          1970-01-01 08:00:00
inserts_oldest_time:        1970-01-01 08:00:00
merges_oldest_time:         1970-01-01 08:00:00
part_mutations_oldest_time: 1970-01-01 08:00:00
oldest_part_to_get:
oldest_part_to_merge_to:
oldest_part_to_mutate_to:
log_max_index:              100157
log_pointer:                100158
last_queue_update:          1970-01-01 08:00:00
absolute_delay:             0
total_replicas:             2
active_replicas:            2
zookeeper_exception:
```

system.storage_policies

该表包含了有关存储策略和卷的优先级相关的信息。

参数	数据类型	描述
policy_name	String	存储策略的名称。
volume_name	String	存储策略中定义的卷的名称。
volume_priority	UInt64	配置中定义的卷的优先级。
disks	String	存储策略中定义的磁盘名称。
max_data_part_size	UInt64	可以存储在磁盘卷上的数据part的最大值。
move_factor	Float64	可用磁盘空间的比率。当比率超过配置参数的值时，数据将会被移动到下一个卷。

示例：您可以执行以下命令，查看表信息。

```
SELECT * FROM system.storage_policies
```

返回信息如下。

```
┌policy_name─┬volume_name─┬volume_priority─┬disks─────────────────┬volume_type─┬max_data_part_size┬move_factor┐
│ default      │ default     │              1 │ ['default']                      │ JBOD        │                 0 │
0 │
│ hdd_in_order │ single      │              1 │ ['disk1','disk2','disk3','disk4'] │ JBOD        │                 0 │
0.1 │
└──────────┴─────┴────────┴──────────────┴─────┴────────┴────────┘
```

system.disks

该表包含了配置中定义的磁盘信息。

参数	数据类型	描述
name	String	配置的磁盘名称。
path	String	文件系统中挂载的磁盘路径。
free_space	UInt64	磁盘上的可用空间（Bytes）。
total_space	UInt64	磁盘的总空间（Bytes）。
keep_free_space	UInt64	磁盘上需要保持空闲的空间。定义在磁盘配置的keep_free_space_bytes 参数中。

示例：您可以执行以下命令，查看表信息。

```
SELECT * FROM system.disks;
```

返回信息如下。

```
┌─name─┬─path────────────┬─free_space─┬─total_space─┬─keep_free_space─┬─type──┐
│ default │ /var/lib/clickhouse/   │ 17236594688 │ 84014424064 │               0 │ local │
│ disk1   │ /mnt/disk1/clickhouse/ │ 17226108928 │ 84003938304 │        10485760 │ local │
│ disk2   │ /mnt/disk2/clickhouse/ │ 28623364096 │ 84003938304 │        10485760 │ local │
│ disk3   │ /mnt/disk3/clickhouse/ │ 34770505728 │ 84003938304 │        10485760 │ local │
│ disk4   │ /mnt/disk4/clickhouse/ │ 59107045376 │ 84003938304 │        10485760 │ local │
└──────┴────────────────┴────────────┴─────────────┴─────────────────┴──────┘
```

6.3.3.3. 监控

E-MapReduce（简称EMR）上的ClickHouse集群提供了完善的监控体系，分为服务监控和节点监控两个维度。本文为您介绍如何查看服务监控和节点监控。

背景信息

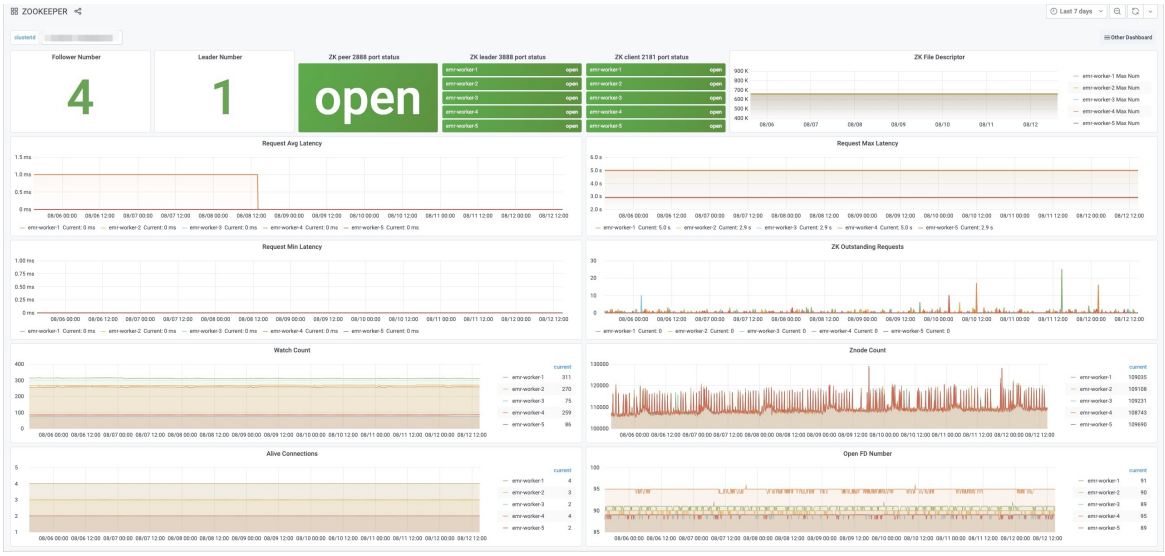
ClickHouse集群服务监控只有ClickHouse和Zookeeper服务。在集群监控大盘的集群指标页面，可以查看不同组件的监控数据，可以根据需求选择时间粒度。

前提条件

已创建ClickHouse集群，详情请参见创建集群。

查看服务监控

1. 进入监控大盘页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**监控大盘**页签。
2. 单击上方的**集群指标**页签。
3. 查看ZOOKEEPER服务和ClickHouse服务的监控数据。
   - 单击**HOST**，选择**ZOOKEEPER**服务。



- 单击**HOST**，选择**CLICKHOUSE**服务。

ClickHouse的监控指标分为3组，分别来自ClickHouse的三个系统表metrics、events和asynchronous_metrics。

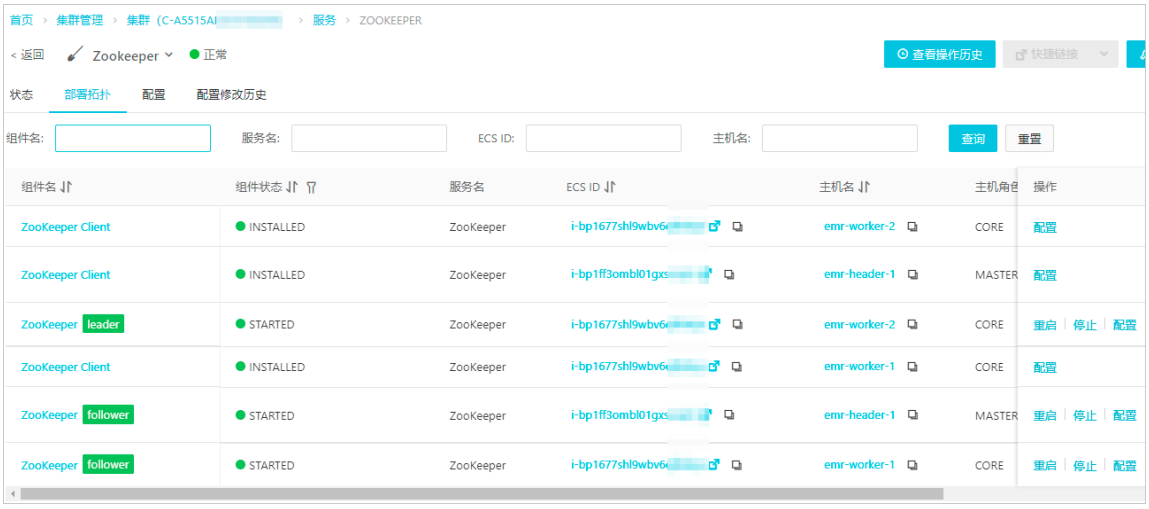
查看节点监控

查看节点监控又分为节点部署状态和查看节点详细监控指标。

1. 进入集群详情页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
2. 查看Zookeeper服务或ClickHouse服务的监控数据。
   i. 在左侧导航栏中，选择**集群服务 > Zookeeper**。
   ClickHouse服务，需要选择**集群服务 > ClickHouse**。

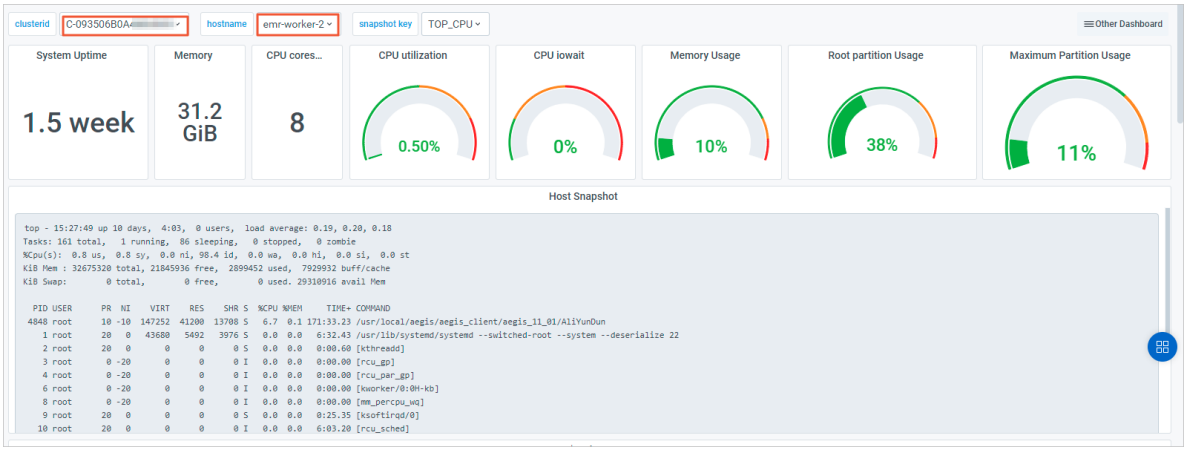
ii. 单击**部署拓扑**页签。

该页面展示了部署服务的进程的实时状态，方便您监控组件的情况。



1. 进入监控大盘页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**监控大盘**页签。
2. 单击上方的**集群指标**页签。
3. 从**clusterid**下拉列表中，选择您创建的集群ID，**hostname**下拉列表中，选择节点名称。

   该页面展示了节点具体的监控指标的情况，包括Load、Disk、内存、网络和TCP等信息。



查看部署状态

查看节点详细监控指标

6.3.3.4. 配置项说明

阿里云E-MapReduce（简称EMR）的ClickHouse集群中，主要提供了四种服务配置项以配置ClickHouse集群，包括客户端配置、服务端配置、用户权限配置和拓展配置。本文为您介绍ClickHouse服务的客户端配置、服务端配置和拓展配置。

背景信息

ClickHouse集群提供的四种服务配置项信息如下表。

配置项	详情
客户端配置	client-config
服务端配置	server-config
拓展配置	server-metrika

配置项	详情
用户权限配置	访问权限控制

前提条件

已创建E-MapReduce的ClickHouse集群，详情请参见创建集群。

注意事项

由于ClickHouse配置采用XML文件形式，所以相对比较灵活，可以进行多层嵌套。自定义配置时规则如下：

- 如果配置可以直接填写在*yandex*标签下，则可以直接新增。
- 如果配置包含多层嵌套，每层需要直接使用半角句号（.）进行连接。

  例如，在**server-users**页签中，添加新的用户aliyun，可以设置参数为**users.aliyun.password**，参数值为密码，您可以自定义。
- 自定义配置中，请勿使用XML类型作为参数或者参数值。

client-config

该服务配置项用于生成clickhouse-client所使用的*config.xml*文件。您可以在EMR控制台ClickHouse服务的**配置**页面，单击**client-config**页签，查看以下参数。

配置项	描述
user	设置*clickhouse-client*使用的用户。默认值为default。
password	设置*clickhouse-client*进程使用的用户密码。默认值为空。
prompt_by_server_display_name.production	在使用*clickhouse-client*的时候，允许自定义提示符。设置这个选项以配置在不同的display_name情况下的提示符，例如在**server-config**选项中设置display_name为default时，所展示的提示符为**prompt_by_server_display_name.default**中所设置的值。设置提示符允许使用的颜色，请参见Color prompts with readline和tip_colors_and_formatting。
prompt_by_server_display_name.default	
prompt_by_server_display_name.test	

server-config

该服务配置项用来生成*clickhouse-server*进程所使用的*config.xml*文件。您可以在EMR控制台ClickHouse服务的**配置**页面，单击**server-config**页签，查看以下参数。

配置项	描述
tcp_port	通过TCP协议与客户端通信的端口。默认值为**9000**。
logger.count	存档的ClickHouse日志文件个数。当存档的日志文件个数达到该参数设置的值时，ClickHouse会将最早的存档删除。默认值为**10**。
logger.errorlog	ClickHouse Server中错误日志的输出路径。默认值为*/var/log/clickhouse-server/clickhouse-server.err.log*。
logger.level	日志的等级，默认等级为information。可以配置的等级从严格到宽松依次为none（关闭日志）、fatal、critical、error、warning、notice、information、debug和trace。
logger.size	日志文件的大小。当文件达到该参数设置的值时，ClickHouse会将其存档并重命名，并创建一个新的日志文件。默认值为**1000M**。
logger.path	ClickHouse Server正常输出的日志文件，默认为*/var/log/clickhouse-server/clickhouse-server.log*，会输出符合*logger.level*所指定的日志等级的日志。
access_control_path	ClickHouse Server用于存储SQL命令创建的用户和角色配置的文件夹的路径。默认值为*/var/lib/clickhouse/access/*。
user_files_path	用户文件的目录。会在表的函数*file()*中被使用。默认值为*/var/lib/clickhouse/user_files/*。
path_to_regions_hierarchy_file	用于ClickHouse内部字典，包含区域层次结构的文件的路径。默认值为空。
path_to_regions_names_files	用于ClickHouse内部字典，包含区域名称的文件的路径。默认值为空。
distributed_ddl.path	该参数指定了ZooKeeper中用于存储DDL查询队列的路径。默认值为*/clickhouse/task_queue/ddl*。默认情况下，ClickHouse的操作CREATE、DROP、ALTER和RENAME等都只会影响正在处理查询的这一台机器。设置distributed_ddl相关参数，允许ClickHouse的查询运行在集群中（当且仅当ZooKeeper被使用时）。

配置项	描述
tmp_policy	用于存放处理大型查询时产生的临时数据。默认值为空。 从server-metrika服务配置项中的storage_configuration选项中设置的磁盘策略选择一个来设置。 说明 如果此项为空，则使用tmp_path，否则tmp_path会被忽略。
path	包含数据的目录的路径，末尾必须加上正斜线（/）。默认值为*/var/lib/clickhouse/*。
https_port	通过HTTPS连接到服务器的端口。指定此参数时，必须配置OpenSSL相关参数。如果指定了http_port，此参数会被忽略。默认值为空。
query_log.flush_interval_milliseconds query_log.engine query_log.partition_by query_log.database query_log.table	如果在使用的profile中设置了*log_queries=1*，则会记录下参与了查询操作的线程信息，这些信息会被记录在表中。query_log系列参数可以来配置此行为，支持的系列参数如下： • flush_interval_milliseconds：用来设置内存中的数据被刷到表中的时间间隔，默认值为7500。 • engine：用来设置表所用的引擎。默认值为空。 注意 设置此项时请勿设置query_log.partition_by，否则可能会引发异常。 • partition_by：用来设置表的分区键。默认值为toYYYYMM(event_date)。 • database：用来设置表所在的库名，默认值为system。 • table：用来设置表名，默认值为*query_thread_log*。
interserver_http_credentials.user interserver_http_credentials.password	如果表使用的引擎是Replicated*类型的，通常情况下，复制是不需要进行身份验证的，但可以通过设置这些参数开启身份验证。这个凭据仅用于副本之间的通信，与ClickHouse客户端的凭据无关。 • user：用户名。默认值为空。 • password：密码。默认值为空。
mlock_executable	当ClickHouse启动后执行mlockall可以降低查询延迟，并防止在高IO负载下调出ClickHouse可执行文件。默认值为false。 说明 建议启用此选项，尽管启用此选项会导致启动时间增加几秒钟。
trace_log.table trace_log.database trace_log.partition_by trace_log.engine trace_log.flush_interval_milliseconds	如果在使用的profile中*query_profiler_real_time_period_ns*和*query_profiler_cpu_time_period_ns*其中任意一个值非0，则会将query profiler记录的stack trace存放到表中。trace_log支持的系列参数如下： • database：用来设置表所在的库名，默认值为system。 • table：用来设置表名，默认值为trace_log。 • partition_by：用来设置表的分区键。默认值为toYYYYMM(event_date)。 • engine：用来设置表所用的引擎。默认值为空。 注意 设置此项时请勿设置trace_log.partition_by，否则可能会引发异常。 • flush_interval_milliseconds：用来设置内存中的数据被刷到表中的时间间隔，默认值为7500。
disable_internal_dns_cache	值非0时禁用内部DNS缓存。默认值为0。 说明 推荐在环境经常变化的系统中使用。例如，Kubernetes。
listen_reuse_port	是否允许Socket间复用相同的Port。取值如下： • 0（默认值）：不允许Socket间复用相同的Port。 • 1：允许Socket间复用相同的Port。

配置项	描述
query_thread_log.table	如果在使用的profile中设置*log_query_threads=1*，则会记录下参与了查询操作的线程信息，这些信息会被记录在表中。 • database：用来设置表所在的库名，默认值为system。 • table：用来设置表名，默认值为query_thread_log。 • partition_by：用来设置表的分区键。默认值为toYYYYMM(event_date)。 • engine：用来设置表所用的引擎。默认值为空。 **注意** 设置此项时请勿设置**query_thread_log.partition_by**，否则可能会引发异常。 • flush_interval_milliseconds：用来设置内存中的数据被刷新到表中的时间间隔，默认值为7500。
query_thread_log.database	
query_thread_log.partition_by	
query_thread_log.engine	
query_thread_log.flush_interval_milliseconds	
default_database	默认数据库。默认值为default。
http_server_default_response	访问ClickHouse的HTTP Server时，默认返回的页面。
display_name	ClickHouse Server端设置的客户端默认提示信息。默认值为空。
builtin_dictionaries_reload_interval	重新加载内置字典的间隔时间，单位为秒。默认值为3600。
umask	文件权限掩码。默认值为027，表示其他用户（操作系统用户）无法读取日志和数据等文件，相同组的用户仅可以读取。
uncompressed_cache_size	表引擎使用MergeTree时，为解压后的block所建立的Cache大小。默认值为0。 如果设置为0，则表示不启用Cache。
timezone	设置服务器的时区。默认值为*Asia/Shanghai*。
max_session_timeout	Session最大超时时间，单位为秒。默认值为3600。
default_session_timeout	Session默认超时时间，单位为秒。默认值为60。
max_open_files	打开文件的最大数量。默认值为262144。 **说明** 该参数与操作系统有关联关系，当此值设置为空时，ClickHouse会使用操作系统设定的*max_open_files*。
tmp_path	用于处理大型查询的临时数据的路径，末尾必须带上正斜杠（/）。默认值为*/var/lib/clickhouse/tmp/*。
max_concurrent_queries	可以同时处理查询的最大数量。默认值为100。
tcp_port_secure	用于与客户端安全通信的TCP端口。默认值为空。 **说明** 配置此参数时，需要设置OpenSSL相关参数。
listen_try	如果通过**listen_host**所指定的协议（IPv4或IPv6）不可用时，是否立刻退出： • 0（默认值）：不立即退出。 • 1：立即退出。
mysql_port	通过MySQL协议与客户端通信的端口。
keep_alive_timeout	ClickHouse在关闭连接之前等待传入请求的秒数，单位为秒。默认值为3。
max_connections	允许连接的最大数量。默认值为4096。
dns_cache_update_period	设置更新存储在ClickHouse内部DNS缓存中的IP地址的周期，单位为秒。默认值为15。 更新会在单独系统线程中异步进行。
path_to_regions_names_files	用于ClickHouse内部字典的，包含区域名称的文件的路径。默认为空。

配置项	描述
include_from	ClickHouse Server中的配置文件是利用XML编写的，其中有一些XML标签中包含了incl属性，这些标签的内容是可以被include_from配置所引用的文件中对应的配置所替换的。默认值为*/etc/ecm/clickhouse-conf/clickhouse-server/metrika.xml*。
interserver_http_port	ClickHouse服务器之间交换数据的端口。默认值为9009。
dictionaries_config	外部字典的配置文件路径。路径可以包含通配符半角句号（.）、星号（*）和半角问号（?）。默认值为**_dictionary.xml*。
http_port	通过HTTP连接到服务器的端口。默认值为8123。 ClickHouse官方JDBC也通过此端口访问ClickHouse，请参见clickhouse-jdbc。
users_config	包含用户配置、访问权限、设置配置文件和资源限制配置等配置的文件路径。默认值为*users.xml*。
dictionaries_lazy_load	延迟加载字典。取值如下： • true（默认值）：会在第一次使用时创建字典，如果字典创建失败，则使用该字典的函数将引发异常。 • false：服务器启动时创建所有字典，如果出现错误，则服务器直接退出。
listen_host	ClickHouse服务器所监听的IP地址。可以指定IPv4和IPv6的地址，如果指定::则代表允许所有地址。可以设置多个IP地址，多个IP地址以逗号（,）分割。例如，*127.0.0.1,localhost*。默认值为*0.0.0.0*。
default_profile	默认会使用的profile。默认值为default。
mark_cache_size	表引擎使用MergeTree时，mark索引所使用的Cache大小的近似值。默认值为5368709120，单位为Byte。
listen_backlog	设置backlog的数量。默认值为64。
format_schema_path	存放输入数据Schema的目录。默认值为*/var/lib/clickhouse/format_schemas/*。

server-metrika

该服务配置项用于生成*metrika.xml*文件，其默认被ClickHouse Server的config所引用。您可以在EMR控制台ClickHouse服务的**配置**页面，在默认的**server-metrika**页签，查看以下参数。

配置项	描述
clickhouse_compression	为使用MergeTree相关引擎的表设置数据压缩，详细信息请参见Server Settings。默认值为空。 如果需要使用ClickHouse的压缩，请自行添加配置。
storage_configuration	用来指定自定义的磁盘信息。阿里云E-MapReduce默认会自动为每块磁盘创建ClickHouse的数据目录，并且为这些磁盘创建一个HDD in order的磁盘策略。
zookeeper_servers	用来配置ClickHouse集群所使用的ZooKeeper信息。默认值为创建ClickHouse集群时同时创建的ZooKeeper的值。多个ZooKeeper节点时，请使用英文逗号（,）进行分隔，例如，`emr-header-1.cluster-12345:2181,emr-worker-1.cluster-12345:2181,emr-worker-2.cluster-12345:2181`。
quotas_default	ClickHouse允许配置不同的quota以灵活的使用不同的资源限制。修改该配置项可以修改名为default的quota设置。如果需要添加新的quota设置，您可以添加自定义设置。
clickhouse_remote_servers	用来自定义集群的分片和副本信息。默认值为创建ClickHouse集群时设置的Shard和Replica数量所生成的拓扑。

例如，创建集群时设置2个Shard和2个Replica，则**clickhouse_remote_servers**值如下所示：

```
<cluster_emr>
  <shard>
    <weight>1</weight>
    <internal_replication>true</internal_replication>
    <replica>
      <host>emr-header-1.cluster-12345</host>
      <port>9000</port>
    </replica>
    <replica>
      <host>emr-worker-1.cluster-12345</host>
      <port>9000</port>
    </replica>
  </shard>
  <shard>
    <weight>1</weight>
    <internal_replication>true</internal_replication>
    <replica>
      <host>emr-worker-2.cluster-12345</host>
      <port>9000</port>
    </replica>
    <replica>
      <host>emr-worker-3.cluster-12345</host>
      <port>9000</port>
    </replica>
  </shard>
</cluster_emr>
```

相关文档

ClickHouse参数的详情信息，可以参见以下官方文档：

- Server Settings
- Settings
- MergeTree tables settings

后续步骤

如果需要修改或添加配置项，请参见管理组件参数。

6.3.3.5. 访问权限控制

阿里云E-MapReduce（简称EMR）的ClickHouse集群中，主要提供了四种服务配置项以配置ClickHouse集群，包括客户端配置、服务端配置、用户权限配置和拓展配置。本文为您介绍ClickHouse服务的用户权限配置。

背景信息

用户访问权限配置在*server-users*和*server-metrika*文件中，包含users、profiles和quotas三部分配置。详细配置信息：

- users配置
- profiles配置
- quotas配置

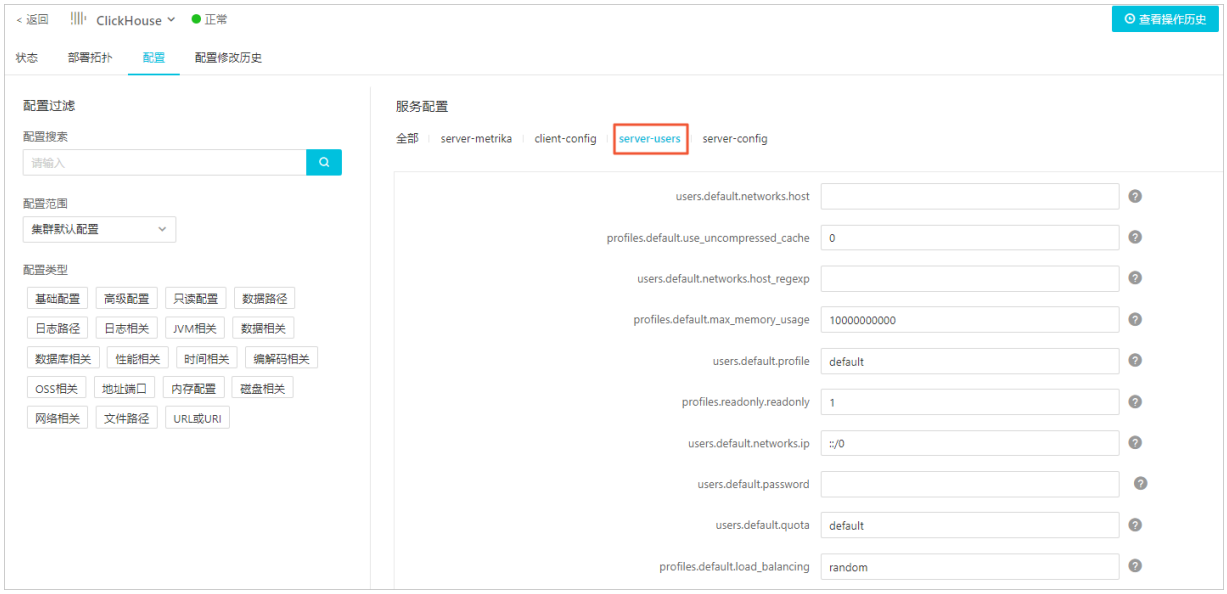
> **说明** ClickHouse服务的客户端配置、服务端配置和拓展配置的详细信息，请参见配置项说明。

前提条件

已创建E-MapReduce的ClickHouse集群，详情请参见创建集群。

users配置

您可以在ClickHouse服务配置页面的服务配置区域，查看或修改配置。users配置在*server-users*页签中。



参数	描述
users.default.networks.host	default用户允许访问的主机名，默认值为空。 多个主机名时，可以使用英文逗号（,）分隔。
users.default.networks.host_regexp	default用户允许访问的主机名的正则表达式，默认值为空。 多个表达式时，可以使用英文逗号（,）分隔。
users.default.networks.ip	default用户允许访问的IP地址。默认值为**::/0**，表示允许所有IP地址访问。 多个IP地址时，可以使用英文逗号（,）分隔。 **注意** 请确保users.default.networks.ip、users.default.networks.host和users.default.networks.host_regexp三个参数中，至少有一个参数值不为空，否则可能会导致网络不通。
users.default.profile	default用户默认使用的profile名称。默认值为**default**。
users.default.password	ClickHouse Server中default用户的密码。默认值为空。 **注意** 不建议使用此配置。建议您添加自定义参数users.default.password_sha256_hex或users.default.password_double_sha1_hex以设置密码。 • users.default.password_sha256_hex：SHA256产生的密钥的16进制字符串。 • users.default.password_double_sha1_hex：通过两次SHA1产生的密钥的16进制字符串。
users.default.quota	default用户默认使用的quota配置。默认值为**default**。

profiles配置

您可以在ClickHouse服务配置页面的服务配置区域，查看或修改配置。profiles配置在*server-users*页签中。

参数	描述
profiles.default.max_memory_usage	用于设置名为default的profile中max_memory_usage的值。修改该参数可以设置单个查询时所能够使用的最大内存。 默认为10,000,000,000，单位为byte。

参数	描述
profiles.default.use_uncompressed_cache	用于设置名为default的profile中use_uncompressed_cache的值。 • 1（默认值）：表示使用未压缩数据块的缓存。 • 0：表示不使用未压缩数据块的缓存。
profiles.default.load_balancing	用于设置名为default的profile中*load_balancing*的值。可以设置在分布式查询处理中选择副本的策略。 策略详细信息，请参见Settings。
profiles.readonly.readonly	用于设置名为readonly的profile中readonly的值。 • 1：使用名为readonly的profile，只允许执行读操作。 • 0：不开启readonly。

quotas配置

您可以在ClickHouse服务配置页面的服务配置区域，查看或修改配置。quotas配置在*server-metrika*页签中。

quotas_default：ClickHouse允许配置不同的quota以灵活的使用不同的资源限制。修改该配置项可以修改名为default的quota设置（**users.default.quota**）。如果需要添加新的quota设置，你可以单击右上角的**自定义配置**，详细操作请参见添加组件参数。

6.3.3.6. 扩容ClickHouse集群

随着业务量的增长，当ClickHouse集群已不能满足业务需求时，需要扩容ClickHouse集群。ClickHouse集群支持分片扩容和副本扩容两种方式，当集群容量不能满足业务需求时，可以进行分片扩容；当集群并发访问量不能满足业务需求时，可以进行副本扩容。本文为您介绍如何扩容ClickHouse集群。

前提条件

已创建EMR-3.38.0及后续版本，EMR-5.4.0及后续版本的ClickHouse集群，详情请参见创建集群。

注意事项

- 扩容的机器数必须是分区或副本的倍数。
- 集群扩容，仅支持表结构迁移，不支持数据迁移。
- default数据库下的表结构不支持迁移。
- 分片扩容是直接在原有的集群上增加节点，并在新增节点上创建分布式表和本地表，扩容后新写入的数据按照原有的分布策略进行写入。

操作步骤

1. 进入集群管理页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
2. 单击目标集群操作列的**更多 > 扩容**。
3. 在**扩容**对话框中，修改以下参数，单击**确定**。

参数		描述
扩容方式	**分片(shard)扩容**	当集群容量不能满足业务需求时，可以进行分片扩容。
	副本(replica)扩容	当集群并发访问量不能满足业务需求时，可以进行副本扩容。
增加数量		扩容的机器数必须是分区或副本的倍数。
E-MapReduce服务条款		阅读并同意服务条款后，选中即可。

4. 查看扩容情况。
   i. 在**集群管理**页面，单击目标集群操作列的**详情**。
   ii. 在左侧导航栏，单击**集群脚本**。
   iii. 单击右上角的**查看操作历史**。

当**操作类型**为**RESIZE**，且**状态**为**成功**时，表示扩容已完成。

6.3.3.7. 扩容磁盘

当EMR的ClickHouse集群数据存储空间不足时，则需要进行扩容操作。ClickHouse磁盘扩容分为云盘扩容和云盘挂载两种，本文主要介绍为您介绍如何挂载云盘。

背景信息

在使用EMR ClickHouse集群时，尽管在创建前已经做好数据规模的估算了，但随着业务的发展，EMR ClickHouse集群的存储可能依旧不够用，无论是临时扩容以应对临时的高峰（例如，应对大促时期的数据量），或是需要长期扩容以应对日益增长的数据，可能都需要对EMR ClickHouse集群的磁盘进行扩容。

如果ClickHouse集群当前所挂载的数据盘为ESSD云盘、SSD云盘或高效云盘等类型的云盘，则可以直接扩容云盘的容量而不需要修改任何配置。扩容详细信息，请参见扩容磁盘。

前提条件

已在EMR控制台创建了ClickHouse集群，详情请参见创建集群。

操作步骤

1. 进入实例的云盘页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在**主机信息**区域，单击目标实例的**ECS ID**。
2. 在ECS控制台创建云盘，详情请参见创建云盘。
   创建云盘时，**是否挂载**选择为**挂载到ECS实例**。按量付费的集群，**释放设置**需要选择为**云盘随实例释放**。
3. 在目标实例的节点，挂载磁盘。
   i. 登录目标实例节点，详情请参见登录集群。
   ii. 执行 `lsblk` 命令，查看未挂载的磁盘。

   ```
   [root@emr-header-1 ~]# lsblk
   NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
   vdf    254:80   0   40G  0 disk
   vdd    254:48   0   80G  0 disk /mnt/disk3
   vdb    254:16   0   80G  0 disk /mnt/disk1
   vde    254:64   0   80G  0 disk /mnt/disk4
   vdc    254:32   0   80G  0 disk /mnt/disk2
   vda    254:0    0  120G  0 disk
   └─vda1 254:1    0  120G  0 part /
   ```

   MOUNTPOINT下显示为空的即是未挂载的磁盘。
   iii. 执行以下命令，格式化磁盘。

   ```
   mkfs.ext4 /dev/vdf
   ```

   > 注意 /dev/vdf为未挂载磁盘的名称，请根据实际情况替换为前一步骤中查找到的未挂载磁盘。

   iv. 执行以下命令，创建挂载目录。

   ```
   mkdir -p /mnt/disk5
   ```

   > 注意 命令中的disk5表示disk的名称，可以是5到n，n取决于您需要挂载多少块盘。本文示例是挂载disk5，请您根据实际情况修改。

   v. 执行以下命令，挂载磁盘。

   ```
   mount -t ext4 /dev/vdf /mnt/disk5
   ```

   > 注意 当前操作仅保证系统未重启可用，重启后磁盘不会自动挂载。

   vi. 将磁盘挂载信息写入到*/etc/fstab*中，使磁盘在操作系统重启后自动挂载。
      a. 执行以下命令，打开*fstab*文件。

      ```
      vim /etc/fstab
      ```

      b. 添加以下内容至文件末尾。

      ```
      /dev/vdf /mnt/disk5 ext4 defaults,noatime,nofail 0 0
      ```

4. 在挂载目录中创建*clickhouse*数据目录。
   i. 执行以下命令，创建*clickhouse*数据目录。

   ```
   mkdir -p /mnt/disk5/clickhouse
   ```

   ii. 执行以下命令，修改目录权限。

   ```
   chown -R clickhouse:hadoop /mnt/disk5/clickhouse/
   ```

5. 在EMR集群中的所有机器上，重复执行步骤1~步骤4。
6. 在EMR控制台，修改ClickHouse配置项。

i. 进入ClickHouse配置页面。
   a. 登录阿里云E-MapReduce控制台。
   b. 单击上方的**集群管理**页签。
   c. 在**集群管理**页面，单击相应集群所在行的**详情**。
   d. 在左侧导航栏中，选择**集群服务 > ClickHouse**。
   e. 在ClickHouse服务页面，单击**配置**页签。
ii. 单击**server-metrika**页签，修改参数**storage_configuration**的参数值。

   具体修改点为以下两处：
   a. 在**disks**中增加扩容的磁盘信息。

```
<disk5>
    <path>/mnt/disk5/clickhouse/</path>
    <keep_free_space_bytes>10485760</keep_free_space_bytes>
  </disk5>
```

   b. 在**single**中增加扩容的磁盘名称。

```
<disk>disk5</disk>
```

   以EMR-3.38.0版本为例，修改后**storage_configuration**的参数值信息如下所示。

```
<disks>
  <disk1>
    <path>/mnt/disk1/clickhouse/</path>
    <keep_free_space_bytes>10485760</keep_free_space_bytes>
  </disk1>
  <disk2>
    <path>/mnt/disk2/clickhouse/</path>
    <keep_free_space_bytes>10485760</keep_free_space_bytes>
  </disk2>
  <disk3>
    <path>/mnt/disk3/clickhouse/</path>
    <keep_free_space_bytes>10485760</keep_free_space_bytes>
  </disk3>
  <disk4>
    <path>/mnt/disk4/clickhouse/</path>
    <keep_free_space_bytes>10485760</keep_free_space_bytes>
  </disk4>
  <disk5>
    <path>/mnt/disk5/clickhouse/</path>
    <keep_free_space_bytes>10485760</keep_free_space_bytes>
  </disk5>
</disks>
<policies>
  <default>
    <volumes>
      <single>
        <disk>disk1</disk>
        <disk>disk2</disk>
        <disk>disk3</disk>
        <disk>disk4</disk>
        <disk>disk5</disk>
      </single>
    </volumes>
  </default>
</policies>
```

iii. 保存配置。
   a. 在ClickHouse服务的**配置**页面，单击**保存**。
   b. 在**确认修改**对话框中，输入执行原因，打开**自动更新配置**开关，单击**确定**。
iv. 部署客户端配置。
   a. 在ClickHouse服务的**配置**页面，单击**部署客户端配置**。
   b. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
   c. 在**确认**对话框中，单击**确定**。

7. 检查配置是否完成。
   i. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。

ii. 执行如下命令，启动ClickHouse客户端。

```
clickhouse-client -m
```

iii. 执行如下命令，查看磁盘信息。

```
select * from system.disks;
```

返回信息如下所示。

```
┌─name────┬─path────────────────────┬─free_space──┬─total_space─┬─keep_free_space─┬─type──┐
│ default │ /var/lib/clickhouse/    │ 83830616064 │ 84014424064 │               0 │ local │
│ disk1   │ /mnt/disk1/clickhouse/ │ 83820130304 │ 84003938304 │        10485760 │ local │
│ disk2   │ /mnt/disk2/clickhouse/ │ 83928371200 │ 84003938304 │        10485760 │ local │
│ disk3   │ /mnt/disk3/clickhouse/ │ 83928371200 │ 84003938304 │        10485760 │ local │
│ disk4   │ /mnt/disk4/clickhouse/ │ 83928371200 │ 84003938304 │        10485760 │ local │
│ disk5   │ /mnt/disk5/clickhouse/ │ 39782125568 │ 41996746752 │        10485760 │ local │
└─────────┴─────────────────────────┴─────────────┴─────────────┴─────────────────┴───────┘
```

iv. 执行如下命令，查看磁盘存储策略。

```
select * from system.storage_policies;
```

返回信息如下所示。

```
┌─policy_name──┬─volume_name─┬─volume_priority─┬─disks─────────────────────────────────┬─volume_type─┬─max_data_part_si
ze─┬─move_factor─┐
│ default      │ default     │               1 │ ['default']                           │ JBOD        │
0 │           0 │
│ hdd_in_order │ single      │               1 │ ['disk1','disk2','disk3','disk4','disk5'] │ JBOD        │
0 │         0.2 │           0 │             0.2 │
└──────────────┴─────────────┴─────────────────┴───────────────────────────────────────┴─────────────┴──────────────────
──┴─────────────┘
```

当回显信息如上文所示时，表示磁盘扩容操作完成。

6.3.3.8. 缩容磁盘

无论是本地盘还是云盘，目前都不支持容量的降低，如果希望进行数据盘的缩容，只能通过卸载数据盘来实现。本文介绍卸载数据盘的流程和具体操作。

前提条件

已在EMR控制台创建了ClickHouse集群，详情请参见创建集群。

使用限制

- 仅支持按量付费类型的集群进行缩容操作，包年包月类型请提交工单处理。
- 本地盘不支持卸载。

注意事项

- 缩容前请暂停向当前ClickHouse Server写入数据。
- 请确保待卸载云盘中的数据皆已移到其他盘中，或云盘中的数据可丢弃。

操作流程

1. 步骤一：从ClickHouse Server中移除数据盘
2. 步骤二：从操作系统中移除数据盘
3. 步骤三：在ECS控制台卸载云盘

步骤一：从ClickHouse Server中移除数据盘

1. 启动ClickHouse客户端。
   i. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。
   ii. 执行如下命令，启动ClickHouse客户端。

   ```
   clickhouse-client -m
   ```

2. 执行如下命令，查看数据盘在ClickHouse中的名称。

```
select * from system.disks;
```

返回信息如下所示。

```
┌─name────┬─path─────────────────┬─free_space──┬─total_space─┬─keep_free_space─┬─type──┐
│ default │ /var/lib/clickhouse/ │ 83830616064 │ 84014424064 │               0 │ local │
│ disk1   │ /mnt/disk1/clickhouse/ │ 83820130304 │ 84003938304 │        10485760 │ local │
│ disk2   │ /mnt/disk2/clickhouse/ │ 83928371200 │ 84003938304 │        10485760 │ local │
│ disk3   │ /mnt/disk3/clickhouse/ │ 83928371200 │ 84003938304 │        10485760 │ local │
│ disk4   │ /mnt/disk4/clickhouse/ │ 83928371200 │ 84003938304 │        10485760 │ local │
│ disk5   │ /mnt/disk5/clickhouse/ │ 39782125568 │ 41996746752 │        10485760 │ local │
└─────────┴──────────────────────┴─────────────┴─────────────┴─────────────────┴───────┘
```

> **说明** path列中包含了name列的值。例如，disk5在ClickHouse中的名称为disk5。本文以移除挂载点在disk5上的数据盘为例。

3. 执行以下命令，查看待移除的数据盘上的part。

```
SELECT name,database,table FROM system.parts where active=1 and disk_name='<disk_name>';
```

本示例命令中的 `<disk_name>` 为disk5，返回信息如下所示。

```
SELECT
    name,
    database,
    table
FROM system.parts
WHERE (active = 1) AND (disk_name = 'disk5')
┌─name──────────┬─database─┬─table─┐
│ all_1_140_3 │ default  │ test  │
└─────────────┴──────────┴───────┘
1 rows in set. Elapsed: 0.002 sec.
```

本示例返回信息中 `all_1_140_3` 为part的名称，该part是属于default数据库中test表的。

4. 当所有数据均移动到其它磁盘上之后，检查其他磁盘上的data part数量。
   i. 执行以下命令，查看磁盘data part数量。

   ```
   SELECT database,table,disk_name,count(1) AS part_number FROM system.parts WHERE database != 'system' GROUP BY datab
   ase,table,disk_name ORDER BY part_number DESC;
   ```

   ii. 执行以下命令，按照part_number进行排序，对part_number数量较大的表执行合并操作。

   ```
   optimize table <database_name>.<table_name> final;
   ```

   等待合并完成并清理过期的data part。本示例命令为 `optimize table default.test final;` ，命令中的 `<database_name>` 为数据库名， `<table_name>` 为表名，请您根据实际情况修改。

5. 执行以下命令，将需移除的数据盘上的part移动到继续使用的磁盘上。

```
ALTER TABLE <database_name>.<table_name> MOVE PART '<part_name>' TO DISK '<disk_name>';
```

本示例命令为 `ALTER TABLE default.test MOVE PART 'all_1_140_3' TO DISK 'disk1';` 。命令中的 `<database_name>` 为数据库名， `<table_name>` 为表名， `<part_name>` 为步骤3获取到的part的名称， `<disk_name>` 为不会卸载的盘，请您根据实际情况修改。

> **说明** 如果有多个data part，请重复执行该命令。

6. 将数据盘从ClickHouse配置中移除。
   i. 在ClickHouse服务的**配置**页面，单击**server-metrika**页签。
   ii. 找到参数**storage_configuration**，删除参数值中待删除磁盘（本示例为disk5）的所有配置。
7. 保存配置。
   i. 在ClickHouse服务的**配置**页面，单击**保存**。
   ii. 在**确认修改**对话框中，输入执行原因，打开**自动更新配置**开关，单击**确定**。
8. 部署客户端配置。
   i. 在ClickHouse服务的**配置**页面，单击**部署客户端配置**。
   ii. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

9. 重启Clickhouse服务。
   i. 在ClickHouse服务的**配置**页面的右上角，选择**操作 > 重启Clickhouse Server**。
   ii. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

步骤二：从操作系统中移除数据盘

1. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。
2. 执行 `df -h` 命令，查看待移除的数据盘的设备信息。

   本示例返回信息如下。

```
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        7.6G     0  7.6G   0% /dev
tmpfs           7.6G  171M  7.5G   3% /dev/shm
tmpfs           7.6G  588K  7.6G   1% /run
tmpfs           7.6G     0  7.6G   0% /sys/fs/cgroup
/dev/vda1       118G   54G   60G  48% /
tmpfs           1.6G     0  1.6G   0% /run/user/0
/dev/vdb         79G  704M   78G   1% /mnt/disk1
/dev/vdc         79G   57M   79G   1% /mnt/disk2
/dev/vdd         79G  1.6G   77G   2% /mnt/disk3
/dev/vdf         40G   49M   38G   1% /mnt/disk5
```

   其中，本示例返回信息中的参数：
   - /dev/vdf：系统中待移除数据盘的设备名，需记录下设备名，待步骤三：在ECS控制台卸载云盘中核对。
   - /mnt/disk5：系统中待移除数据盘的挂载点，下一步骤中会使用。
3. 执行以下命令，卸载磁盘。

```
umount <disk_mounted>
```

   > 注意 命令中的 `<disk_mounted>` 为待卸载磁盘的挂载点，本示例是*/mnt/disk5*，请您根据实际情况修改。

4. （可选）如果在*/etc/fstab*中配置了磁盘的自动挂载，则修改并删除其中涉及到的配置。

步骤三：在ECS控制台卸载云盘

1. 进入实例的云盘页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在**主机信息**区域，单击目标实例的**ECS ID**。
   vi. 在ECS控制台，单击**云盘**页签。
2. 在待卸载云盘所在行，选择**更多 > 卸载**。

   如果无法确定需要卸载的数据盘，可以在ECS控制台的云盘页面，将鼠标悬浮于各个云盘ID之上，即可查看各个云盘的设备名称。

   例如，在系统中的设备名为*/dev/vdf*，则云盘的设备名通常为*/dev/xvdf*。

3. 在弹出的对话框中，单击**确认卸载**。

6.3.4. 数据导入

6.3.4.1. 从Spark导入数据至ClickHouse

本文为您介绍如何将Spark中的数据导入至ClickHouse集群。

前提条件

- 已创建Hadoop集群，详情请参见创建集群。
- 已创建ClickHouse集群，详情请参见创建集群。

背景信息

关于Spark的更多介绍，请参见概述。

代码示例

代码示例如下。

```
package com.company.packageName
import java.util.Properties
import java.util.concurrent.ThreadLocalRandom
import scala.annotation.tailrec
import com.google.common.collect.ImmutableMap
import org.apache.spark.internal.Logging
import org.apache.spark.sql.{SaveMode, SparkSession}
case class Test(id: Int, key1: String, value1: Boolean, key2: Long, value2: Double)
object CKDataImporter extends Logging {
  private var dbName: String = "default"
  private var tableName: String = ""
  private var ckHost: String = ""
  private var ckPort: String = "8123"
  private var user: String = "default"
  private var password: String = ""
  private var local: Boolean = false
  def main(args: Array[String]): Unit = {
    parse(args.toList)
    checkArguments()
    val jdbcUrl = s"jdbc:clickhouse://$ckHost:$ckPort/$dbName"
    logInfo(s"Use jdbc: $jdbcUrl")
    logInfo(s"Use table: $tableName")
    val spark = getSparkSession
    // generate test data
    val rdd = spark.sparkContext.parallelize(1 to 1000).map(i => {
      val rand = ThreadLocalRandom.current()
      val randString = (0 until rand.nextInt(10, 20))
        .map(_ => rand.nextLong())
        .mkString("")
      Test(i, randString, rand.nextBoolean(), rand.nextLong(), rand.nextGaussian())
    })
    val df = spark.createDataFrame(rdd)
    df.write
      .mode(SaveMode.Append)
      .jdbc(jdbcUrl, tableName, getCKJdbcProperties(user, password))
  }
  private def printUsageAndExit(exitCode: Int = 0): Unit = {
    logError("Usage: java -jar /path/to/CKDataImporter.jar [options]")
    logError("  --dbName      设置ClickHouse数据库的名称，默认为default")
    logError("  --tableName   设置ClickHouse库中表的名称")
    logError("  --ckHost      设置ClickHouse地址")
    logError("  --ckPort      设置ClickHouse端口，默认为8123")
    logError("  --user        设置ClickHouse所使用的用户名")
    logError("  --password    设置ClickHouse用户的密码，默认为空")
    logError("  --local       设置此程序使用Spark Local模式运行")
    System.exit(exitCode)
  }
  @tailrec
  private def parse(args: List[String]): Unit = args match {
    case ("--help" | "-h") :: _ =>
      printUsageAndExit()
    case "--dbName" :: value :: tail =>
      dbName = value
      parse(tail)
    case "--tableName" :: value :: tail =>
      tableName = value
      parse(tail)
    case "--ckHost" :: value :: tail =>
      ckHost = value
      parse(tail)
```

```
    case "--ckPort" :: value :: tail =>
      ckPort = value
      parse(tail)
    case "--user" :: value :: tail =>
      user = value
      parse(tail)
    case "--password" :: value :: tail =>
      password = value
      parse(tail)
    case "--local" :: tail =>
      local = true
      parse(tail)
    case Nil =>
    case _ =>
      printUsageAndExit(1)
  }
  private def checkArguments(): Unit = {
    if ("".equals(tableName) || "".equals(ckHost)) {
      printUsageAndExit(2)
    }
  }
  private def getCKJdbcProperties(
      user: String,
      password: String,
      batchSize: String = "1000",
      socketTimeout: String = "300000",
      numPartitions: String = "8",
      rewriteBatchedStatements: String = "true"): Properties = {
    val kvMap = ImmutableMap.builder()
      .put("driver", "ru.yandex.clickhouse.ClickHouseDriver")
      .put("user", user)
      .put("password", password)
      .put("batchsize", batchSize)
      .put("socket_timeout", socketTimeout)
      .put("numPartitions", numPartitions)
      .put("rewriteBatchedStatements", rewriteBatchedStatements)
      .build()
    val properties = new Properties
    properties.putAll(kvMap)
    properties
  }
  private def getSparkSession: SparkSession = {
    val builder = SparkSession.builder()
    if (local) {
      builder.master("local[*]")
    }
    builder.appName("ClickHouse-Data-Importer")
    builder.getOrCreate()
  }
}
```

操作流程

1. 步骤一：创建ClickHouse表
2. 步骤二：编译并打包
3. 步骤三：提交作业

步骤一：创建ClickHouse表

1. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。
2. 执行如下命令，进入ClickHouse客户端。

   ```
   clickhouse-client -m
   ```

3. 创建ClickHouse信息。
   i. 执行如下命令，创建数据库*clickhouse_database_name*。

   ```
   CREATE DATABASE clickhouse_database_name ON CLUSTER cluster_emr;
   ```

   阿里云EMR会为ClickHouse集群自动生成一个名为cluster_emr的集群。数据库名您可以自定义。

ii. 执行如下命令，创建表*clickhouse_table_name_local*。

```
CREATE TABLE clickhouse_database_name.clickhouse_table_name_local ON CLUSTER cluster_emr (
  id            UInt32,
  key1            String,
  value1        UInt8,
  key2            Int64,
  value2        Float64
) ENGINE = ReplicatedMergeTree('/clickhouse/tables/{layer}-{shard}/clickhouse_database_name/clickhouse_table_name_l
ocal', '{replica}')
ORDER BY id;
```

> **说明** 表名您可以自定义，但请确保表名是以*_local*结尾。layer、shard和replica是阿里云EMR为ClickHouse集群自动生成的宏定义，可以直接使用。

iii. 执行如下命令，创建与表*clickhouse_table_name_local*字段定义一致的表*clickhouse_table_name_all*。

> **说明** 表名您可以自定义，但请确保表名是以*_all*结尾。

```
CREATE TABLE clickhouse_database_name.clickhouse_table_name_all ON CLUSTER cluster_emr (
  id                    UInt32,
  key1                  String,
  value1                UInt8,
  key2                  Int64,
  value2                Float64
) ENGINE = Distributed(cluster_emr, clickhouse_database_name, clickhouse_table_name_local, rand());
```

步骤二：编译并打包

1. 下载并解压CKDataImporter示例到本地。
2. 在CMD命令行中，进入到下载文件中*pom.xml*所在的目录下，执行如下命令打包文件。

```
mvn clean package
```

根据您*pom.xml*文件中artifactId的信息，下载文件中的*target*目录下会出现*CKDataImporter-1.0.0.jar*的JAR包。

步骤三：提交作业

1. 使用SSH方式登录Hadoop集群，详情请参见登录集群。
2. 上传打包好的*CKDataImporter-1.0.0.jar*至Hadoop集群的根目录下。

> **说明** 本文示例中*CKDataImporter-1.0.0.jar*是上传至root根目录下，您也可以自定义上传路径。

3. 执行如下命令提交作业。

```
spark-submit --master yarn \
             --class com.aliyun.emr.CKDataImporter \
              CKDataImporter-1.0.0.jar \
             --dbName clickhouse_database_name \
             --tableName clickhouse_table_name_all \
             --ckHost ${clickhouse_host};
```

参数	说明
dbName	ClickHouse集群数据库的名称，默认为default。本文示例为*clickhouse_database_name*。
tableName	ClickHouse集群数据库中表的名称。本文示例为*clickhouse_table_name_all*。
ckHost	ClickHouse集群的Master节点的内网IP地址或公网IP地址。IP地址获取方式，请参见获取主节点的IP地址。

获取主节点的IP地址

1. 进入详情页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。

iv. 在**集群管理**页面，单击相应集群所在行的**详情**。

2. 在**集群基础信息**页面的**主机信息**区域，获取主节点的内网或公网IP地址。

6.3.4.2. 从Flink导入数据至ClickHouse

本文为您介绍如何将Flink中的数据导入至ClickHouse集群。

前提条件

- 已创建Flink集群，详情请参见创建集群。
- 已创建ClickHouse集群，详情请参见创建集群。

背景信息

关于Flink的更多介绍，请参见Apache Flink。

代码示例

代码示例如下：

- 流处理

```
package com.company.packageName
import java.util.concurrent.ThreadLocalRandom
import scala.annotation.tailrec
import org.apache.flink.api.common.typeinfo.Types
import org.apache.flink.api.java.io.jdbc.JDBCAppendTableSink
import org.apache.flink.streaming.api.scala._
import org.apache.flink.table.api.scala.{StreamTableEnvironment, table2RowDataStream}
object StreamingJob {
  case class Test(id: Int, key1: String, value1: Boolean, key2: Long, value2: Double)
  private var dbName: String = "default"
  private var tableName: String = ""
  private var ckHost: String = ""
  private var ckPort: String = "8123"
  private var user: String = "default"
  private var password: String = ""
  def main(args: Array[String]) {
    parse(args.toList)
    checkArguments()
    // set up the streaming execution environment
    val env = StreamExecutionEnvironment.getExecutionEnvironment
    val tableEnv = StreamTableEnvironment.create(env)
    val insertIntoCkSql =
      s"""
        | INSERT INTO $tableName (
        |   id, key1, value1, key2, value2
        | ) VALUES (
        |   ?, ?, ?, ?, ?
        | )
        |""".stripMargin
    val jdbcUrl = s"jdbc:clickhouse://$ckHost:$ckPort/$dbName"
    println(s"jdbc url: $jdbcUrl")
    println(s"insert sql: $insertIntoCkSql")
    val sink = JDBCAppendTableSink
      .builder()
      .setDrivername("ru.yandex.clickhouse.ClickHouseDriver")
      .setDBUrl(jdbcUrl)
      .setUsername(user)
      .setPassword(password)
      .setQuery(insertIntoCkSql)
      .setBatchSize(1000)
      .setParameterTypes(Types.INT, Types.STRING, Types.BOOLEAN, Types.LONG, Types.DOUBLE)
      .build()
    val data: DataStream[Test] = env.fromCollection(1 to 1000).map(i => {
```

```
    val data: DataStream[Test] = env.fromCollection(1 to 1000).map(i => {
      val rand = ThreadLocalRandom.current()
      val randString = (0 until rand.nextInt(10, 20))
        .map(_ => rand.nextLong())
        .mkString("")
      Test(i, randString, rand.nextBoolean(), rand.nextLong(), rand.nextGaussian())
    })
    val table = table2RowDataStream(tableEnv.fromDataStream(data))
    sink.emitDataStream(table.javaStream)
    // execute program
    env.execute("Flink Streaming Scala API Skeleton")
  }
  private def printUsageAndExit(exitCode: Int = 0): Unit = {
    println("Usage: flink run com.company.packageName.StreamingJob /path/to/flink-clickhouse-demo-1.0.0.jar [options]")
    println("  --dbName      设置ClickHouse数据库的名称，默认为default")
    println("  --tableName   设置ClickHouse库中表的名称")
    println("  --ckHost      设置ClickHouse地址")
    println("  --ckPort      设置ClickHouse端口，默认为8123")
    println("  --user        设置ClickHouse所使用的用户名")
    println("  --password    设置ClickHouse用户的密码，默认为空")
    System.exit(exitCode)
  }
  @tailrec
  private def parse(args: List[String]): Unit = args match {
    case ("--help" | "-h") :: _ =>
      printUsageAndExit()
    case "--dbName" :: value :: tail =>
      dbName = value
      parse(tail)
    case "--tableName" :: value :: tail =>
      tableName = value
      parse(tail)
    case "--ckHost" :: value :: tail =>
      ckHost = value
      parse(tail)
    case "--ckPort" :: value :: tail =>
      ckPort = value
      parse(tail)
    case "--user" :: value :: tail =>
      user = value
      parse(tail)
    case "--password" :: value :: tail =>
      password = value
      parse(tail)
    case Nil =>
    case _ =>
      printUsageAndExit(1)
  }
  private def checkArguments(): Unit = {
    if ("".equals(tableName) || "".equals(ckHost)) {
      printUsageAndExit(2)
    }
  }
}
```

- 批处理

```
package com.company.packageName
import java.util.concurrent.ThreadLocalRandom
import scala.annotation.tailrec
import org.apache.flink.Utils
import org.apache.flink.api.common.typeinfo.Types
import org.apache.flink.api.java.io.jdbc.JDBCAppendTableSink
import org.apache.flink.api.scala._
import org.apache.flink.table.api.scala.{BatchTableEnvironment, table2RowDataSet}
object BatchJob {
  case class Test(id: Int, key1: String, value1: Boolean, key2: Long, value2: Double)
  private var dbName: String = "default"
  private var tableName: String = ""
  private var ckHost: String = ""
  private var ckPort: String = "8123"
  private var user: String = "default"
  private var password: String = ""
```

```
def main(args: Array[String]) {
  parse(args.toList)
  checkArguments()
  // set up the batch execution environment
  val env = ExecutionEnvironment.getExecutionEnvironment
  val tableEnv = BatchTableEnvironment.create(env)
  val insertIntoCkSql =
    s"""
      | INSERT INTO $tableName (
      |   id, key1, value1, key2, value2
      | ) VALUES (
      |   ?, ?, ?, ?, ?
      | )
      |""".stripMargin
  val jdbcUrl = s"jdbc:clickhouse://$ckHost:$ckPort/$dbName"
  println(s"jdbc url: $jdbcUrl")
  println(s"insert sql: $insertIntoCkSql")
  val sink = JDBCAppendTableSink
    .builder()
    .setDrivername("ru.yandex.clickhouse.ClickHouseDriver")
    .setDBUrl(jdbcUrl)
    .setUsername(user)
    .setPassword(password)
    .setQuery(insertIntoCkSql)
    .setBatchSize(1000)
    .setParameterTypes(Types.INT, Types.STRING, Types.BOOLEAN, Types.LONG, Types.DOUBLE)
    .build()
  val data = env.fromCollection(1 to 1000).map(i => {
    val rand = ThreadLocalRandom.current()
    val randString = (0 until rand.nextInt(10, 20))
      .map(_ => rand.nextLong())
      .mkString("")
    Test(i, randString, rand.nextBoolean(), rand.nextLong(), rand.nextGaussian())
  })
  val table = table2RowDataSet(tableEnv.fromDataSet(data))
  sink.emitDataSet(Utils.convertScalaDatasetToJavaDataset(table))
  // execute program
  env.execute("Flink Batch Scala API Skeleton")
}
private def printUsageAndExit(exitCode: Int = 0): Unit = {
  println("Usage: flink run com.company.packageName.StreamingJob /path/to/flink-clickhouse-demo-1.0.0.jar [options]")
  println("  --dbName      设置ClickHouse数据库的名称，默认为default")
  println("  --tableName   设置ClickHouse库中表的名称")
  println("  --ckHost      设置ClickHouse地址")
  println("  --ckPort      设置ClickHouse端口，默认为8123")
  println("  --user        设置ClickHouse所使用的用户名")
  println("  --password    设置ClickHouse用户的密码，默认为空")
  System.exit(exitCode)
}
@tailrec
private def parse(args: List[String]): Unit = args match {
  case ("--help" | "-h") :: _ =>
    printUsageAndExit()
  case "--dbName" :: value :: tail =>
    dbName = value
    parse(tail)
  case "--tableName" :: value :: tail =>
    tableName = value
    parse(tail)
  case "--ckHost" :: value :: tail =>
    ckHost = value
    parse(tail)
  case "--ckPort" :: value :: tail =>
    ckPort = value
    parse(tail)
  case "--user" :: value :: tail =>
    user = value
    parse(tail)
  case "--password" :: value :: tail =>
    password = value
    parse(tail)
  case Nil =>
  case _ =>
```

```
    case _ =>
      printUsageAndExit(1)
  }
  private def checkArguments(): Unit = {
    if ("".equals(tableName) || "".equals(ckHost)) {
      printUsageAndExit(2)
    }
  }
}
```

操作流程

1. 步骤一：创建ClickHouse表
2. 步骤二：编译并打包
3. 步骤三：提交作业

步骤一：创建ClickHouse表

1. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。
2. 执行如下命令，进入ClickHouse客户端。

   ```
   clickhouse-client -m
   ```

3. 创建ClickHouse信息。
   i. 执行如下命令，创建数据库*clickhouse_database_name*。

   ```
   CREATE DATABASE clickhouse_database_name ON CLUSTER cluster_emr;
   ```

   阿里云EMR会为ClickHouse集群自动生成一个名为cluster_emr的集群。数据库名您可以自定义。

   ii. 执行如下命令，创建表*clickhouse_table_name_local*。

   ```
   CREATE TABLE clickhouse_database_name.clickhouse_table_name_local ON CLUSTER cluster_emr (
     id            UInt32,
     key1            String,
     value1        UInt8,
     key2            Int64,
     value2        Float64
   ) ENGINE = ReplicatedMergeTree('/clickhouse/tables/{layer}-{shard}/clickhouse_database_name/clickhouse_table_name_local', '{replica}')
   ORDER BY id;
   ```

   > **说明** 表名您可以自定义，但请确保表名是以*_local*结尾。layer、shard和replica是阿里云EMR为ClickHouse集群自动生成的宏定义，可以直接使用。

   iii. 执行如下命令，创建与表*clickhouse_table_name_local*字段定义一致的表*clickhouse_table_name_all*。

   > **说明** 表名您可以自定义，但请确保表名是以*_all*结尾。

   ```
   CREATE TABLE clickhouse_database_name.clickhouse_table_name_all ON CLUSTER cluster_emr (
     id                    UInt32,
     key1                  String,
     value1                UInt8,
     key2                  Int64,
     value2                Float64
   ) ENGINE = Distributed(cluster_emr, clickhouse_database_name, clickhouse_table_name_local, rand());
   ```

步骤二：编译并打包

1. 下载并解压flink-clickhouse-demo.tgz示例到本地。
2. 在CMD命令行中，进入到下载文件中*pom.xml*所在的目录下，执行如下命令打包文件。

   ```
   mvn clean package
   ```

   根据您*pom.xml*文件中artifactId的信息，下载文件中的*target*目录下会出现*flink-clickhouse-demo-1.0.0.jar*的JAR包。

步骤三：提交作业

1. 使用SSH方式登录Flink集群，详情请参见登录集群。
2. 上传打包好的*flink-clickhouse-demo-1.0.0.jar*至Flink集群的根目录下。

> **说明** 本文示例中*flink-clickhouse-demo-1.0.0.jar*是上传至root根目录下，您也可以自定义上传路径。

3. 执行如下命令提交作业。

   代码示例如下：

   - 流作业

   ```
   flink run -m yarn-cluster \
             -c com.aliyun.emr.StreamingJob \
             flink-clickhouse-demo-1.0.0.jar \
             --dbName clickhouse_database_name \
             --tableName clickhouse_table_name_all \
             --ckHost ${clickhouse_host};
   ```

   - 批作业

   ```
   flink run -m yarn-cluster \
             -c com.aliyun.emr.BatchJob \
             flink-clickhouse-demo-1.0.0.jar \
             --dbName clickhouse_database_name \
             --tableName clickhouse_table_name_all \
             --ckHost ${clickhouse_host};
   ```

参数	说明
dbName	ClickHouse集群数据库的名称，默认为default。本文示例为*clickhouse_database_name*。
tableName	ClickHouse集群数据库中表的名称。本文示例为*clickhouse_table_name_all*。
ckHost	ClickHouse集群的Master节点的内网IP地址或公网IP地址。ip地址获取方式，请参见获取主节点的IP地址。

获取主节点的IP地址

1. 进入详情页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
2. 在**集群基础信息**页面的**主机信息**区域，获取主节点的内网或公网IP地址。

6.3.4.3. 从HDFS导入数据至ClickHouse

您可以通过HDFS表引擎或表函数读写数据。本文为您介绍如何将HDFS中的数据导入至ClickHouse集群。

前提条件

- 已创建HDFS集群，详情请参见创建集群。
- 已创建ClickHouse集群，详情请参见创建集群。

注意事项

本文代码示例中HDFS URL中的9000为非HA模式下NameNode的端口，如果使用的是HA模式下的NameNode，则端口通常为8020。

使用HDFS表引擎读写数据

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name
(
  name1 [type1],
  name2 [type2],
  ...
)
Engine = HDFS(uri, format);
```

其中，涉及参数描述如下表所示。

参数	描述
db	数据库名。
table_name	表名。
name1/name2	列名。
tyep1/type2	列的类型。
uri	HDFS上文件的地址。 **说明** • 不可以是目录地址。 • 文件所属的目录需要存在，如果不存在，则写数据时会报错。
format	文件的类型。

1. 创建业务表和HDFS表
   i. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。
   ii. 执行以下命令，进入ClickHouse客户端。

   ```
   clickhouse-client -m
   ```

   iii. 执行以下命令，创建数据库hdfs。

   ```
   CREATE DATABASE IF NOT EXISTS hdfs ON CLUSTER cluster_emr;
   ```

   iv. 执行以下命令，创建表orders。

   ```
   CREATE TABLE IF NOT EXISTS hdfs.orders ON CLUSTER cluster_emr
   (
       `uid` UInt32,
       `date` DateTime,
       `skuId` UInt32,
       `order_revenue` UInt32
   )
   ENGINE = HDFS('hdfs://192.168.**.**:9000/orders.csv', 'CSV');
   ```

   说明 本文示例是将示例数据上传到了HDFS集群的根目录下。代码中的 `192.168.**.**` 为HDFS集群的emr-header-1节点的内网IP地址，您可以在EMR控制台**集群管理**页签中的**主机列表**页面查看。

   v. 执行以下命令，创建数据库product。

   ```
   CREATE DATABASE IF NOT EXISTS product ON CLUSTER cluster_emr;
   ```

vi. 执行以下命令，创建业务表orders。

```
CREATE TABLE IF NOT EXISTS product.orders ON CLUSTER cluster_emr
(
    `uid` UInt32,
    `date` DateTime,
    `skuId` UInt32,
    `order_revenue` UInt32
)
Engine = ReplicatedMergeTree('/cluster_emr/product/orders/{shard}', '{replica}')
PARTITION BY toYYYYMMDD(date)
ORDER BY toYYYYMMDD(date);
```

> **说明** 示例中的{shard}和{replica}是阿里云EMR为ClickHouse集群自动生成的宏定义，可以直接使用。

vii. 执行以下命令，创建业务表orders_all。

```
CREATE TABLE IF NOT EXISTS product.orders_all ON CLUSTER cluster_emr
(
    `uid` UInt32,
    `date` DateTime,
    `skuId` UInt32,
    `order_revenue` UInt32
)
Engine = Distributed(cluster_emr, product, orders, rand());
```

2. 使用HDFS表引擎导入数据

i. 下载并上传示例数据orders.csv至HDFS集群的目录下。

> **说明** 本文示例上传到了HDFS集群的根目录下。

ii. 执行以下命令，导入数据。

```
INSERT INTO product.orders_all
SELECT
  uid,
  date,
  skuId,
  order_revenue
FROM
  hdfs.orders;
```

iii. 执行以下命令，检查数据一致性。

```
SELECT
  a.*
FROM
  hdfs.orders a
LEFT ANTI JOIN
  product.orders_all
USING uid;
```

3. 使用HDFS表引擎导出数据

i. 执行以下命令，构造数据。

```
INSERT INTO product.orders_all VALUES \
  (60333391,'2021-08-04 11:26:01',49358700,89) \
  (38826285,'2021-08-03 10:47:29',25166907,27) \
  (10793515,'2021-07-31 02:10:31',95584454,68) \
  (70246093,'2021-08-01 00:00:08',82355887,97) \
  (70149691,'2021-08-02 12:35:45',68748652,1)  \
  (87307646,'2021-08-03 19:45:23',16898681,71) \
  (61694574,'2021-08-04 23:23:32',79494853,35) \
  (61337789,'2021-08-02 07:10:42',23792355,55) \
  (66879038,'2021-08-01 16:13:19',95820038,89);
```

ii. 执行以下命令，导出数据。

```
INSERT INTO hdfs.orders
SELECT
  uid,
  date,
  skuId,
  order_revenue
FROM
 product.orders_all;
```

iii. 执行以下命令，可以检查数据一致性。

```
SELECT
  a.*
FROM
  hdfs.orders
RIGHT ANTI JOIN
  product.orders_all a
USING uid;
```

语法

示例

使用HDFS表函数读写数据

```
hdfs(uri, format, structure);
```

其中，涉及参数描述如下表所示。

参数	描述
`uri`	HDFS上文件的地址。 **说明** • 不可以是目录地址。 • 文件所属的目录需要存在，如果不存在，则写数据时会报错。
`format`	文件的类型。
`structure`	表中字段的类型。例如，column1 UInt32，column2 String。

1. 创建数据库和业务表

i. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。

ii. 执行以下命令，进入ClickHouse客户端。

```
clickhouse-client -m
```

iii. 执行以下命令，创建数据库product。

```
CREATE DATABASE IF NOT EXISTS product ON CLUSTER cluster_emr;
```

iv. 执行以下命令，创建业务表orders。

```
CREATE TABLE IF NOT EXISTS product.orders ON CLUSTER cluster_emr
(
    `uid` UInt32,
    `date` DateTime,
    `skuId` UInt32,
    `order_revenue` UInt32
)
Engine = ReplicatedMergeTree('/cluster_emr/product/orders/{shard}', '{replica}')
PARTITION BY toYYYYMMDD(date)
ORDER BY toYYYYMMDD(date);
```

v. 执行以下命令，创建业务表orders_all。

```
CREATE TABLE IF NOT EXISTS product.orders_all ON CLUSTER cluster_emr
(
    `uid` UInt32,
    `date` DateTime,
    `skuId` UInt32,
    `order_revenue` UInt32
)
Engine = Distributed(cluster_emr, product, orders, rand());
```

2. 使用HDFS表函数导入数据

i. 下载并上传示例数据orders.csv至HDFS集群的目录下。

> **说明** 本文示例上传到了HDFS集群的根目录下。

ii. 执行以下命令，导入数据。

```
INSERT INTO product.orders_all
SELECT
  uid,
  date,
  skuId,
  order_revenue
FROM
  hdfs('hdfs://192.168.**.**:9000/orders.csv', 'CSV', 'uid UInt32, date DateTime, skuId UInt32, order_revenue UInt3
2');
```

iii. 执行以下命令，可以检查数据一致性。

```
SELECT
  a.*
FROM
  hdfs('hdfs://192.168.**.**:9000/orders.csv', 'CSV', 'uid UInt32, date DateTime, skuId UInt32, order_revenue UInt3
2') a
LEFT ANTI JOIN
  product.orders_all
USING uid;
```

3. 使用HDFS表函数导出数据

i. 执行以下命令，构造数据。

```
INSERT INTO product.orders_all VALUES \
  (60333391,'2021-08-04 11:26:01',49358700,89) \
  (38826285,'2021-08-03 10:47:29',25166907,27) \
  (10793515,'2021-07-31 02:10:31',95584454,68) \
  (70246093,'2021-08-01 00:00:08',82355887,97) \
  (70149691,'2021-08-02 12:35:45',68748652,1)  \
  (87307646,'2021-08-03 19:45:23',16898681,71) \
  (61694574,'2021-08-04 23:23:32',79494853,35) \
  (61337789,'2021-08-02 07:10:42',23792355,55) \
  (66879038,'2021-08-01 16:13:19',95820038,89);
```

ii. 执行以下命令，导出数据。

```
INSERT INTO FUNCTION
  hdfs('hdfs://192.168.**.**:9000/orders.csv', 'CSV', 'uid UInt32, date DateTime, skuId UInt32, order_revenue UInt3
2')
SELECT
  uid,
  date,
  skuId,
  order_revenue
FROM
  product.orders_all;
```

iii. 执行以下命令，可以检查数据一致性。

```
SELECT
  a.*
FROM
  hdfs('hdfs://192.168.**.**:9000/orders.csv', 'CSV', 'uid UInt32, date DateTime, skuId UInt32, order_revenue UInt32')
RIGHT ANTI JOIN
  product.orders_all a;
```

语法

示例

配置

EMR ClickHouse允许使用对HDFS进行配置：

- 全局生效的HDFS配置。

```
<hdfs>
  <dfs_default_replica>3</dfs_default_replica>
</hdfs>
```

HDFS参数的详细信息，请参见官网文档HDFS Configuration Reference。

> **说明** 查询参数时将下划线（_）替换为半角句号（.）即可。例如，您要查询EMR中的参数 `dfs_default_replica`，则可以在官网文档中搜索 `dfs.default.replica`。

- 仅对${user}用户生效的HDFS配置，用户配置与全局配置相同的键不同值时，会覆盖全局配置。

```
<hdfs_${user}>
  <dfs_default_replica>3</dfs_default_replica>
</hdfs_${user}>
```

6.3.4.4. 从OSS导入数据至ClickHouse

在EMR ClickHouse集群，您可以在通过OSS表引擎读写数据，或者通过OSS表函数读数据。本文为您介绍如何将OSS中的数据导入至ClickHouse集群。

前提条件

- 已在OSS上创建存储空间，详情请参见创建存储空间。
- 已创建ClickHouse集群，详情请参见创建集群。

使用限制

EMR-3.38.0及后续版本，EMR-5.4.0及后续版本的ClickHouse集群，支持从OSS导入数据至ClickHouse集群。

使用OSS表引擎读写数据

创建OSS表的语法如下所示。

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
  name1 [type1] [NULL|NOT NULL] [DEFAULT|MATERIALIZED|ALIAS expr1] [compression_codec] [TTL expr1],
  name2 [type2] [NULL|NOT NULL] [DEFAULT|MATERIALIZED|ALIAS expr2] [compression_codec] [TTL expr2],
  ...
)
ENGINE = OSS(path, [access_key_id, access_key_secret,] format, [compression]);
```

其中，涉及参数描述如下表所示。

参数	描述
db	数据库名。
table_name	表名。
name1/name2	列名。
tyep1/type2	列的类型。

参数	描述
path	OSS路径。 ClickHouse集群访问OSS使用地址详情，请参见ECS实例通过OSS内网地址访问OSS资源。 path支持virtual hosted style和path style两种形式。推荐您使用virtual hosted style。 path支持使用以下通配符： • `*` 表示除了 '/' 以外的任意字符，包括空字符串。 • `?` 表示单个字符。 • `{str1,str2,...,strn}` 表示 `str1/str2/.../strn` 中任意一个字符串。 • `{N..M}` 表示从N到M的任意数字。N和M可以包含前导0，例如 {001..099}。 **注意** 根据通配符来决定哪些文件会被使用在 SELECT 时而非创建表时，因此如果是为了写入数据至OSS，不应该使用通配符。
access_key_id	阿里云账号的AccessKey ID。
access_key_secret	阿里云账号的AccessKey Secret。
format	path所指向的对象（文件）的格式。例如，CSV和XML等类型，详细信息请参见Formats for Input and Output Data。
compression	压缩类型。 该参数为可选参数，默认会根据文件扩展选择合适的压缩类型。 根据您创建的集群版本，设置压缩类型： • EMR-3.x系列版本：支持 `none` 、 `gzip/gz` 、 `brotli/b` 、 `deflate` 或 `auto` 。 • EMR-5.x系列版本：支持 `none` 、 `gzip/gz` 、 `brotli/b` 、 `lzma(xz)` 、 `auto` 或 `zstd/zst` 。

1. 下载并上传示例数据orders.csv至OSS。

   上传文件至OSS的详细操作，请参见上传文件。
2. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。
3. 执行以下命令，进入ClickHouse客户端。

   ```
   clickhouse-client -m
   ```
4. 执行以下命令，创建数据库test。

   ```
   CREATE DATABASE IF NOT EXISTS test ON CLUSTER cluster_emr;
   ```
5. 执行以下命令，创建业务表。

   i. 创建复制表。

   ```
   CREATE TABLE test.orders ON CLUSTER cluster_emr
   (
     uid UInt32,
     date DateTime,
     skuId UInt32,
     order_revenue UInt32
   ) ENGINE = ReplicatedMergeTree('/clickhouse/cluster_emr/test/orders/{shard}', '{replica}')
   PARTITION BY toYYYYMMDD(date)
   ORDER BY toYYYYMMDD(date);
   ```

   ii. 创建分布式表。

   ```
   CREATE TABLE test.orders_all ON CLUSTER cluster_emr
   (
     uid UInt32,
     date DateTime,
     skuId UInt32,
     order_revenue UInt32
   ) ENGINE = Distributed(cluster_emr, test, orders, rand());
   ```
6. 执行以下命令，创建表orders_oss。

```
CREATE TABLE test.orders_oss
(
  uid UInt32,
  date DateTime,
  skuId UInt32,
  order_revenue UInt32
) ENGINE = OSS('http://test.oss-cn-beijing.aliyuncs.com/orders.csv', '<access_key_id>', '<access_key_secret>', 'CSV');
```

> **说明** 示例中的数据目录*http://test.oss-cn-beijing.aliyuncs.com/orders.csv*，表示cn-beijing地域下名称为test的Bucket中的*orders.csv*文件。

7. 将OSS上的数据写入业务表orders_all。

```
INSERT INTO test.orders_all
SELECT
  uid,
  date,
  skuId,
  order_revenue
FROM
  test.orders_oss;
```

您可以通过以下命令检查数据一致性：

- 查看表orders_all的数据。

```
SELECT count(1) FROM test.orders_all;
```

- 查看表orders_oss的数据。

```
SELECT count(1) FROM test.orders_oss;
```

1. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。
2. 执行以下命令，进入ClickHouse客户端。

```
clickhouse-client -m
```

3. 执行以下命令，创建要写入数据的OSS表。

```
CREATE TABLE test.orders_oss
(
  uid UInt32,
  date DateTime,
  skuId UInt32,
  order_revenue UInt32
) ENGINE = OSS('http://test.oss-cn-beijing.aliyuncs.com/orders.csv', '<access_key_id>', '<access_key_secret>', 'CSV');
```

4. 执行以下命令，向表中写入数据。

```
--假设业务表为test.orders_all
INSERT INTO test.orders_oss
SELECT
  uid,
  date,
  skuId,
  order_revenue
FROM
  test.orders_all;
```

5. 在 OSS管理控制台上查看数据。

语法

示例：使用OSS表将数据导入至ClickHouse集群

示例：将ClickHouse集群数据导出至OSS

使用OSS表函数读数据

创建OSS表的语法如下所示。

```
oss(path, [access_key_id, access_key_secret,] format, structure, [compression])
```

其中，涉及参数描述如下表所示。

参数	描述
path	OSS路径。 ClickHouse集群访问OSS使用地址详情，请参见ECS实例通过OSS内网地址访问OSS资源。 path支持virtual hosted style和path style两种形式。推荐您使用virtual hosted style。 path支持使用以下通配符： • * 表示除了 '/' 以外的任意字符，包括空字符串。 • ? 表示单个字符。 • {str1,str2,...,strn} 表示 str1/str2/.../strn 中任意一个字符串。 • {N..M} 表示从N到M的任意数字。N和M可以包含前导0，例如 {001..099}。 注意 根据通配符来决定哪些文件会被使用在 SELECT 时而非创建表时，因此如果是为了写入数据至OSS，不应该使用通配符。
access_key_id	阿里云账号的AccessKey ID。
access_key_secret	阿里云账号的AccessKey Secret。
format	path所指向的对象（文件）的格式。例如，CSV和XML等类型，详细信息请参见Formats for Input and Output Data。
structure	表中字段的类型。例如，column1 UInt32、column2 String。
compression	压缩类型。 该参数为可选参数，默认会根据文件扩展选择合适的压缩类型。 根据您创建的集群版本，设置压缩类型： • EMR-3.x系列版本：支持 none 、 gzip/gz 、 brotli/b 、 deflate 或 auto 。 • EMR-5.x系列版本：支持 none 、 gzip/gz 、 brotli/b 、 lzma(xz) 、 auto 或 zstd/zst 。

1. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。
2. 执行以下命令，进入ClickHouse客户端。

```
clickhouse-client -m
```

3. 执行以下命令，查询OSS表数据。

```
SELECT * FROM oss(
  'http://test.oss-cn-beijing.aliyuncs.com/orders.csv',
  '<your-access-key>',
  '<your-access-key-secret>',
  'CSV',
  'uid UInt32, date DateTime, skuId UInt32, order_revenue UInt32');
```

返回信息如下。

```
┌──uid┬────────date─┬─skuId─┬─order_revenue─┐
│ 60333391 │ 2021-08-04 11:26:01 │ 49358700 │             89 │
│ 38826285 │ 2021-08-03 10:47:29 │ 25166907 │             27 │
│ 10793515 │ 2021-07-31 02:10:31 │ 95584454 │             68 │
│ 70246093 │ 2021-08-01 00:00:08 │ 82355887 │             97 │
│ 70149691 │ 2021-08-02 12:35:45 │ 68748652 │              1 │
│ 87307646 │ 2021-08-03 19:45:23 │ 16898681 │             71 │
│ 61694574 │ 2021-08-04 23:23:32 │ 79494853 │             35 │
│ 61337789 │ 2021-08-02 07:10:42 │ 23792355 │             55 │
│ 66879038 │ 2021-08-01 16:13:19 │ 95820038 │             89 │
└─────┴─────────┴─────┴─────────┘
```

4. 使用OSS表函数将数据导入至ClickHouse集群。

```
INSERT INTO test.orders_all
SELECT
  uid,
  date,
  skuId,
  order_revenue
FROM
  oss('http://test.oss-cn-beijing.aliyuncs.com/orders.csv',
      '<your-access-key>',
      '<your-access-secret>',
      'CSV',
      'uid UInt32, date DateTime, skuId UInt32, order_revenue UInt32');
```

语法

示例

OSS相关配置

1. 支持的profile

   如果使用MultipartUpload上传文件到OSS，则可以设置 `oss_min_upload_part_size` 参数以指定每个part最小的大小，默认值为512 MB，必须使用UInt64范围内的整数。

2. 设置方法

   - 在一次SQL中，代码设置如下。

     ```
     INSERT INTO OSS_TABLE
     SELECT
       ...
     FROM
       ...
     SETTINGS
       oss_min_upload_part_size=1073741824;
     ```

   - 在一次Session中，代码设置如下。

     ```
     SET oss_min_upload_part_size=1073741824;
     INSERT INTO OSS_TABLE
     SELECT
       ...
     FROM
       ...
     ;
     ```

   - 针对某一个表，代码设置如下。

     ```
     CREATE TABLE OSS_TABLE
     (
       ...
     ) ENGINE = OSS(...)
     SETTINGS
       oss_min_upload_part_size=1073741824;
     ```

   - 针对某一个用户，设置如下。

     在EMR控制台ClickHouse服务的**配置**页面，单击**server-users**页签，新增参数为**users.<YourUserName>.oss_min_upload_part_size**，参数值为1073741824的配置项。

EMR中的ClickHouse支持使用如下参数配置OSS，代码示例如下。

```
<oss>
    <endpoint-name>
        <endpoint>https://oss-cn-beijing.aliyuncs.com/bucket</endpoint>
        <access_key_id>ACCESS_KEY_ID</access_key_id>
        <secret_access_key>ACCESS_KEY_SECRET</secret_access_key>
    </endpoint-name>
</oss>
```

其中，相关参数描述如下。

参数	描述
endpoint-name	Endpoint的名称。

参数	描述
endpoint	OSS的访问域名，详情请参见OSS访问域名使用规则。
access_key_id	阿里云账号的AccessKey ID。
secret_access_key	阿里云账号的AccessKey Secret。

您也可以在EMR控制台ClickHouse服务的**配置**页面，单击**server-config**页签，通过以下两个方式新增自定义配置。

方式	操作
方法一	新增参数oss.<endpoint-name>.endpoint、oss.<endpoint-name>.access_key_id和oss.<endpoint-name>.secret_access_key及其对应的参数值。 **说明** 参数中的 `<endpoint-name>` 需要替换为Endpoint的名称。

方法二	新增参数为**oss**，参数值如下的配置项。
	`<endpoint-name>`
	`  <endpoint>https://oss-cn-beijing.aliyuncs.com/bucket</endpoint>`
	`  <access_key_id>ACCESS_KEY_ID</access_key_id>`
	`  <secret_access_key>ACCESS_KEY_SECRET</secret_access_key>`
	`</endpoint-name>`
	说明 参数值请替换为您实际的值。

如果您已进行如上配置，则在创建OSS表或使用OSS表函数时，可以使用如下命令。

- OSS表

```
CREATE TABLE OSS_TABLE
(
  column1 UInt32,
  column2 String
  ...
)
ENGINE = OSS(path, format, [compression]);
```

- OSS表函数

```
oss(path, format, structure, [compression]);
```

profile

configuration

6.3.4.5. 从RDS导入数据至ClickHouse

您可以通过RDS MySQL表引擎或表函数导入数据至ClickHouse集群。本文为您介绍如何将RDS中的数据导入至ClickHouse集群。

前提条件

- 已购买RDS，详情请参见创建RDS MySQL实例。
- 已创建ClickHouse集群，详情请参见创建集群。

使用RDS MySQL表引擎导入数据

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1] [TTL expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2] [TTL expr2],
    ...
) ENGINE = MySQL('host:port', 'database', 'table', 'user', 'password'[, replace_query, 'on_duplicate_clause']);
```

其中，涉及参数描述如下表所示。

参数	描述
db	数据库名。
table_name	表名。
cluster	集群标识。
name1/name2	列名。
tyep1/type2	列的类型。
host:port	RDS MySQL的地址，可以在RDS MySQL管理控制台中数据库连接中进行查看。
database	RDS MySQL中的数据库名。
table	RDS MySQL中的表名。
user	用户名，该用户具有访问上述RDS MySQL中库中的表的权限。
password	user 对应的密码。
replace_query	是否将INSERT INTO查询转换为REPLACE INTO的标志。设置为1，表示替换查询。
on_duplicate_clause	会被添加到INSERT语句中。例如， INSERT INTO t (c1,c2) VALUES ('a', 2) ON DUPLICATE KEY UPDATE c2 = c2 + 1 ，此时需要指定 on_duplicate_clause 为 UPDATE c2 = c2 + 1 。

1. 在RDS MySQL实例中，创建原始数据表并导入原始数据。
   i. 连接MySQL实例，详情请参见通过客户端、命令行连接RDS MySQL。
   ii. 执行以下命令，创建原始数据表。

```
CREATE TABLE `origin`.`orders` (
  `uid` int(10) unsigned DEFAULT NULL,
  `date` datetime DEFAULT NULL,
  `skuId` int(10) unsigned DEFAULT NULL,
  `order_revenue` int(10) unsigned DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

   iii. 执行以下命令，导入原始数据。

```
INSERT INTO `origin`.`orders` VALUES(60333391, '2021-08-04 11:26:01', 49358700, 89),
       (38826285, '2021-08-03 10:47:29', 25166907, 27),
       (10793515, '2021-07-31 02:10:31', 95584454, 68),
       (70246093, '2021-08-01 00:00:08', 82355887, 97),
       (70149691, '2021-08-02 12:35:45', 68748652, 1),
       (87307646, '2021-08-03 19:45:23', 16898681, 71),
       (61694574, '2021-08-04 23:23:32', 79494853, 35),
       (61337789, '2021-08-02 07:10:42', 23792355, 55),
       (66879038, '2021-08-01 16:13:19', 95820038, 89);
```

2. 在ClickHouse集群中，执行以下操作。
   i. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。
   ii. 执行以下命令，进入ClickHouse客户端。

```
clickhouse-client -m
```

   iii. 执行以下命令，创建数据库mysql。

```
CREATE DATABASE IF NOT EXISTS mysql;
```

   iv. 执行以下命令，创建表orders。

```
CREATE TABLE mysql.orders
(
    `uid` UInt32,
    `date` DateTime,
    `skuId` UInt32,
    `order_revenue` UInt32
)
ENGINE = MySQL('host:port', 'origin', 'orders', 'user', 'password');
```

v. 执行以下命令，创建数据库product。

```
CREATE DATABASE IF NOT EXISTS product ON CLUSTER cluster_emr;
```

vi. 执行以下命令，创建业务表orders。

```
CREATE TABLE IF NOT EXISTS product.orders ON CLUSTER cluster_emr
(
    `uid` UInt32,
    `date` DateTime,
    `skuId` UInt32,
    `order_revenue` UInt32
)
Engine = ReplicatedMergeTree('/cluster_emr/product/orders/{shard}', '{replica}')
PARTITION BY toYYYYMMDD(date)
ORDER BY toYYYYMMDD(date);
```

vii. 执行以下命令，创建业务表orders_all。

```
CREATE TABLE IF NOT EXISTS product.orders_all ON CLUSTER cluster_emr
(
    `uid` UInt32,
    `date` DateTime,
    `skuId` UInt32,
    `order_revenue` UInt32
)
Engine = Distributed(cluster_emr, product, orders, rand());
```

viii. 执行以下命令，导入数据。

```
INSERT INTO product.orders_all
SELECT
  uid,
  date,
  skuId,
  order_revenue
FROM
  mysql.orders;
```

ix. 执行以下命令，查询数据。

```
SELECT a.*
FROM mysql.orders AS a
ANTI LEFT JOIN product.orders_all USING (uid);
```

> **说明** 查询数据为空时正常。

语法

示例

使用RDS MySQL表函数导入数据

```
mysql('host:port', 'database', 'table', 'user', 'password'[, replace_query, 'on_duplicate_clause'])
```

其中，涉及参数描述如下表所示。

参数	描述
`host:port`	RDS MySQL的地址，您可以在RDS MySQL管理控制台中的数据库连接中查看。
`database`	RDS MySQL中的数据库名。
`table`	RDS MySQL中的表名。
`user`	用户名，该用户具有访问上述RDS MySQL中库中的表的权限。
`password`	`user` 对应的密码。
`replace_query`	是否将INSERT INTO查询转换为REPLACE INTO的标志。设置为1，表示替换查询。

参数	描述
on_duplicate_clause	会被添加到INSERT语句中。例如，INSERT INTO t (c1,c2) VALUES ('a', 2) ON DUPLICATE KEY UPDATE c2 = c2 + 1，此时需要指定on_duplicate_clause为UPDATE c2 = c2 + 1。

1. 在RDS MySQL实例中，创建表并插入数据。
   i. 连接MySQL实例，详情请参见通过客户端、命令行连接RDS MySQL。
   ii. 执行以下命令，创建表orders。

```
CREATE TABLE `origin`.`orders` (
  `uid` int(10) unsigned DEFAULT NULL,
  `date` datetime DEFAULT NULL,
  `skuId` int(10) unsigned DEFAULT NULL,
  `order_revenue` int(10) unsigned DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

   iii. 执行以下命令，插入数据。

```
INSERT INTO `origin`.`orders` VALUES(60333391, '2021-08-04 11:26:01', 49358700, 89),
       (38826285, '2021-08-03 10:47:29', 25166907, 27),
       (10793515, '2021-07-31 02:10:31', 95584454, 68),
       (70246093, '2021-08-01 00:00:08', 82355887, 97),
       (70149691, '2021-08-02 12:35:45', 68748652, 1),
       (87307646, '2021-08-03 19:45:23', 16898681, 71),
       (61694574, '2021-08-04 23:23:32', 79494853, 35),
       (61337789, '2021-08-02 07:10:42', 23792355, 55),
       (66879038, '2021-08-01 16:13:19', 95820038, 89);
```

2. 在ClickHouse集群中，执行以下操作。
   i. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。
   ii. 执行以下命令，进入ClickHouse客户端。

```
clickhouse-client -m
```

   iii. 执行以下命令，创建数据库product。

```
CREATE DATABASE IF NOT EXISTS product ON CLUSTER cluster_emr;
```

   iv. 执行以下命令，创建表orders。

```
CREATE TABLE IF NOT EXISTS product.orders ON CLUSTER cluster_emr
(
    `uid` UInt32,
    `date` DateTime,
    `skuId` UInt32,
    `order_revenue` UInt32
)
Engine = ReplicatedMergeTree('/cluster_emr/product/orders/{shard}', '{replica}')
PARTITION BY toYYYYMMDD(date)
ORDER BY toYYYYMMDD(date);
```

   v. 执行以下命令，创建表orders_all。

```
CREATE TABLE IF NOT EXISTS product.orders_all ON CLUSTER cluster_emr
(
    `uid` UInt32,
    `date` DateTime,
    `skuId` UInt32,
    `order_revenue` UInt32
)
Engine = Distributed(cluster_emr, product, orders, rand());
```

vi. 执行以下命令，导入数据。

```
INSERT INTO product.orders_all
SELECT
  uid,
  date,
  skuId,
  order_revenue
FROM
  mysql('host:port', 'origin', 'orders', 'user', 'password');
```

vii. 执行以下命令，查询数据。

```
SELECT a.*
FROM
  mysql('host:port', 'origin', 'orders', 'user', 'password') AS a
ANTI LEFT JOIN product.orders_all USING (uid);
```

> **说明** 查询数据为空时正常。

如果您需要导出数据，则将业务表数据写入MySQL表函数即可。写入命令如下。

```
INSERT INTO FUNCTION
  mysql('host:port', 'origin', 'orders', 'user', 'password')
FROM
  product.orders_all;
```

语法

示例

6.3.4.6. 从Kafka导入数据至ClickHouse

您可以通过Kafka表引擎导入数据至ClickHouse集群。本文为您介绍如何将Kafka中的数据导入至ClickHouse集群。

前提条件

- 已创建Kafka集群，详情请参见创建集群。
- 已创建ClickHouse集群，详情请参见创建集群。

使用限制

Kafka集群和ClickHouse集群需要在同一VPC下。

语法

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2],
    ...
) ENGINE = Kafka()
SETTINGS
    kafka_broker_list = 'host1:port1,host2:port2',
    kafka_topic_list = 'topic1,topic2,...',
    kafka_group_name = 'group_name',
    kafka_format = 'data_format';
```

其中，涉及参数描述如下表所示。

参数	描述
db	数据库名。
table_name	表名。
cluster	集群标识。
name1/name2	列名。
tyep1/type2	列的类型。

参数	描述
kafka_broker_list	Kafka Broker的地址及端口。 Kafka集群所有节点的内网IP地址及端口，您可以在EMR控制台**集群管理**页签中的**主机列表**页面查看。
kafka_topic_list	订阅的Topic名称。
kafka_group_name	Kafka consumer的分组名称。
kafka_format	数据的类型。例如，CSV和JSONEachRow等，详细信息请参见Formats for Input and Output Data。

示例

1. 在ClickHouse集群中执行以下操作。
   i. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。
   ii. 执行如下命令，进入ClickHouse客户端。

   ```
   clickhouse-client -m
   ```

   iii. 执行如下命令，创建数据库kafka。

   ```
   CREATE DATABASE IF NOT EXISTS kafka ON CLUSTER cluster_emr;
   ```

   > **说明** 数据库名您可以自定义。本文示例中的 cluster_emr 是集群默认的标识，如果您修改过，请填写正确的集群标识，您也可以在EMR控制台ClickHouse服务的**配置**页面，在搜索区域搜索clickhouse_remote_servers参数查看。

   iv. 执行如下命令，创建Kafka表。

   ```
   CREATE TABLE IF NOT EXISTS kafka.consumer ON CLUSTER cluster_emr
   (
       `uid` UInt32,
       `date` DateTime,
       `skuId` UInt32,
       `order_revenue` UInt32
   )
   ENGINE = Kafka()
   SETTINGS
     kafka_broker_list = '192.168.**.**:9092,192.168.**.**:9092,192.168.**.**:9092',
     kafka_topic_list = 'clickhouse_test',
     kafka_group_name = 'clickhouse_test',
     kafka_format = 'CSV';
   ```

   kafka_broker_list 为Kafka集群所有节点的内网IP地址及端口，您可以在EMR控制台**集群管理**页签中的**主机列表**页面查看。其余参数含义请参见语法。

v. 执行如下命令，创建数据库product。

```
CREATE DATABASE IF NOT EXISTS product ON CLUSTER cluster_emr;
```

vi. 执行以下命令，创建本地表。

```
CREATE TABLE IF NOT EXISTS product.orders ON CLUSTER cluster_emr
(
    `uid` UInt32,
    `date` DateTime,
    `skuId` UInt32,
    `order_revenue` UInt32
)
Engine = ReplicatedMergeTree('/cluster_emr/product/orders/{shard}', '{replica}')
PARTITION BY toYYYYMMDD(date)
ORDER BY toYYYYMMDD(date);
```

vii. 执行以下命令，创建分布式表。

```
CREATE TABLE IF NOT EXISTS product.orders_all ON CLUSTER cluster_emr
(
    `uid` UInt32,
    `date` DateTime,
    `skuId` UInt32,
    `order_revenue` UInt32
)
Engine = Distributed(cluster_emr, product, orders, rand());
```

viii. 执行以下命令，创建MATERIALIZED VIEW自动导数据。

```
CREATE MATERIALIZED VIEW IF NOT EXISTS product.kafka_load ON CLUSTER cluster_emr TO product.orders AS
SELECT *
FROM kafka.consumer;
```

2. 在Kafka集群中执行以下操作。

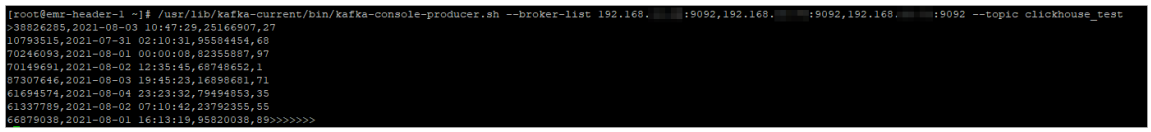
i. 使用SSH方式登录Kafka集群，详情请参见登录集群。

ii. 在Kafka集群的命令行窗口，执行如下命令运行Kafka的生产者。

```
/usr/lib/kafka-current/bin/kafka-console-producer.sh --broker-list 192.168.**.**:9092,192.168.**.**:9092,192.168.**
.**:9092 --topic clickhouse_test
```

iii. 执行以下命令，输入测试数据。

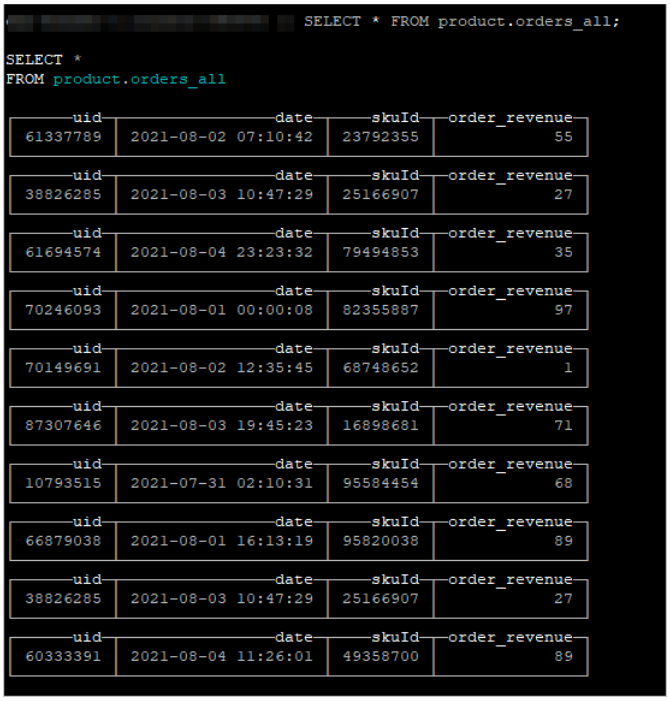
```
38826285,2021-08-03 10:47:29,25166907,27
10793515,2021-07-31 02:10:31,95584454,68
70246093,2021-08-01 00:00:08,82355887,97
70149691,2021-08-02 12:35:45,68748652,1
87307646,2021-08-03 19:45:23,16898681,71
61694574,2021-08-04 23:23:32,79494853,35
61337789,2021-08-02 07:10:42,23792355,55
66879038,2021-08-01 16:13:19,95820038,89
```



3. 在ClickHouse命令窗口中，执行以下命令，可以查看从Kafka中导入至ClickHouse集群的数据。

您可以校验查询到的数据与源数据是否一致。

```
SELECT * FROM product.orders_all;
```



6.3.5. 冷热分离

6.3.5.1. 使用HDFS进行数据冷热分离

本文为您介绍在阿里云E-MapReduce的ClickHouse集群上，如何通过HDFS进行数据的冷热分离。通过本文操作，您既可以在保证集群读写性能的基础上，自动维护集群上的冷热数据，又可以充分利用计算和存储资源，以降低成本。

前提条件

- 已在EMR控制台上创建EMR-5.5.0及以上版本的ClickHouse集群，详情请参见创建集群。
- 在同一VPC下具有一个HDFS服务（例如，EMR Hadoop集群）。
- 拥有HDFS服务的读写权限。

使用限制

本文操作仅适用于EMR-5.5.0及以上版本的ClickHouse集群。

操作流程

1. 步骤一：在EMR控制台添加磁盘
2. 步骤二：验证配置
3. 步骤三：进行冷热分离

步骤一：在EMR控制台添加磁盘

1. 进入ClickHouse配置页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 单击上方的**集群管理**页签。
   iii. 在**集群管理**页面，单击相应集群所在行的**详情**。
   iv. 在左侧导航栏中，选择**集群服务 > ClickHouse**。
   v. 在ClickHouse服务页面，单击**配置**页签。
2. 在**服务配置**区域，单击**server-metrika**页签。
3. 修改**storage_configuration**的参数值。

i. 在**disks**中添加一个HDFS类型的磁盘。

详细信息如下。

```
<disk_hdfs>
    <type>hdfs</type>
    <endpoint>hdfs://${your-hdfs-url}</endpoint>
    <min_bytes_for_seek>1048576</min_bytes_for_seek>
    <thread_pool_size>16</thread_pool_size>
    <objects_chunk_size_to_delete>1000</objects_chunk_size_to_delete>
</disk_hdfs>
```

相关参数描述如下。

参数	是否必填	描述
disk_hdfs	是	磁盘的名称，您可以自定义。
type	是	磁盘的类型，固定值为hdfs。
endpoint	是	HDFS服务的目录地址。 **注意** HDFS的地址通常是NameNode的地址，如果NameNode是HA模式，其端口通常为8020，否则为9000。
min_bytes_for_seek	否	最小使用Seek的Byte数量，低于该值时会用Skip代Seek。默认值为1048576。
thread_pool_size	否	用于Disk用于执行restore时所使用的线程池的大小。默认值为16。
objects_chunk_size_to_delete	否	一次最多可以删除HDFS文件的数量。默认为1000。

ii. 在**policies**中添加一个新的策略。

策略内容如下。

```
<hdfs_ttl>
    <volumes>
      <local>
        <!-- 包含默认存储策略下所有的磁盘 -->
        <disk>disk1</disk>
        <disk>disk2</disk>
        <disk>disk3</disk>
        <disk>disk4</disk>
      </local>
      <remote>
        <disk>disk_hdfs</disk>
      </remote>
    </volumes>
    <move_factor>0.2</move_factor>
</hdfs_ttl>
```

说明 该部分内容也可以直接添加在default策略中。

4. 保存配置。
   i. 在ClickHouse服务的**配置**页面，单击**保存**。
   ii. 在**确认修改**对话框中，输入执行原因，打开**自动更新配置**开关，单击**确定**。
5. 部署客户端配置。
   i. 在ClickHouse服务的**配置**页面，单击**部署客户端配置**。
   ii. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

步骤二：验证配置

1. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。
2. 执行如下命令，启动ClickHouse客户端。

```
clickhouse-client -m
```

3. 执行如下命令，查看磁盘信息。

```
select * from system.disks;
```

返回信息如下所示。

```
┌─name─────┬─path──────────────────────────────────┬─────────free_space─┬──────────total_space─┬─keep_free_space─┬─typ
e─┐
│ default  │ /var/lib/clickhouse/                  │          83868921856 │          84014424064 │               0 │ loc
al │
│ disk1    │ /mnt/disk1/clickhouse/                │          83858436096 │          84003938304 │        10485760 │ loc
al │
│ disk2    │ /mnt/disk2/clickhouse/                │          83928215552 │          84003938304 │        10485760 │ loc
al │
│ disk3    │ /mnt/disk3/clickhouse/                │          83928301568 │          84003938304 │        10485760 │ loc
al │
│ disk4    │ /mnt/disk4/clickhouse/                │          83928301568 │          84003938304 │        10485760 │ loc
al │
│ disk_hdfs│ /var/lib/clickhouse/disks/disk_hdfs/  │ 18446744073709551615 │ 18446744073709551615 │               0 │ hdf
s  │
└──────────┴───────────────────────────────────────┴──────────────────────┴──────────────────────┴─────────────────┴────
──┘
```

4. 执行如下命令，查看磁盘存储策略。

```
select * from system.storage_policies;
```

返回信息如下所示。

```
┌─policy_name─┬─volume_name─┬─volume_priority─┬─disks─────────────────────────┬─volume_type─┬─max_data_part_size─┬─
move_factor─┬─prefer_not_to_merge─┐
│ default     │ default     │               1 │ ['disk1','disk2','disk3','disk4'] │ JBOD        │                  0 │
0 │                   0 │
│ hdfs_ttl    │ local       │               1 │ ['disk1','disk2','disk3','disk4'] │ JBOD        │                  0 │
0.2 │                   0 │
│ hdfs_ttl    │ remote      │               2 │ ['disk_hdfs']                     │ JBOD        │                  0 │
0.2 │                   0 │
└─────────────┴─────────────┴─────────────────┴───────────────────────────────────┴─────────────┴────────────────────┴──
─────────────┴─────────────────────┘
3 rows in set. Elapsed: 0.001 sec.
```

当回显信息如上文所示时，表示磁盘扩容操作完成。

步骤三：进行冷热分离

1. 查看当前的存储策略。
   i. 在ClickHouse客户端执行如下命令，查看磁盘信息。

```
SELECT
  storage_policy
FROM system.tables
WHERE database='<database_name>' AND name='<table_name>';
```

命令中的 `<database_name>` 为数据库名，`<table_name>` 为表名。

如果返回信息如下所示，则需要参见下一步骤添加一个volume。

```
<default>
  <volumes>
    <default>
      <disk>disk1</disk>
      <disk>disk2</disk>
      <disk>disk3</disk>
      <disk>disk4</disk>
    </default>
  </volumes>
</default>
```

2. 扩展当前的存储策略。

在EMR控制台ClickHouse服务的配置页签，增加volume内容，详细信息如下。

```
<default>
  <volumes>
    <single>
      <disk>disk1</disk>
      <disk>disk2</disk>
      <disk>disk3</disk>
      <disk>disk4</disk>
    </single>
    <!-- 以下是新增的volume remote -->
    <remote>
      <disk>disk_hdfs</disk>
    </remote>
  </volumes>
  <!-- 多个volume时需要指定move_factor -->
  <move_factor>0.2</move_factor>
</default>
```

3. 执行以下命令，修改TTL。

```
ALTER TABLE <yourDataName>.<yourTableName>
  MODIFY TTL toStartOfMinute(addMinutes(t, 5)) TO VOLUME 'remote';
```

4. 执行以下命令，查看各个part的分布。

```
select partition,name,path from system.parts where database='<yourDataName>' and table='<yourTableName>' and active=1
```

返回信息如下。

```
┌─partition───────────┬─name─────────────────┬─path──────────────────────────────────────────────────────────────────────────
──────────────────────┐
│ 2022-01-12 11:30:00 │ 1641958200_1_96_3    │ /var/lib/clickhouse/disks/disk_hdfs/store/156/156008ff-41bf-460c-8848-e3
4fad88c25d/1641958200_1_96_3/ │
│ 2022-01-12 11:35:00 │ 1641958500_97_124_2  │ /mnt/disk3/clickhouse/store/156/156008ff-41bf-460c-8848-e34fad88c25d/164
1958500_97_124_2/             │
│ 2022-01-12 11:35:00 │ 1641958500_125_152_2 │ /mnt/disk4/clickhouse/store/156/156008ff-41bf-460c-8848-e34fad88c25d/164
1958500_125_152_2/            │
│ 2022-01-12 11:35:00 │ 1641958500_153_180_2 │ /mnt/disk1/clickhouse/store/156/156008ff-41bf-460c-8848-e34fad88c25d/164
1958500_153_180_2/            │
│ 2022-01-12 11:35:00 │ 1641958500_181_186_1 │ /mnt/disk4/clickhouse/store/156/156008ff-41bf-460c-8848-e34fad88c25d/164
1958500_181_186_1/            │
│ 2022-01-12 11:35:00 │ 1641958500_187_192_1 │ /mnt/disk3/clickhouse/store/156/156008ff-41bf-460c-8848-e34fad88c25d/164
1958500_187_192_1/            │
└─────────────────────┴──────────────────────┴──────────────────────────────────────────────────────────────────────────
──────────────────────┘

6 rows in set. Elapsed: 0.002 sec.
```

> 说明 如果返回信息如上所示，则表明数据根据时间做了冷热分离。热数据存放在本地盘中，冷数据存放在HDFS中。

其中，*/var/lib/clickhouse/disks/disk_hdfs*是disk_hdfs元数据的目录，*/mnt/disk{1..4}/clickhouse*是本地盘路径。

- 创建语法

```
CREATE TABLE <yourDataName>.<yourTableName> [ON CLUSTER cluster_emr]
(
  column1 Type1,
  column2 Type2,
  ...
) Engine = MergeTree() -- 也可以使用Replicated*MergeTree()
PARTITION BY <yourPartitionKey>
ORDER BY <yourPartitionKey>
TTL <yourTtlKey> TO VOLUME 'remote'
SETTINGS storage_policy='hdfs_ttl';
```

> 说明 命令中的<yourPartitionKey>为ClickHouse的分区键。<yourTtlKey>为您设置的TTL信息。

- 示例

```
CREATE TABLE test.test
(
    `id` UInt32,
    `t` DateTime
)
ENGINE = MergeTree()
PARTITION BY toStartOfFiveMinute(t)
ORDER BY id
TTL toStartOfMinute(addMinutes(t, 5)) TO VOLUME 'remote'
SETTINGS storage_policy='hdfs_ttl';
```

说明 本示例中，表格会将5分钟内的数据存放在本地，过了5分钟后，数据会被移动到remote volume中，也就是HDFS中。

对已有的表进行改造

创建新的表

相关配置

- **server-config**

  merge_tree.allow_remote_fs_zero_copy_replication：设置为true，以在Replicated*MergeTree使用DiskHDFS等远程存储时利用其自身的多副本进行备份，ClickHouse的一个Shard下的多个副本中的数据仅会备份元数据。

- **server-users**

  profile.${your-profile-name}.hdfs_replication：设置数据在HDFS上存储的副本个数。

6.3.5.2. 使用OSS进行数据冷热分离

本文为您介绍在阿里云E-MapReduce的ClickHouse集群上，如何通过OSS进行数据的冷热分离。通过本文操作，您既可以在保证集群读写性能的基础上，自动维护集群上的冷热数据，又可以充分利用计算和存储资源，以降低成本。

前提条件

已在EMR控制台上创建EMR-5.5.0及以上版本的ClickHouse集群，详情请参见创建集群。

使用限制

本文操作仅适用于EMR-5.5.0及以上版本的ClickHouse集群。

操作流程

1. 步骤一：在EMR控制台添加磁盘
2. 步骤二：验证配置
3. 步骤三：进行冷热分离

步骤一：在EMR控制台添加磁盘

1. 进入ClickHouse配置页面。
   i. 登录阿里云E-MapReduce控制台。
   ii. 单击上方的**集群管理**页签。
   iii. 在**集群管理**页面，单击相应集群所在行的**详情**。
   iv. 在左侧导航栏中，选择**集群服务 > ClickHouse**。
   v. 在ClickHouse服务页面，单击**配置**页签。
2. 在**服务配置**区域，单击**server-metrika**页签。
3. 修改**storage_configuration**的参数值。
   i. 在**disks**中添加一个OSS类型的磁盘。
   详细信息如下。

```
<disk_oss>
    <type>oss</type>
    <endpoint>http(s)://${yourBucketName}.${yourEndpoint}/${yourFlieName}</endpoint>
    <access_key_id>${yourAccessKeyId}</access_key_id>
    <secret_access_key>${yourAccessKeySecret}</secret_access_key>
    <send_metadata>false</send_metadata>
    <metadata_path>${yourMetadataPath}</metadata_path>
    <cache_enabled>true</cache_enabled>
    <cache_path>${yourCachePath}</cache_path>
    <skip_access_check>false</skip_access_check>
    <min_bytes_for_seek>1048576</min_bytes_for_seek>
    <thread_pool_size>16</thread_pool_size>
    <list_object_keys_size>1000</list_object_keys_size>
  </disk_oss>
```

相关参数描述如下。

参数	是否必填	描述
disk_oss	是	磁盘的名称，您可以自定义。
type	是	磁盘的类型，固定值为oss。
endpoint	是	OSS服务的地址。格式为*http(s)://${yourBucketName}.${yourEndpoint}/${yourFlieName}*。 **说明** endpoint参数值必须以HTTP或HTTPS开头，其中，`${yourBucketName}` 为OSS的Bucket名称，`${yourEndpoint}` 为OSS的访问域名，`{yourFlieName}` 为OSS上的文件名称。例如 `http://clickhouse.oss-cn-hangzhou-internal.aliyuncs.com/test` 。
access_key_id	是	阿里云账号的AccessKey ID。 获取方法请参见获取AccessKey。
secret_access_key	是	阿里云账号的AccessKey Secret。 用于加密签名字符串和OSS，用来验证签名字符串的密钥。获取方法请参见获取AccessKey。
send_metadata	否	在操作OSS文件时，是否添加元数据信息。参数取值如下： ■ true：添加元数据信息。 ■ false（默认值）：不添加元数据信息。
metadata_path	否	用于存放本地文件与OSS文件的映射关系。 默认值为*${path}/disks/<disk_name>/*。 **说明** `<disk_name>` 是磁盘的名称，对应参数disk_oss。
cache_enabled	否	是否开启缓存。参数取值如下： ■ true（默认值）：开启缓存。 ■ false：不开启缓存。 Cache具体作用如下： ■ OSS目前所使用的Cache仅用于本地缓存以*.idx*、*.mrk*、*.mrk2*、*.mrk3*、*.txt*和*.dat*为后缀的文件。除了这些文件外，仍然会直接读OSS而非读缓存。 ■ 本地Cache没有容量限制，最大容量即为存储的磁盘容量。 ■ 本地Cache不会以LRU（Least Recently Used）算法之类的策略清理缓存，而是随着文件的生命周期而存在。 ■ 数据第一次被读取时，如果在本地Cache中不存在，则会从OSS中将文件下载至Cache中。 ■ 第一次写入数据时，仅会先写入本地Cache，之后才会从本地Cache写入至OSS中。 ■ 如果一个文件在OSS中被删除，则也会将本地Cache清除；如果一个文件在OSS中被重命名，则也会在本地Cache中重命名。
cache_path	否	缓存路径。 默认值为*${path}/disks/<disk_name>/cache/*。

参数	是否必填	描述
skip_access_check	否	在加载磁盘时，是否检查具有对磁盘的读写权限。参数取值如下： ■ true（默认值）：检查。 ■ false：不检查。
min_bytes_for_seek	否	最小使用Seek的Byte数量，低于该值时会用Skip代替Seek。默认值为1048576。
thread_pool_size	否	磁盘用于执行 `restore` 命令时所使用的线程池的大小。默认值为16。
list_object_keys_size	否	在某一个key下，单次能够列出的对象最大数目。默认值为1000。

ii. 在**policies**中添加一个新的策略。

策略内容如下。

```
<oss_ttl>
    <volumes>
      <local>
        <!-- 包含默认存储策略下所有的磁盘 -->
        <disk>disk1</disk>
        <disk>disk2</disk>
        <disk>disk3</disk>
        <disk>disk4</disk>
      </local>
      <remote>
        <disk>disk_oss</disk>
      </remote>
    </volumes>
    <move_factor>0.2</move_factor>
</oss_ttl>
```

> **说明** 该部分内容也可以直接添加在default策略中。

4. 保存配置。
   i. 在ClickHouse服务的**配置**页面，单击**保存**。
   ii. 在**确认修改**对话框中，输入执行原因，打开**自动更新配置**开关，单击**确定**。
5. 部署客户端配置。
   i. 在ClickHouse服务的**配置**页面，单击**部署客户端配置**。
   ii. 在**执行集群操作**对话框中，输入执行原因，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

步骤二：验证配置

1. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。
2. 执行如下命令，启动ClickHouse客户端。

```
clickhouse-client -m
```

3. 执行如下命令，查看磁盘信息。

```
select * from system.disks;
```

返回信息如下所示。

```
┌─name─────┬─path──────────────────────────────┬──────────free_space─┬─────────total_space─┬─keep_free_space─┬─type
──┐
│ default  │ /var/lib/clickhouse/              │          83868921856 │          84014424064 │               0 │ loca
l │
│ disk1    │ /mnt/disk1/clickhouse/            │          83858436096 │          84003938304 │        10485760 │ loca
l │
│ disk2    │ /mnt/disk2/clickhouse/            │          83928215552 │          84003938304 │        10485760 │ loca
l │
│ disk3    │ /mnt/disk3/clickhouse/            │          83928301568 │          84003938304 │        10485760 │ loca
l │
│ disk4    │ /mnt/disk4/clickhouse/            │          83928301568 │          84003938304 │        10485760 │ loca
l │
│ disk_oss │ /var/lib/clickhouse/disks/disk_oss/ │ 18446744073709551615 │ 18446744073709551615 │               0 │ oss
  │
└──────────┴───────────────────────────────────┴──────────────────────┴──────────────────────┴─────────────────┴─────
──┘
```

4. 执行如下命令，查看磁盘存储策略。

```
select * from system.storage_policies;
```

返回信息如下所示。

```
┌─policy_name─┬─volume_name─┬─volume_priority─┬─disks─────────────────────────┬─volume_type─┬─max_data_part_size─┬─
move_factor─┬─prefer_not_to_merge─┐
│ default     │ default     │               1 │ ['disk1','disk2','disk3','disk4'] │ JBOD        │                  0 │
0 │                   0 │
│ oss_ttl     │ local       │               1 │ ['disk1','disk2','disk3','disk4'] │ JBOD        │                  0 │
0.2 │                   0 │
│ oss_ttl     │ remote      │               2 │ ['disk_oss']                      │ JBOD        │                  0 │
0.2 │                   0 │
└─────────────┴─────────────┴─────────────────┴───────────────────────────────────┴─────────────┴────────────────────┴─
────────────┴─────────────────────┘
3 rows in set. Elapsed: 0.001 sec.
```

当回显信息如上文所示时，表示磁盘扩容操作完成。

步骤三：进行冷热分离

1. 在ClickHouse客户端执行如下命令，查看当前的存储策略。

```
SELECT
  storage_policy
FROM system.tables
WHERE database='<yourDatabaseName>' AND name='<yourTableName>';
```

本文示例中命令中的 `<yourDataName>` 为数据库名， `<yourTableName>` 为表名。

如果返回信息如下所示，则需要参见下一步骤添加一个volume。

```
<default>
  <volumes>
    <default>
      <disk>disk1</disk>
      <disk>disk2</disk>
      <disk>disk3</disk>
      <disk>disk4</disk>
    </default>
  </volumes>
</default>
```

2. 扩展当前的存储策略。

在EMR控制台ClickHouse服务的配置页签，增加volume内容，详细信息如下。

```
<default>
  <volumes>
    <default>
      <disk>disk1</disk>
      <disk>disk2</disk>
      <disk>disk3</disk>
      <disk>disk4</disk>
    </default>
    <!-- 以下是新增的volume remote -->
    <remote>
      <disk>disk_oss</disk>
    </remote>
  </volumes>
  <!-- 多个volume时需要指定move_factor -->
  <move_factor>0.2</move_factor>
</default>
```

3. 执行以下命令，修改TTL。

```
ALTER TABLE <yourDataName>.<yourTableName>
  MODIFY TTL toStartOfMinute(addMinutes(t, 5)) TO VOLUME 'remote';
```

4. 执行以下命令，查看各个part的分布。

```
select partition,name,path from system.parts where database='<yourDataName>' and table='<yourTableName>' and active=1
```

返回信息如下。

```
┌─partition───────────┬─name─────────────────────┬─path──────────────────────────────────────────────────────────────────
──────────────────────────────────┐
│ 2022-01-11 19:55:00 │ 1641902100_1_90_3_193    │ /var/lib/clickhouse/disks/disk_oss/store/fc5/fc50a391-4c16-406b-a396
-6e1104873f68/1641902100_1_90_3_193/  │
│ 2022-01-11 19:55:00 │ 1641902100_91_96_1_193   │ /var/lib/clickhouse/disks/disk_oss/store/fc5/fc50a391-4c16-406b-a396
-6e1104873f68/1641902100_91_96_1_193/ │
│ 2022-01-11 20:00:00 │ 1641902400_97_124_2_193  │ /mnt/disk3/clickhouse/store/fc5/fc50a391-4c16-406b-a396-6e1104873f68
/1641902400_97_124_2_193/          │
│ 2022-01-11 20:00:00 │ 1641902400_125_152_2_193 │ /mnt/disk2/clickhouse/store/fc5/fc50a391-4c16-406b-a396-6e1104873f68
/1641902400_125_152_2_193/         │
│ 2022-01-11 20:00:00 │ 1641902400_153_180_2_193 │ /mnt/disk4/clickhouse/store/fc5/fc50a391-4c16-406b-a396-6e1104873f68
/1641902400_153_180_2_193/         │
│ 2022-01-11 20:00:00 │ 1641902400_181_186_1_193 │ /mnt/disk3/clickhouse/store/fc5/fc50a391-4c16-406b-a396-6e1104873f68
/1641902400_181_186_1_193/         │
│ 2022-01-11 20:00:00 │ 1641902400_187_192_1_193 │ /mnt/disk4/clickhouse/store/fc5/fc50a391-4c16-406b-a396-6e1104873f68
/1641902400_187_192_1_193/         │
└─────────────────────┴──────────────────────────┴──────────────────────────────────────────────────────────────────────
──────────────────────────────────┘
7 rows in set. Elapsed: 0.002 sec.
```

说明 如果结果类似上所示，则表明数据根据时间做了冷热分离。热数据存放在本地盘中，冷数据存放在OSS中。

其中，*/var/lib/clickhouse/disks/disk_oss*是disk_oss的metadata_path的默认值。*/mnt/disk{1..4}/clickhouse*是本地盘路径。

- 创建语法

```
CREATE TABLE <yourDataName>.<yourTableName> [ON CLUSTER cluster_emr]
(
  column1 Type1,
  column2 Type2,
  ...
) Engine = MergeTree() -- or Replicated*MergeTree()
PARTITION BY <yourPartitionKey>
ORDER BY <yourPartitionKey>
TTL <yourTtlKey> TO VOLUME 'remote'
SETTINGS storage_policy='oss_ttl';
```

说明 命令中的<yourPartitionKey>为ClickHouse的分区键。<yourTtlKey>为您设置的TTL信息。

- 示例

```
CREATE TABLE test.test
(
    `id` UInt32,
    `t` DateTime
)
ENGINE = MergeTree()
PARTITION BY toStartOfFiveMinute(t)
ORDER BY id
TTL toStartOfMinute(addMinutes(t, 5)) TO VOLUME 'remote'
SETTINGS storage_policy='oss_ttl';
```

说明 本示例中，表格会将5分钟内的数据存放在本地，过了5分钟后，数据会被移动到remote volume中，也就是OSS中。

对已有的表进行改造

创建新的表

相关配置

- server-config

  merge_tree.allow_remote_fs_zero_copy_replication：设置为true，以在Replicated*MergeTree使用DiskOSS等远程存储时利用OSS的多副本进行备份，ClickHouse的一个Shard下的多个副本中的数据仅会备份元数据。

- server-users
  - profile.${your-profile-name}.oss_min_upload_part_size：Write Buffer中的数据量高于该参数值时，会将数据写到OSS中。
  - profile.${your-profile-name}.oss_max_single_part_upload_size}：Write Buffer中的数据量高于该参数值时，使用MultipartUpload，详情请参见分片上传（MultipartUpload）。

6.3.6. 事务使用

事务是阿里云E-MapReduce（简称EMR）产品自研的特性，目前处于Experimental阶段。本文为您介绍事务的状态、参数以及事务的相关操作。

前提条件

已在EMR控制台上创建ClickHouse集群，且集群中已配置了可使用的ZooKeeper服务，创建详情请参见创建集群。

使用限制

本文操作适用于EMR-3.39.0及以上版本和EMR-5.5.0及以上版本的ClickHouse集群。

注意事项

- 事务仅支持Insert，且Insert的对象必须是以Replicated*MergeTree或者*MergeTree为存储引擎的表。
- 当前的事务为单机事务。
- 本功能处在Experimental阶段，请谨慎使用。

事务API

事务允许的操作如下表所示。

操作	描述
begin	开启一个事务。
write_data	在一个事务内写数据。
commit	提交一个事务。
rollback	回滚一个未提交的事务。

事务状态如下表所示。

状态	描述
UNKNOWN	事务未开启，此时仅允许begin操作。
INITIALIZED	事务已开启，此时允许所有操作。
COMMITTING	事务正在被提交，不允许执行begin或write_data两种操作。

状态	描述
COMMITTED	事务已经被提交，不再允许任何操作。
ABORTING	事务正在被回滚，不再允许任何操作。
ABORTED	事务已经被回滚，不再允许任何操作。

错误码	描述
0	执行成功。
11001	未知的事务操作类型。
11002	未知的事务状态。
11003	多个机器存在一个相同的事务处理。
11004	重定向至其他机器。
11005	不支持的存储引擎。
11006	事务未在任何机器上处理。
11201	当前事务状态为COMMITTING，但收到了begin请求。
11202	当前事务状态为COMMITTED，但收到了begin请求。
11203	当前事务状态为ABORTING，但收到了begin请求。
11204	当前事务状态为ABORTED，但收到了begin请求。
11301	当前事务状态为UNKNOWN，但收到了commit请求。
11302	当前事务状态为ABORTING，但收到了commit请求。
11303	当前事务状态为ABORTED，但收到了commit请求。
11401	当前事务状态为UNKNOWN，但收到了rollback请求。
11402	当前事务状态为COMMITTED，但收到了rollback请求。
11501	当前事务状态为UNKNOWN，但收到了write_data请求。
11502	当前事务状态为COMMITTING，但收到了write_data请求。
11503	当前事务状态为COMMITTED，但收到了write_data请求。
11504	当前事务状态为ABORTING，但收到了write_data请求。

名称	类型	是否必选	描述
action	字符串	是	事务的操作。
id	字符串	是	事务的ID。
stacktrace	布尔值	否	是否在执行出错时向Client发送详细的堆栈信息，默认值为false。

名称	类型	是否必选	描述
Host	字符串	是	ClickHouse HTTP Server的可访问地址，通常由*ip:port*或*hostname:port*组成，其中port通常为8123。
Authorization	字符串	否	ClickHouse用以验证身份的字段，使用HTTP Basic Authentication作为验证手段。 注意 如果指定了该验证方式，则不能指定X-ClickHouse-User和X-ClickHouse-Key，否则会报错。

名称	类型	是否必选	描述
X-ClickHouse-User	字符串	否	ClickHouse用户名。 注意 通常不推荐该验证方式，因为该验证方式为明文。指定该验证方式，则不能指定 Authorization，否则会报错。
X-ClickHouse-Key	字符串	否	ClickHouse用户密码。 注意 通常不推荐该验证方式，因为该验证方式为明文。指定该验证方式，则不能指定 Authorization，否则会报错。
X-ClickHouse-TransactionId	字符串	否	ClickHouse事务ID，仅用于事务写。

名称	类型	描述
Location	字符串	重定向的位置。

Response Body会以JSON的形式返回，JSON内含字段如下。

名称	类型	描述
code	整型	错误码。
message	字符串	错误信息。

ACTION TYPE

STATE TYPE

错误码

HTTP PARAM

Request Header

Response Header

Response Body

使用示例

在emr-header-1节点上开始一个ID为1f6676e3-496f-409e-b64e-0eb22b1c****的事务，执行命令如下。

```
POST /transaction?action=begin&id=1f6676e3-496f-409e-b64e-0eb22b1c**** HTTP/1.1
Host: emr-header-1:8123
Authorization: Basic ZGVmYXVsdDo=
```

- 执行成功时，返回如下信息。

```
HTTP/1.1 200 OK
Date: Fri, 14 Jan 2022 07:45:32 GMT
Connection: Close
Content-Type: application/json;charset=UTF-8
{
  "code": 0,
  "message": "Success."
}
```

- 执行失败时，返回如下类似信息。

```
HTTP/1.1 200 OK
Date: Fri, 14 Jan 2022 08:00:43 GMT
Connection: Close
Content-Type: application/json;charset=UTF-8
{
  "code": 11202,
  "message": "Received the request of transaction begin after the transaction was committed."
}
```

- 重定向时，返回如下信息。

```
HTTP/1.1 307 OK
Date: Fri, 14 Jan 2022 07:47:18 GMT
Connection: Close
Location: http://192.168.**.**:8123/transaction?action=begin&id=1f6676e3-496f-409e-b64e-0eb22b1c****
Content-Type: application/json;charset=UTF-8
{
  "code": 11004,
  "message": "Redirect to other replica."
}
```

> **说明** 向其他节点上发送请求会被重定向至emr-header-1，其IP地址为192.168.**.**。

在emr-header-1节点上，向以下表结构中写入数据，写的过程包含在事务1f6676e3-496f-409e-b64e-0eb22b1c****中。

```
CREATE TABLE test.test ON CLUSTER cluster_emr
(
    `id` UInt32,
    `t` DateTime
)
ENGINE = MergeTree
PARTITION BY toStartOfFiveMinute(t)
ORDER BY id;
```

执行命令如下。

```
POST /?query=INSERT%20INTO%20test.test%20VALUES%20(1,'2022-01-13%2020:03:01') HTTP/1.1
Authorization: Basic ZGVmYXVsdDo=
Host: emr-header-1.cluster-276031:8123
Content-Length: 0
X-ClickHouse-TransactionId: 31a8b711-c174-481a-be9d-e2c229c8****
```

- 执行成功时，返回如下信息。

```
HTTP/1.1 200 OK
Date: Fri, 14 Jan 2022 07:53:20 GMT
Connection: Keep-Alive
Content-Type: text/tab-separated-values; charset=UTF-8
X-ClickHouse-Server-Display-Name: emr-header-1.cluster-276031
Transfer-Encoding: chunked
X-ClickHouse-Query-Id: 1135ae61-a38d-400d-94e1-df3a67cd****
X-ClickHouse-Format: TabSeparated
X-ClickHouse-Timezone: Asia/Shanghai
Keep-Alive: timeout=3
X-ClickHouse-Summary: {"read_rows":"0","read_bytes":"0","written_rows":"0","written_bytes":"0","total_rows_to_read":"0"}
```

- 执行失败时，返回如下类似信息。

```
HTTP/1.1 500 Internal Server Error
Date: Fri, 14 Jan 2022 08:01:54 GMT
Connection: Keep-Alive
Content-Type: text/tab-separated-values; charset=UTF-8
X-ClickHouse-Server-Display-Name: emr-header-1.cluster-276031
Transfer-Encoding: chunked
X-ClickHouse-Query-Id: c9a65f05-d52e-4f2a-a892-77f11799****
X-ClickHouse-Format: TabSeparated
X-ClickHouse-Timezone: Asia/Shanghai
X-ClickHouse-Exception-Code: 11503
Keep-Alive: timeout=3
X-ClickHouse-Summary: {"read_rows":"0","read_bytes":"0","written_rows":"0","written_bytes":"0","total_rows_to_read":"0"}
Code: 11503, e.displayText() = DB::Exception: Received the request of writing data after the transaction was committed. (
version 21.3.13.1)
```

- 重定向时，返回如下信息。

```
HTTP/1.1 307 Temporary Redirect
Date: Fri, 14 Jan 2022 07:52:15 GMT
Connection: Keep-Alive
Content-Type: text/plain; charset=UTF-8
X-ClickHouse-Server-Display-Name: emr-worker-1.cluster-276031
Transfer-Encoding: chunked
Location: http://192.168.**.**:8123/?query=INSERT%20INTO%20test.test%20VALUES%20(1,'2022-01-13%2020:04:21')
X-ClickHouse-Exception-Code: 11004
Code: 11004, e.displayText() = DB::Exception: Failed to check replica under transaction path in zookeeper. (version 21.3.
13.1)
```

说明 向其他节点上发送请求会被重定向至emr-header-1，其IP地址为192.168.**.**。

在emr-header-1节点上提交一个ID为1f6676e3-496f-409e-b64e-0eb22b1c****的事务，执行命令如下。

```
POST /transaction?action=commit&id=1f6676e3-496f-409e-b64e-0eb22b1c**** HTTP/1.1
Host: emr-header-1:8123
Authorization: Basic ZGVmYXVsdDo=
```

- 提交成功时，返回如下信息。

```
HTTP/1.1 200 OK
Date: Fri, 14 Jan 2022 07:55:09 GMT
Connection: Close
Content-Type: application/json;charset=UTF-8
{
  "code": 0,
  "message": "Success."
}
```

- 提交失败时，返回如下类似信息。

```
HTTP/1.1 200 OK
Date: Fri, 14 Jan 2022 08:03:39 GMT
Connection: Close
Content-Type: application/json;charset=UTF-8
{
  "code": 11303,
  "message": "Received the request of transaction commit after the transaction was aborted."
}
```

- 重定向时，返回如下信息。

```
HTTP/1.1 307 OK
Date: Fri, 14 Jan 2022 07:56:30 GMT
Connection: Close
Location: http://192.168.**.**:8123/transaction?action=commit&id=1f6676e3-496f-409e-b64e-0eb22b1c****
Content-Type: application/json;charset=UTF-8
{
  "code": 11004,
  "message": "Redirect to other replica."
}
```

说明 向其他节点上发送请求会被重定向至emr-header-1，其IP地址为192.168.**.**。

在emr-header-1节点上回滚一个ID为1f6676e3-496f-409e-b64e-0eb22b1c****的事务，执行命令如下。

```
POST /transaction?action=rollback&id=1f6676e3-496f-409e-b64e-0eb22b1c**** HTTP/1.1
Host: emr-header-1:8123
Authorization: Basic ZGVmYXVsdDo=s
```

- 回滚成功时，返回如下信息。

```
HTTP/1.1 200 OK
Date: Fri, 14 Jan 2022 08:03:08 GMT
Connection: Close
Content-Type: application/json;charset=UTF-8
{
  "code": 0,
  "message": "Success."
}
```

- 回滚失败时，返回如下类似信息。

```
HTTP/1.1 200 OK
Date: Fri, 14 Jan 2022 07:58:41 GMT
Connection: Close
Content-Type: application/json;charset=UTF-8
{
  "code": 11402,
  "message": "Received the request of transaction rollback after the transaction was committed."
}
```

- 重定向时，返回如下信息。

```
HTTP/1.1 307 OK
Date: Fri, 14 Jan 2022 07:59:33 GMT
Connection: Close
Location: http://192.168.**.**:8123/transaction?action=rollback&id=1f6676e3-496f-409e-b64e-0eb22b1c****
Content-Type: application/json;charset=UTF-8
{
  "code": 11004,
  "message": "Redirect to other replica."
}
```

> **说明** 向其他节点上发送请求会被重定向至emr-header-1，其IP地址为192.168.**.**。

开始事务

事务写

提交事务

回滚事务

相关配置

参数	描述
transaction.enable_public_ip	ClickHouse Server中Transaction需要一个用于标识自身的IP地址，默认使用私网IP地址。 在EMR控制台ClickHouse服务的**server-config**页签中，设置此参数值为true，以使用公网IP地址标识自身，但需要所有节点均开启公网IP。
transaction.path_prefix	Transaction在ZooKeeper中的目录前缀。默认为*/clickhouse/transactions*。
transaction.transaction_timeout_ms	Transaction超时时间，默认值为86400000毫秒（1天）。
transaction.auxiliary_zookeepers	Transaction使用的ZooKeeper名称，默认为default，对应了配置在EMR控制台ClickHouse服务的**server-metrika**页签中参数**zookeeper_servers**的Zookeeper。
transaction.distributed_lock_timeout_ms	分布式锁超时的时间，默认值为120000毫秒（2分钟）。

6.3.7. 常见问题

本文汇总了ClickHouse使用时的常见问题。

- 如何创建ClickHouse用户？
- 如何修改default用户的密码？
- 如何修改ClickHouse集群标识？
- 如何设置ClickHouse多盘存储？
- 数据丢失，如何处理？
- 如何扩缩容磁盘？

- 异常处理
  - 报错提示Memory limit (for total) exceeded时，该如何处理？
  - 报错提示Memory limit (for query) exceeded时，该如何处理？
  - 报错提示Memory limit (for user) exceeded时，该如何处理？
  - 报错提示Too many parts in all partitions in total，该如何处理？
  - 报错提示Too many parts，该如何处理？

如何创建ClickHouse用户？

您可以通过以下两种方法创建ClickHouse用户：

- 通过在EMR控制台新增自定义配置项创建ClickHouse用户

  在EMR控制台ClickHouse服务的**配置**页面，单击**server-users**页签，新增参数为**users.<YourUserName>.password**或**users.<YourUserName>.password_sha256_hex**或**users.<YourUserName>.password_double_sha1_hex**，参数值您可以自定义的配置项，保存该配置项并重启服务，即可创建用户。

  参数中的<YourUserName>需要替换为您待创建用户的名称。

  > **说明** 添加组件参数详情，请参见管理组件参数。重启服务详情，请参见重启服务。

- 通过ClickHouse客户端创建ClickHouse用户

  i. 在EMR控制台ClickHouse服务的**配置**页面，单击**server-users**页签，新增参数为**users.default.access_management**，参数值为1的配置项，保存该配置并重启服务。使用默认用户连接ClickHouse集群。

  ii. 使用SSH方式登录ClickHouse集群，详情请参见登录集群。

  iii. 执行以下命令，进入ClickHouse客户端。

  ```
  clickhouse-client -m
  ```

  iv. 执行以下命令，创建用户。

  ```
  CREATE USER IF NOT EXISTS user_test ON CLUSTER new_cluster_emr IDENTIFIED WITH plaintext_password BY '123456';
  ```

  返回信息如下图所示。

  ```
  emr-header-1.cluster-240[redacted] :) CREATE USER IF NOT EXISTS user_0818 ON CLUSTER new_cluster_emr IDENTIFIED BY '123456';

  CREATE USER IF NOT EXISTS user_0818 ON CLUSTER new_cluster_emr IDENTIFIED WITH sha256_hash BY '8D969EEF6ECAD3C29A3A629280E686CF0C3F5D[redacted]'

  ┌─host──────────────────────────┬─port─┬─status─┬─error─┬─num_hosts_remaining─┬─num_hosts_active─┐
  │ emr-worker-2.cluster-240[redacted] │ 9000 │      0 │       │                   3 │                0 │
  │ emr-worker-1.cluster-240[redacted] │ 9000 │      0 │       │                   2 │                0 │
  │ emr-header-1.cluster-240[redacted] │ 9000 │      0 │       │                   1 │                0 │
  │ emr-worker-3.cluster-240[redacted] │ 9000 │      0 │       │                   0 │                0 │
  └───────────────────────────────┴──────┴────────┴───────┴─────────────────────┴──────────────────┘

  4 rows in set. Elapsed: 0.119 sec.
  ```

  创建用户命令格式如下。

  ```
  CREATE USER [IF NOT EXISTS | OR REPLACE] name1 [ON CLUSTER cluster_name1]
          [, name2 [ON CLUSTER cluster_name2] ...]
      [NOT IDENTIFIED | IDENTIFIED {[WITH {no_password | plaintext_password | sha256_password | sha256_hash | double_sha1_password | double_sha1_hash}] BY {'password' | 'hash'}} | {WITH ldap SERVER 'server_name'} | {WITH kerberos [REALM 'realm']}]
      [HOST {LOCAL | NAME 'name' | REGEXP 'name_regexp' | IP 'address' | LIKE 'pattern'} [,...] | ANY | NONE]
      [DEFAULT ROLE role [,...]]
      [GRANTEES {user | role | ANY | NONE} [,...] [EXCEPT {user | role} [,...]]]
      [SETTINGS variable [= value] [MIN [=] min_value] [MAX [=] max_value] [READONLY | WRITABLE] | PROFILE 'profile_name'] [,...];
  ```

  v. 执行以下命令，查看已有的用户。

  ```
  SHOW USERS;
  ```

  返回已有的用户。

  ```
  ┌─name─────┐
  │ default  │
  │ user_test │
  │ user_test2 │
  └──────────┘
  ```

如何修改default用户的密码？

> **注意** 暂不支持将default密码修改为sha256_hex等加密类型的文本。

您可以在EMR控制台ClickHouse服务的**配置**页面，修改以下配置项以修改default用户的密码。

1. 单击**server-users**页签，修改参数users.default.password，参数值您可以自定义。
2. 单击**client-config**页签，修改参数password，参数值为您自定义的密码，即**users.default.password**的参数值。
3. 单击**server-metrika**页签，修改参数clickhouse_remote_servers，参数值格式如下。

```
<cluster_emr>
  <shard>
    <replica>
      <host>host1</host>
      <port>port1</port>
      <user>default</user>
      <password>${users.default.password}</password>
    </replica>
    ...
  </shard>
  ...
</cluster_emr>
```

> **说明** 代码示例中的 `${users.default.password}` 为users.default.password的参数值，即default用户的密码。

如何修改ClickHouse集群标识？

您可以在EMR控制台ClickHouse服务的**配置**页面，在搜索区域搜索**clickhouse_remote_servers**参数，将默认配置中的cluster_emr修改为需要的名称。例如，修改cluster_emr为new_cluster_name，修改信息如下：

- 修改前

```
<cluster_emr>
  <shard>
    ...
  </shard>
  ...
</cluster_emr>
```

- 修改后

```
<new_cluster_emr>
  <shard>
    ...
  </shard>
  ...
</new_cluster_emr>
```

如何设置ClickHouse多盘存储？

> **注意** EMR-3.39.x之前版本和EMR-5.5.x之前版本，请按照以下操作设置ClickHouse多盘存储。EMR-3.39.x及之后版本，EMR-5.5.x及之后版本无需手动设置，默认多盘存储。

ClickHouse默认数据存储策略是存储在一块磁盘上，如果您希望将数据存储到所有机器的磁盘上，请按照以下方法操作：

- 方式一：在建表语句中指定storage_policy参数。

  创建表时需要配置存储策略，代码示例如下。

```
CREATE TABLE db (
)ENGINE = <your_MergeTree>
SETTINGS storage_policy = '<your_storage_policy>';
```

- <your_storage_policy>：您可以在EMR控制台ClickHouse服务的**配置**页面，在搜索区域搜索**storage_configuration**参数，本文示例中**policies**层级下的hdd_in_order即为参数值。

  > **注意** 参数值请填写为您实际查询到的。

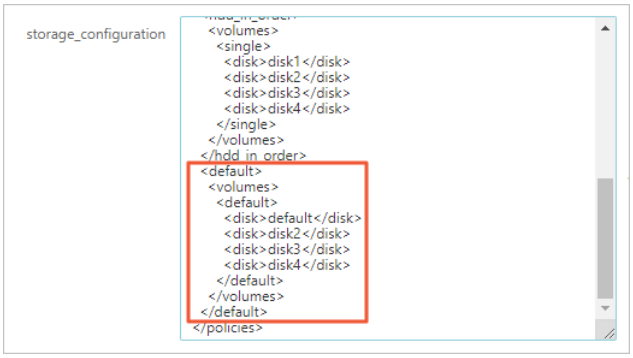
- <your_MergeTree>：引擎名和参数。根据您实际需求自定义，引擎（ENGINE）详细信息，请参见MergeTree。

- 方式二：在EMR控制台新增default策略。

  i. 在EMR控制台ClickHouse服务的**配置**页面，在配置搜索区域搜索**storage_configuration**参数，在**storage_configuration**中新增一个default策略，新增内容如下。

  ```
  <default>
     <volumes>
       <default>
         <disk>default</disk>
         <disk>disk2</disk>
         <disk>disk3</disk>
         <disk>disk4</disk>
       </default>
     </volumes>
   </default>
  ```

  修改后参数内容如下图所示。

  ii. 完成上述配置后，重启服务，详情请参见重启服务。

数据丢失，如何处理？

- 问题现象：向ClickHouse写入A条数据，但实际读出来只有B条，且B小于A。
- 问题原因：通常情况下，ClickHouse中的数据不会丢失。但是在shard内存在至少两个副本，本地表的表引擎为*MergeTree，同时使用分布式表进行读数的场景下，可能会出现读出来的数据少于写入数据的情况。

  无论是通过分布式表写数据，还是直接写入本地表，每个机器上都会存在数据；而通过分布式表读数据时，默认每个shard仅会使用一个连接，此时的连接数是会少于机器数的，所以存在一些机器上的数据无法被读取的情况。代码示例如下。

```
CREATE TABLE db.table_local
(
  ...
)
Engine = MergeTree()
CREATE TABLE db.table_distributed
(
  ...
)
Engine = Distributed(cluster_emr, db, table_local, rand());
```

代码示例中cluster_emr参数的配置形式如下。

```
<clickhouse_remote_servers>
  <cluster_emr>
    <shard>
       <replica>
         <host>host1</host>
         </port>port2</port>
       </replica>
       <replica>
         <host>host2</host>
         </port>port2</port>
       </replica>
     </shard>
     <shard>
       <replica>
         <host>host3</host>
         </port>port3</port>
       </replica>
       <replica>
         <host>host4</host>
         </port>port4</port>
       </replica>
     </shard>
  </cluster_emr>
</clickhouse_remote_servers>
```

- 处理方法：

方式	操作
方式一（推荐）	删除*db.table_local*并后重新创建表，使用复制表作为本地表。
方式二	您可以在EMR控制台ClickHouse服务的**配置**页面，单击**server-metrika**页签，修改clickhouse_remote_servers的参数值，修改为每个shard下一个replica。
方式三（不推荐）	您可以在EMR控制台ClickHouse服务的**配置**页面，单击**server-users**页签，单击右上角的**自定义配置**，新增参数profiles.<your_profile_name>.max_parallel_replicas，参数值至少为每个shard下replica的数量，并确保users.<your_clickhouse-client_name>.profile值为*<your_profile_name>*。 **说明** *<your_profile_name>*和*<your_clickhouse-client_name>*均需要替换为您实际的名称。各参数详细信息，请参见访问权限控制。

如果以上方法还是无法解决您的问题，请提交工单或购买专家服务处理。

报错提示Memory limit (for total) exceeded时，该如何处理？

- 问题原因：内存超过了server可使用的总内存。
- 处理方法：在EMR控制台ClickHouse服务的**配置**页面，单击**server-config**页签，单击右上角的**自定义配置**，新增参数max_server_memory_usage，该参数可以配置的最大值为 **机器物理内存大小** * max_server_memory_usage_to_ram_ratio 。

说明 ClickHouse中max_server_memory_usage_to_ram_ratio参数的默认值为0.9，如果您需要调整该参数值，可以新增max_server_memory_usage_to_ram_ratio参数，参数值您可以根据实际情况调整。

报错提示Memory limit (for query) exceeded时，该如何处理？

- 问题原因：内存超过了单次Query可使用的最大内存。
- 处理方法：

场景	操作
全局方式	在EMR控制台ClickHouse服务的**配置**页面，单击**server-config**页签，单击右上角的**自定义配置**，新增参数profiles.<your_profile_name>.max_memory_usage，并确保users.<your_clickhouse-client_name>.profile值为*<your_profile_name>*。 **说明** *<your_profile_name>*和*<your_clickhouse-client_name>*均需要替换为您实际的名称。各参数详细信息，请参见访问权限控制。
针对使用clickhouse-client	在EMR控制台ClickHouse服务的**配置**页面，单击**client-config**页签，单击右上角的**自定义配置**，新增参数max_memory_usage。
针对某一次会话Session	可以直接`SET max_memory_usage=xxxx`，该配置在Session生命周期内均会生效。
针对某一次Query	可以在SQL中添加配置，该配置仅对当前Query生效。 例如，`SELECT column FROM table SETTINGS max_memory_usage=xxxx`。

报错提示Memory limit (for user) exceeded时，该如何处理？

- 问题原因：内存超过了单个用户可使用的最大内存。
- 处理方法：

场景	操作
全局方式	在EMR控制台ClickHouse服务的**配置**页面，单击**server-users**页签，单击右上角的**自定义配置**，新增参数profiles.<your_profile_name>.max_memory_usage_for_user，并确保users.<your_clickhouse-client_name>.profile值为*<your_profile_name>*。 **说明** *<your_profile_name>*和*<your_clickhouse-client_name>*均需要替换为您实际的名称。各参数详细信息，请参见访问权限控制。
针对使用clickhouse-client	在EMR控制台ClickHouse服务的**配置**页面，单击**client-config**页签，单击右上角的**自定义配置**，新增参数max_memory_usage_for_user。
针对某一次会话Session	可以直接`SET max_memory_usage_for_user=xxxx`，该配置在Session生命周期内均会生效。
针对某一次Query	可以在SQL中添加配置，该配置仅对当前Query生效。 例如，`SELECT column FROM table SETTINGS max_memory_usage_for_user=xxxx`。

如何扩缩容磁盘？

- 扩容磁盘的详细信息，请参见扩容磁盘。
- 缩容磁盘的详细信息，请参见缩容磁盘。

报错提示Too many parts in all partitions in total，该如何处理？

在插入数据时，ClickHouse会对表中的data parts进行检查，如果data parts的数量超过了阈值，则会报错。ClickHouse中包含了多种对data parts进行检查的阈值。需要根据详细的提示信息判断并处理。

该问题是由于一个表的所有活跃的data parts数量超过了max_parts_in_total的参数值导致的，该参数的默认值为100000。该问题的解决方法如下：

- 一个表下可能存在着若干个partition，每个partition下的活跃data parts受限于参数parts_to_throw_insert，请确保一个表下的partition数量乘以parts_to_throw_insert的值小于max_parts_in_total。您可以根据业务需求，对没有设置TTL（Time To Live）的partition设置TTL，或调小TTL。
- 对于整个表执行`optimize table $<db>.$<table> final`命令，以减少未合并的data parts数量。
- 调整写入的客户端。例如，利用JDBC写入数据的时候，降低写入数据的频次，提高每一批次写入数据的大小。在客户端和业务允许的范围内，每一批次的数据量越大，越能够降低出现这一问题的几率，直到达到硬件瓶颈。
- 如果已经是低频大批次的写入但仍然超过了这一阈值max_parts_in_total，且机器的CPU、IOUtil等指标达到瓶颈，可以考虑升配或扩容操作。
- 调整参数延缓插入的速度：
  - 进入EMR控制台ClickHouse服务的配置页面，在**server-config**页签中，添加自定义参数**merge_tree.parts_to_delay_insert**，该参数默认值为150，您可以调小该参数值为100或50。
  - 进入EMR控制台ClickHouse服务的配置页面，在**server-config**页签中，添加自定义参数merge_tree.max_delay_to_insert ，该参数的默认值为1，表示1秒，您可以调大该参数值为2或5。

- 进入EMR控制台ClickHouse服务的配置页面，在**server-config**页签中，添加自定义参数**merge_tree.max_parts_in_total**，该参数的默认值为100000，您可以根据实际情况调大该参数值。

  > 警告 使用此方法可能会导致集群的性能和稳定性下降，请谨慎操作。

报错提示Too many parts，该如何处理？

在插入数据时，ClickHouse会对表中的data parts进行检查，如果data parts的数量超过了阈值，则会报错。ClickHouse中包含了多种对data parts进行检查的阈值。需要根据详细的提示信息判断并处理。

如果too many parts信息中没有详情的错误信息，则通常是超过了partition的活跃data parts数量的阈值parts_to_throw_insert，默认值为300。解决方法如下：

- 调整写入的客户端。例如，利用JDBC写入数据的时候，降低写入数据的频次，提高每一批次写入数据的大小。在客户端和业务允许的范围内，每一批次的数据量越大，越能够降低出现这一问题的几率，直到达到硬件瓶颈。
- 如果已经是低频大批次的写入但仍然超过了这一阈值parts_to_throw_insert，且机器的CPU、IOUtil等指标达到瓶颈，可以考虑升配或扩容操作。
- 调整参数延缓插入的速度：
  - 进入EMR控制台ClickHouse服务的配置页面，在**server-config**页签中，添加自定义参数**merge_tree.parts_to_delay_insert**，该参数的默认值为150，您可以调小该参数值为100或50。
  - 进入EMR控制台ClickHouse服务的配置页面，在**server-config**页签中，添加自定义参数merge_tree.max_delay_to_insert ，该参数的默认值为1，表示1秒，您可以调大该参数值为2或5。
- 进入EMR控制台ClickHouse服务的配置页面，在**server-config**页签中，添加自定义参数**merge_tree.parts_to_throw_insert**，该参数的默认值为300，您可以调大该参数值。

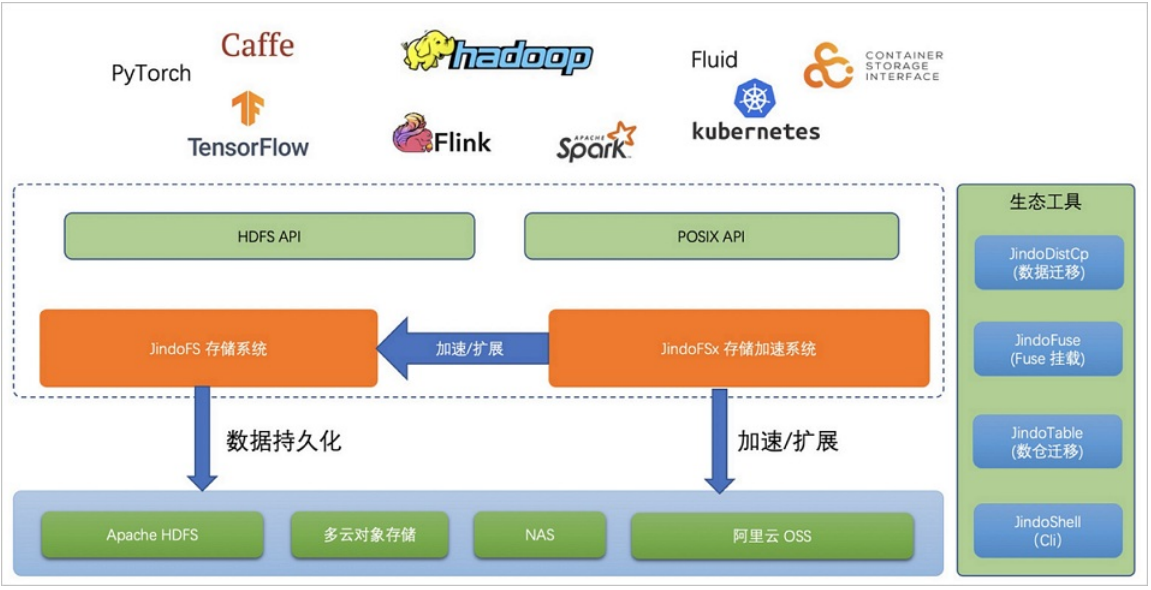
  > 警告 使用此方法可能会导致集群的性能和稳定性下降，请谨慎操作。

6.4. JindoData

6.4.1. JindoData概述

JindoData是阿里云开源大数据团队自研的数据湖存储加速套件，面向大数据和AI生态，为阿里云和业界主要数据湖存储系统提供全方位访问加速解决方案。

JindoData套件基于统一架构和内核实现，主要包括JindoFS存储系统（原JindoFS Block模式）、JindoFSx存储加速系统（原JindoFS Cache模式），JindoSDK大数据万能SDK和全面兼容的生态工具（JindoFuse、JindoDistCp）以及插件支持。



JindoData主要组件如下：

- JindoFS存储系统
- JindoFSx存储加速系统
- 生态支持和工具

JindoFS存储系统

基于阿里云OSS的云原生存储系统，二进制兼容Apache HDFS，并且与Apache HDFS基本功能对齐，提供优化的HDFS使用和平迁体验。JindoFS存储系统是原JindoFS Block模式的全新升级版本。

阿里云OSS-HDFS服务（JindoFS服务）是JindoFS存储系统在阿里云上的服务化部署形态，和阿里云OSS深度融合，开箱即用，无须在自建集群部署维护JindoFS，即免运维。

OSS-HDFS服务的详细信息，请参见OSS-HDFS服务概述。

JindoFSx存储加速系统

JindoFSx（JindoData服务）是原JindoFS Cache模式的全新升级版本，是面向大数据和AI生态的云原生数据湖存储加速系统，为大数据和AI应用访问各种云存储提供访问加速，支持数据缓存、元数据缓存和P2P加速等功能。JindoFSx支持管理多个后端存储系统，可以通过统一命名空间进行管理，也可以兼容各系统原生的访问协议，也支持为这些系统提供统一的权限管理。原生优化支持阿里云OSS和阿里云OSS-HDFS服务，同时也支持业界多云对象存储（例如，AWS S3）、 Apache HDFS和NAS。

生态支持和工具

- 支持JindoSDK。

  支持面向云时代的大数据Hadoop SDK和HDFS接口，内置优化访问阿里云OSS，较Hadoop社区版本性能大幅提升。同时支持JindoFS存储系统和服务、JindoFSx存储加速系统，支持多云对象存储。

- 支持JindoShell CLI。

  JindoData除了支持HDFS Shell命令，还提供了一套JindoShell CLI命令，从功能、性能上大幅扩展和优化一些数据访问操作。

- 支持JindoFuse POSIX。

  JindoData为阿里云OSS、JindoFS存储系统和服务、JindoFSx存储加速系统提供了POSIX支持。

- 支持JindoDistCp数据迁移。

  IDC机房数据（HDFS）上云迁移和多云迁移利器，支持多种存储数据迁移到阿里云OSS和JindoFS服务，使用上类似Hadoop DistCp。

- 支持JindoTable。

  结合计算引擎的使用推出的一套解决方案，支持Spark、Hive和Presto等引擎，以及表格式数据的管理功能。

- 生态插件。

  除了默认提供JindoSDK支持Hadoop，另外还支持Flink Connector等插件。

6.4.2. JindoData 4.3.0

6.4.2.1. JindoData 4.3.0版本简介

JindoData 4.x是阿里云E-MapReduce产品SmartData自研组件（SmartData 3.8.0版本）架构升级之后首次发布的版本，重点对接和支持了阿里云OSS存储产品和阿里云OSS-HDFS服务（JindoFS服务）。本文为您介绍JindoData 4.3.0版本支持的功能。

功能介绍

JindoData是阿里云开源大数据团队自研的数据湖存储加速套件，面向大数据和AI生态，为阿里云和业界主要数据湖存储系统提供全方位访问加速解决方案。

JindoSDK和工具支持

- JindoSDK支持多云存储，包括Amazon S3、COS和OBS。
- JindoSDK提供JindoTable工具。
- JindoSDK优化了Flink Connector插件。
- JindoSDK完善了JindoDistCp。

JindoFSx存储加速系统

- JindoFSx支持多云存储，包括Amazon S3、COS和OBS。
- JindoFSx优化了数据缓存及元数据缓存。
- JindoFSx支持Kerberos + Ranger的鉴权方案。
- JindoFSx大幅完善了可观测性指标。
- JindoFSx完成与Fluid的对接。

JindoFS存储系统

- JindoFS支持POSIX Lock和Fallocate能力。
- JindoFS支持老版本JindoFS Block模式集群升级。

JindoFuse POSIX支持

- JindoFuse新增XAttr相关接口支持，包括setxattr、getxattr、listxattr和removexattr。

- JindoFuse支持POSIX Lock和Fallocate能力。
- JindoFuse支持OSS可追加写对象，包括append、flush和边写边读功能。

6.4.2.2. 基础功能

6.4.2.2.1. 阿里云OSS透明缓存加速

JindoFSx存储加速系统提供了透明缓存的使用方式，兼容原生OSS存储方式，文件以对象的形式存储在OSS上，每个文件根据实际访问情况会在本地进行缓存，提升访问OSS的效率，同时兼容了原有OSS文件形式，数据访问上能够与其他OSS客户端完全兼容，作业访问OSS的方式无需做任何修改。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> **说明** 本文以EMR-3.40.0版本为例介绍。

使用限制

该功能适用的集群版本：EMR-3.40.0及后续版本、EMR-5.6.0及后续版本。

操作流程

1. 步骤一：配置AccessKey
2. 步骤二：配置JindoSDK
3. 步骤三：磁盘空间水位控制

步骤一：配置AccessKey

1. 进入新增配置项页面。
    i. 进入JindoData服务页面。
        a. 登录阿里云E-MapReduce控制台。
        b. 在顶部菜单栏处，根据实际情况选择地域和资源组。
        c. 单击上方的**集群管理**页签。
        d. 在**集群管理**页面，单击相应集群所在行的**详情**。
        e. 在左侧导航栏，选择**集群服务 > JindoData**。
    ii. 在JindoData服务页面，单击**配置**页签。
    iii. 在**服务配置**区域，单击**common**页签。
2. 新增配置。
    i. 单击**自定义配置**。

ii. 在**新增配置项**对话框中，新增以下配置项。

新增配置项的具体操作，请参见添加组件参数。

- 全局方式配置（所有Bucket使用同一种方式）

参数	描述
jindofsx.oss.accessKeyId	OSS的AccessKey ID。
jindofsx.oss.accessKeySecret	OSS的AccessKey Secret。
jindofsx.oss.endpoint	OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。

- 按照Bucket配置

参数	描述
jindofsx.oss.bucket.XXX.accessKeyId	`XXX`的Bucket的AccessKey ID。
jindofsx.oss.bucket.XXX.accessKeySecret	`XXX`的Bucket的AccessKey Secret。
jindofsx.oss.bucket.XXX.endpoint	`XXX`的Bucket的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
jindofsx.oss.bucket.YYY.accessKeyId	`YYY`的Bucket的AccessKey ID。
jindofsx.oss.bucket.YYY.accessKeySecret	`YYY`的Bucket的AccessKey Secret。
jindofsx.oss.bucket.YYY.endpoint	`YYY`的Bucket的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。

> **说明** `XXX`和`YYY`为OSS Bucket的名称。

iii. 单击**确定**。

3. 重启服务。
   i. 在JindoData服务页面，选择右上角的**操作 > 重启All Components**。
   ii. 在**执行集群操作**对话框中，输入执行原因（其他参数保持默认），单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

步骤二：配置JindoSDK

> **注意** 此配置为客户端配置，无需重启JindoData服务。

1. 进入配置页面。
   i. 进入HDFS服务页面。
      a. 登录EMR on ECS控制台。
      b. 在顶部菜单栏处，根据实际情况选择地域和资源组。
      c. 单击上方的**集群管理**页签。
      d. 在**集群管理**页面，单击相应集群所在行的**详情**。
      e. 在左侧导航栏，选择**集群服务 > HDFS**。
   ii. 在HDFS服务页面，单击**配置**页签。
   iii. 在**服务配置**区域，单击**core-site**页签。
2. 修改以下配置项。

   修改配置项的具体操作，请参见修改组件参数。

内容	参数	描述
配置OSS实现类	fs.AbstractFileSystem.oss.impl	固定值为com.aliyun.jindodata.oss.OSS。
	fs.oss.impl	固定值为com.aliyun.jindodata.oss.JindoOssFileSystem。
	fs.xengine	固定值为jindofsx。

内容	参数	描述
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 说明 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
启用缓存加速功能	fs.jindofsx.data.cache.enable	数据缓存开关： false（默认值）：禁用数据缓存。 true：启用数据缓存。 说明 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。

3. 保存配置。
   i. 单击**服务配置**区域的**保存**。
   ii. 在**确认修改**对话框中，输入执行原因，开启**自动更新配置**，单击**确定**。
4. 在**core-site**页签，新增配置项。
   i. 单击**服务配置**区域的**自定义配置**。
   ii. 在**新增配置项**对话框中，新增以下配置项。
   新增配置项的具体操作，请参见添加组件参数。

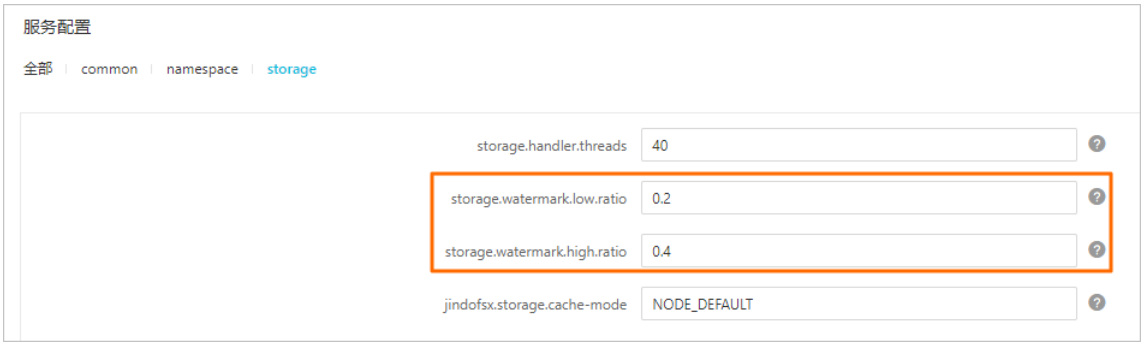
内容	参数	描述
配置Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
其他缓存加速功能	fs.jindofsx.meta.cache.enable	元数据缓存开关： false（默认值）：禁用元数据缓存。 true：启用元数据缓存。
	fs.jindofsx.slice.cache.enable	小文件缓存优化开关： false（默认值）：禁用小文件缓存。 true：启用小文件缓存。
	fs.jindofsx.ram.cache.enable	内存缓存开关： false（默认值）：禁用内存缓存。 true：启用内存缓存。
	fs.jindofsx.short.circuit.enable	短路读开关： true（默认值）：打开短路读开关。 false：关闭短路读开关。

   iii. 单击**确定**。

步骤三：磁盘空间水位控制

缓存启用后，JindoFSx服务会自动管理本地缓存备份，通过水位清理本地缓存，请您根据需求配置一定的比例用于缓存。JindoFSx后端基于OSS，可以提供海量的存储，但是本地盘的容量是有限的，因此JindoFSx会自动淘汰本地较冷的数据备份。您可以通过修改**storage.watermark.high.ratio**和**storage.watermark.low.ratio**两个参数来调节本地存储的使用容量，值均为0～1的小数，表示使用磁盘空间的比例。

1. 修改磁盘水位配置。
   在JindoData服务**配置**页面的**服务配置**区域，单击**storage**页签，修改以下参数。



参数	描述
storage.watermark.low.ratio	表示使用量的下水位比例，触发清理后会自动清理冷数据，将JindoFS数据目录占用空间清理到下水位。默认值：0.2。
storage.watermark.high.ratio	表示磁盘使用量的上水位比例，每块数据盘的JindoFS数据目录占用的磁盘空间到达上水位即会触发清理。默认值：0.4。

> **说明** 修改该参数时，下水位必须小于上水位，设置合理的值即可。

2. 保存配置。
   i. 单击**服务配置**区域的**保存**。
   ii. 在**确认修改**对话框中，输入执行原因，开启**自动更新配置**，单击**确定**。
3. 重启服务。
   i. 在JindoData服务页面，选择右上角的**操作 > 重启All Components**。
   ii. 在**执行集群操作**对话框中，输入执行原因（其他参数保持默认），单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

相关文档

JindoSDK包含一些高级调优参数，配置方式以及配置项的详细信息，请参见JindoSDK高级参数配置。

6.4.2.2.2. 阿里云OSS统一挂载缓存加速

本文为您介绍JindoFSx支持阿里云OSS统一挂载缓存加速的使用方式。

背景信息

JindoSDK包含一些高级调优参数，配置方式以及配置项请参见JindoSDK高级参数配置。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> **说明** 本文以EMR-3.40.0版本为例介绍。

操作流程

1. 步骤一：配置AccessKey
2. 步骤二：配置JindoSDK
3. 步骤三：磁盘空间水位控制
4. 步骤四：挂载阿里云OSS
5. 步骤五：访问阿里云OSS

步骤一：配置AccessKey

1. 进入新增配置项页面。

i. 进入JindoData服务页面。
   a. 登录阿里云E-MapReduce控制台。
   b. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   c. 单击上方的**集群管理**页签。
   d. 在**集群管理**页面，单击相应集群所在行的**详情**。
   e. 在左侧导航栏，选择**集群服务 > JindoData**。
ii. 在JindoData服务页面，单击**配置**页签。
iii. 在**服务配置**区域，单击**common**页签。

2. 新增配置。
   i. 单击**自定义配置**。
   ii. 在**新增配置项**对话框中，新增以下配置项。
   新增配置项的具体操作，请参见添加组件参数。
   - 全局方式配置（所有Bucket使用同一种方式）

参数	描述
jindofsx.oss.accessKeyId	OSS的AccessKey ID。
jindofsx.oss.accessKeySecret	OSS的AccessKey Secret。
jindofsx.oss.endpoint	OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。

   - 按照Bucket配置

参数	描述
jindofsx.oss.bucket.XXX.accessKeyId	`XXX`的Bucket的AccessKey ID。
jindofsx.oss.bucket.XXX.accessKeySecret	`XXX`的Bucket的AccessKey Secret。
jindofsx.oss.bucket.XXX.endpoint	`XXX`的Bucket的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
jindofsx.oss.bucket.YYY.accessKeyId	`YYY`的Bucket的AccessKey ID。
jindofsx.oss.bucket.YYY.accessKeySecret	`YYY`的Bucket的AccessKey Secret。
jindofsx.oss.bucket.YYY.endpoint	`YYY`的Bucket的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。

   > **说明** `XXX`和`YYY`为OSS Bucket的名称。

   iii. 单击**确定**。
3. 重启服务。
   i. 在JindoData服务页面，选择右上角的**操作 > 重启All Components**。
   ii. 在**执行集群操作**对话框中，输入执行原因（其他参数保持默认），单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

步骤二：配置JindoSDK

> **注意** 此配置为客户端配置，无需重启JindoData服务。

1. 进入配置页面。
   i. 进入HDFS服务页面。
      a. 登录EMR on ECS控制台。
      b. 在顶部菜单栏处，根据实际情况选择地域和资源组。
      c. 单击上方的**集群管理**页签。
      d. 在**集群管理**页面，单击相应集群所在行的**详情**。
      e. 在左侧导航栏，选择**集群服务 > HDFS**。
   ii. 在HDFS服务页面，单击**配置**页签。
   iii. 在**服务配置**区域，单击**core-site**页签。

2. 修改以下配置项。

   修改配置项的具体操作，请参见修改组件参数。

内容	参数	描述
配置JINDO实现类	fs.AbstractFileSystem.jindo.impl	固定值为com.aliyun.jindodata.jindo.JINDO。
	fs.jindo.impl	固定值为com.aliyun.jindodata.jindo.JindoFileSystem。
	fs.xengine	固定值为jindofsx。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 说明 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
启用缓存加速功能	fs.jindofsx.data.cache.enable	数据缓存开关： false（默认值）：禁用数据缓存。 true：启用数据缓存。 说明 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。

3. 保存配置。
   i. 单击**服务配置**区域的**保存**。
   ii. 在**确认修改**对话框中，输入执行原因，开启**自动更新配置**，单击**确定**。
4. 在**core-site**页签，新增配置项。
   i. 单击**服务配置**区域的**自定义配置**。
   ii. 在**新增配置项**对话框中，新增以下配置项。

   新增配置项的具体操作，请参见添加组件参数。

内容	参数	描述
配置Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
其他缓存加速功能	fs.jindofsx.meta.cache.enable	元数据缓存开关： false（默认值）：禁用元数据缓存。 true：启用元数据缓存。
	fs.jindofsx.slice.cache.enable	小文件缓存优化开关： false（默认值）：禁用小文件缓存。 true：启用小文件缓存。
	fs.jindofsx.ram.cache.enable	内存缓存开关： false（默认值）：禁用内存缓存。 true：启用内存缓存。
	fs.jindofsx.short.circuit.enable	短路读开关： true（默认值）：打开短路读开关。 false：关闭短路读开关。

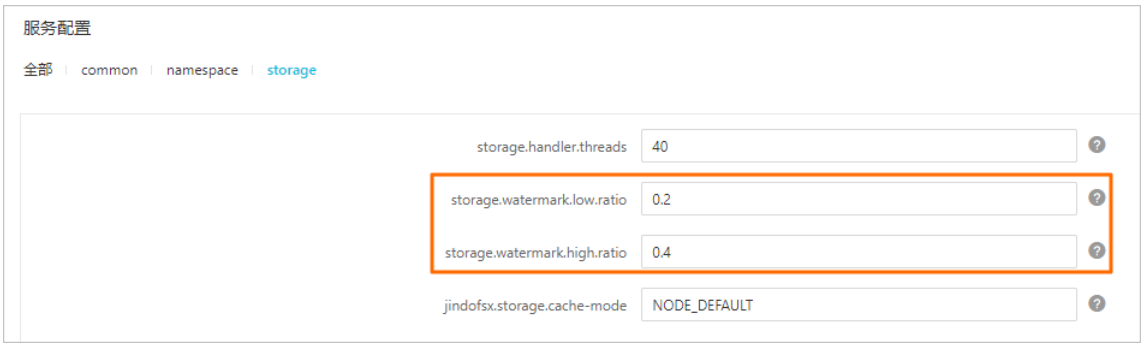
   iii. 单击**确定**。

步骤三：磁盘空间水位控制

缓存启用后，JindoFSx服务会自动管理本地缓存备份，通过水位清理本地缓存，请您根据需求配置一定的比例用于缓存。JindoFSx后端基于OSS，可以提供海量的存储，但是本地盘的容量是有限的，因此JindoFSx会自动淘汰本地较冷的数据备份。您可以通过修改**storage.watermark.high.ratio**和**storage.watermark.low.ratio**两个参数来调节本地存储的使用容量，值均为0～1的小数，表示使用磁盘空间的比例。

1. 修改磁盘水位配置。

   在JindoData服务**配置**页面的**服务配置**区域，单击**storage**页签，修改以下参数。

参数	描述
storage.watermark.low.ratio	表示使用量的下水位比例，触发清理后会自动清理冷数据，将JindoFS数据目录占用空间清理到下水位。默认值：0.2。
storage.watermark.high.ratio	表示磁盘使用量的上水位比例，每块数据盘的JindoFS数据目录占用的磁盘空间到达上水位即会触发清理。默认值：0.4。

   > **说明** 修改该参数时，下水位必须小于上水位，设置合理的值即可。

2. 保存配置。
   i. 单击**服务配置**区域的**保存**。
   ii. 在**确认修改**对话框中，输入执行原因，开启**自动更新配置**，单击**确定**。
3. 重启服务。
   i. 在JindoData服务页面，选择右上角的**操作 > 重启All Components**。
   ii. 在**执行集群操作**对话框中，输入执行原因（其他参数保持默认），单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

步骤四：挂载阿里云OSS

您可以执行以下命令，挂载OSS目录：

- 语法

```
jindo admin -mount <path> <realpath>
```

- 示例

```
jindo admin -mount /jindooss oss://<yourBucketName>/
```

执行以下命令，返回信息为 `jindo://emr-header-1:8101/jindooss` 。

```
hdfs dfs -ls jindo://emr-header-1:8101/
```

即访问 `jindo://emr-header-1:8101/jindooss/` 等价于访问 `oss://<yourBucketName>/` 。

步骤五：访问阿里云OSS

您通过 `jindo://` 前缀读取OSS上的数据后，在数据缓存开关打开时，会自动缓存到JindoFSx存储加速系统中，后续通过 `jindo://` 访问相同的数据就能够命中缓存。

6.4.2.2.3. 阿里云OSS-HDFS服务透明缓存加速

本文为您介绍JindoFSx支持OSS-HDFS服务（JindoFS服务）透明缓存加速的使用方式。

背景信息

OSS-HDFS服务的详细信息，请参见OSS-HDFS服务概述。

前提条件

- 已在E-MapReduce上创建EMR-3.40.0及后续版本的集群，具体操作请参见创建集群。

  > 注意 本文以EMR-3.40.0版本为例介绍。

- 已开通并授权访问OSS-HDFS服务，具体操作请参见开通并授权访问OSS-HDFS服务。

操作流程

1. 步骤一：配置OSS-HDFS AccessKey
2. 步骤二：配置JindoSDK
3. 步骤三：磁盘空间水位控制

步骤一：配置OSS-HDFS AccessKey

1. 进入新增配置项页面。
   i. 进入JindoData服务页面。
      a. 登录阿里云E-MapReduce控制台。
      b. 在顶部菜单栏处，根据实际情况选择地域和资源组。
      c. 单击上方的**集群管理**页签。
      d. 在**集群管理**页面，单击相应集群所在行的**详情**。
      e. 在左侧导航栏，选择**集群服务 > JindoData**。
   ii. 在JindoData服务页面，单击**配置**页签。
   iii. 在**服务配置**区域，单击**common**页签。
2. 在**新增配置项**对话框中，新增以下配置项，单击**确定**。
   i. 单击**自定义配置**。
   ii. 在**新增配置项**对话框中，新增以下配置项。

   新增配置项的具体操作，请参见添加组件参数。

   - 全局方式配置（所有Bucket使用同一种方式）

参数	描述
jindofsx.oss.accessKeyId	OSS的AccessKey ID。
jindofsx.oss.accessKeySecret	OSS的AccessKey Secret。
jindofsx.oss.endpoint	OSS-HDFS服务的Endpoint。例如，cn-***.oss-dls.aliyuncs.com。

   - 按照Bucket配置

参数	描述
jindofsx.oss.bucket.XXX.accessKeyId	`XXX`的Bucket的AccessKey ID。
jindofsx.oss.bucket.XXX.accessKeySecret	`XXX`的Bucket的AccessKey Secret。
jindofsx.oss.bucket.XXX.endpoint	`XXX`的Bucket的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
jindofsx.oss.bucket.YYY.accessKeyId	`YYY`的Bucket的AccessKey ID。
jindofsx.oss.bucket.YYY.accessKeySecret	`YYY`的Bucket的AccessKey Secret。
jindofsx.oss.bucket.YYY.endpoint	`YYY`的Bucket的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
jindofsx.oss.bucket.YYY.user	Storage Service访问OSS-HDFS服务使用的用户名。

     > 说明 `XXX`和`YYY`为OSS Bucket的名称。

   iii. 单击**确定**。
3. 重启服务。
   i. 在JindoData服务页面，选择右上角的**操作 > 重启All Components**。

ii. 在**执行集群操作**对话框中，输入执行原因（其他参数保持默认），单击**确定**。

iii. 在**确认**对话框中，单击**确定**。

步骤二：配置JindoSDK

> 注意 此配置为客户端配置，无需重启JindoData服务。

1. 进入配置页面。
   i. 进入HDFS服务页面。
      a. 登录EMR on ECS控制台。
      b. 在顶部菜单栏处，根据实际情况选择地域和资源组。
      c. 单击上方的**集群管理**页签。
      d. 在**集群管理**页面，单击相应集群所在行的**详情**。
      e. 在左侧导航栏，选择**集群服务 > HDFS**。
   ii. 在HDFS服务页面，单击**配置**页签。
   iii. 在**服务配置**区域，单击**core-site**页签。
2. 修改以下配置项。

   修改配置项的具体操作，请参见修改组件参数。

内容	参数	描述
配置OSS实现类	fs.AbstractFileSystem.oss.impl	固定值为com.aliyun.jindodata.oss.OSS。
	fs.oss.impl	固定值为com.aliyun.jindodata.oss.JindoOssFileSystem。
	fs.xengine	固定值为jindofsx。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
启用缓存加速功能	fs.jindofsx.data.cache.enable	数据缓存开关： ○ false（默认值）：禁用数据缓存。 ○ true：启用数据缓存。 **说明** 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。

3. 保存配置。
   i. 单击**服务配置**区域的**保存**。
   ii. 在**确认修改配置**对话框中，输入执行原因，开启**自动更新配置**，单击**确定**。
4. 在**core-site**页签，新增配置项。
   i. 单击**服务配置**区域的**自定义配置**。

ii. 在**新增配置项**对话框中，新增以下配置项。

新增配置项的具体操作，请参见添加组件参数。

内容	参数	描述
配置Accesskey	fs.oss.accessKeyId	OSS的Bucket的AccessKey ID。
	fs.oss.accessKeySecret	OSS的Bucket的AccessKey Secret。
	fs.oss.endpoint	OSS-HDFS服务的Endpoint。格式为oss://<yourBucketName>.<yourBucketEndpoint>/<yourBucketObject>。 例如，`oss://mydlsbucket.cn-shanghai.oss-dls.aliyuncs.com/Test`。
其他缓存加速	fs.jindofsx.meta.cache.enable	元数据缓存开关： ■ false（默认值）：禁用元数据缓存。 ■ true：启用元数据缓存。
	fs.jindofsx.slice.cache.enable	小文件缓存优化开关： ■ false（默认值）：禁用小文件缓存。 ■ true：启用小文件缓存。
	fs.jindofsx.ram.cache.enable	内存缓存开关： ■ false（默认值）：禁用内存缓存。 ■ true：启用内存缓存。
	fs.jindofsx.short.circuit.enable	短路读开关： ■ true（默认值）：打开短路读开关。 ■ false：关闭短路读开关。

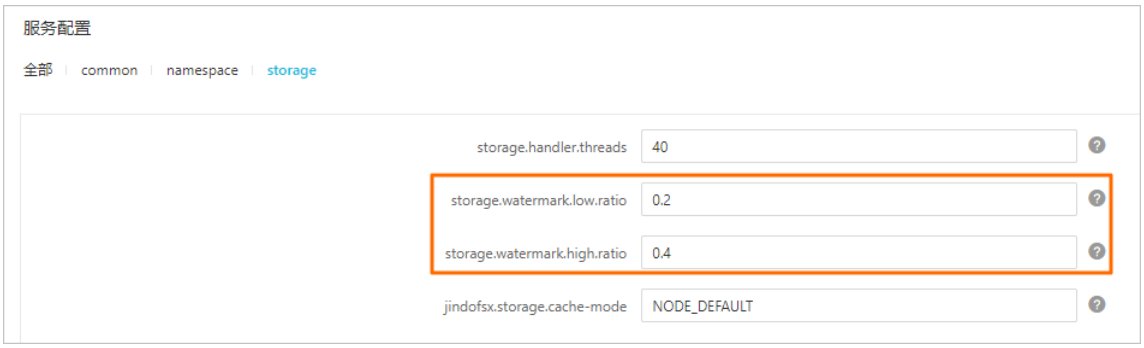
iii. 单击**确定**。

步骤三：磁盘空间水位控制

缓存启用后，JindoFSx服务会自动管理本地缓存备份，通过水位清理本地缓存，请您根据需求配置一定的比例用于缓存。JindoFSx后端基于OSS，可以提供海量的存储，但是本地盘的容量是有限的，因此JindoFSx会自动淘汰本地较冷的数据备份。您可以通过修改**storage.watermark.high.ratio**和**storage.watermark.low.ratio**两个参数来调节本地存储的使用容量，值均为0～1的小数，表示使用磁盘空间的比例。

1. 修改磁盘水位配置。

在JindoData服务**配置**页面的**服务配置**区域，单击**storage**页签，修改以下参数。



参数	描述
storage.watermark.low.ratio	表示使用量的下水位比例，触发清理后会自动清理冷数据，将JindoFS数据目录占用空间清理到下水位。默认值：0.2。
storage.watermark.high.ratio	表示磁盘使用量的上水位比例，每块数据盘的JindoFS数据目录占用的磁盘空间到达上水位即会触发清理。默认值：0.4。

> **说明** 修改该参数时，下水位必须小于上水位，设置合理的值即可。

2. 保存配置。
    i. 单击**服务配置**区域的**保存**。
    ii. 在**确认修改**对话框中，输入执行原因，开启**自动更新配置**，单击**确定**。
3. 重启服务。
    i. 在JindoData服务页面，选择右上角的**操作 > 重启All Components**。
    ii. 在**执行集群操作**对话框中，输入执行原因（其他参数保持默认），单击**确定**。
    iii. 在**确认**对话框中，单击**确定**。

6.4.2.2.4. 阿里云OSS-HDFS服务统一挂载缓存加速

本文为您介绍JindoFSx支持OSS-HDFS服务（JindoFS服务）统一挂载缓存加速的使用方式。

前提条件

- 已在E-MapReduce上创建EMR-3.40.0及后续版本的集群，具体操作请参见创建集群。

> 注意 本文以EMR-3.40.0版本为例介绍。

- 已开通并授权访问OSS-HDFS服务，具体操作请参见开通并授权访问OSS-HDFS服务。

操作流程

1. 步骤一：配置OSS-HDFS AccessKey
2. 步骤二：配置JindoSDK
3. 步骤三：磁盘空间水位控制
4. 步骤四：挂载OSS-HDFS服务
5. 步骤五：访问阿里云OSS

步骤一：配置OSS-HDFS AccessKey

1. 进入新增配置项页面。
    i. 进入JindoData服务页面。
        a. 登录阿里云E-MapReduce控制台。
        b. 在顶部菜单栏处，根据实际情况选择地域和资源组。
        c. 单击上方的**集群管理**页签。
        d. 在**集群管理**页面，单击相应集群所在行的**详情**。
        e. 在左侧导航栏，选择**集群服务 > JindoData**。
    ii. 在JindoData服务页面，单击**配置**页签。
    iii. 在**服务配置**区域，单击**common**页签。
2. 在**新增配置项**对话框中，新增以下配置项，单击**确定**。
    i. 单击**自定义配置**。

ii. 在**新增配置项**对话框中，新增以下配置项。

新增配置项的具体操作，请参见添加组件参数。

- 全局方式配置（所有Bucket使用同一种方式）

参数	描述
jindofsx.oss.accessKeyId	OSS的AccessKey ID。
jindofsx.oss.accessKeySecret	OSS的AccessKey Secret。
jindofsx.oss.endpoint	OSS-HDFS服务的Endpoint。例如，cn-***.oss-dls.aliyuncs.com。

- 按照Bucket配置

参数	描述
jindofsx.oss.bucket.XXX.accessKeyId	`XXX`的Bucket的AccessKey ID。
jindofsx.oss.bucket.XXX.accessKeySecret	`XXX`的Bucket的AccessKey Secret。
jindofsx.oss.bucket.XXX.endpoint	`XXX`的Bucket的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
jindofsx.oss.bucket.YYY.accessKeyId	`YYY`的Bucket的AccessKey ID。
jindofsx.oss.bucket.YYY.accessKeySecret	`YYY`的Bucket的AccessKey Secret。
jindofsx.oss.bucket.YYY.endpoint	`YYY`的Bucket的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
jindofsx.oss.bucket.YYY.user	Storage Service访问OSS-HDFS服务使用的用户名。

> **说明** `XXX`和`YYY`为OSS-HDFS服务Bucket的名称。

iii. 单击**确定**。

3. 重启服务。
   i. 在JindoData服务页面，选择右上角的**操作 > 重启All Components**。
   ii. 在**执行集群操作**对话框中，输入执行原因（其他参数保持默认），单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

步骤二：配置JindoSDK

> **注意** 此配置为客户端配置，无需重启JindoData服务。

1. 进入配置页面。
   i. 进入HDFS服务页面。
      a. 登录EMR on ECS控制台。
      b. 在顶部菜单栏处，根据实际情况选择地域和资源组。
      c. 单击上方的**集群管理**页签。
      d. 在**集群管理**页面，单击相应集群所在行的**详情**。
      e. 在左侧导航栏，选择**集群服务 > HDFS**。
   ii. 在HDFS服务页面，单击**配置**页签。
   iii. 在**服务配置**区域，单击**core-site**页签。
2. 新增和修改配置项。

   新增配置项的具体操作，请参见添加组件参数。修改配置项的具体操作，请参见修改组件参数。

内容	参数	描述
配置统一名字空间使用的实现类	fs.AbstractFileSystem.jindo.impl	固定值为com.aliyun.jindodata.jindo.JINDO。
	fs.jindo.impl	固定值为com.aliyun.jindodata.jindo.JindoFileSystem。
	fs.xengine	固定值为jindofsx。

内容	参数	描述
配置OSS-HDFS服务的AccessKey	fs.oss.accessKeyId	OSS-HDFS服务Bucket对应的AccessKey ID。
	fs.oss.accessKeySecret	OSS-HDFS服务Bucket对应的AccessKey Secret。
配置OSS-HDFS服务的Endpoint	访问OSS-HDFS服务时需要配置Endpoint。 ○ （推荐）方式一：在访问路径中指定Endpoint。 推荐访问路径格式为 `oss://<Bucket>.<Endpoint>/<Object>` ，例如 `oss://example****.cn-shanghai.oss-dls.aliyuncs.com/example****.txt` 。 该方式在访问路径中包含Endpoint，JindoSDK会根据路径中的Endpoint访问对应接口。 ○ 方式二：配置Bucket级别的Endpoint。 如果使用 `oss://<Bucket>/<Object>` 格式的访问路径，即访问路径中未设置Endpoint，JindoSDK会在配置中查找Bucket级别的Endpoint。 您可以在HDFS服务的**core-site**页签添加自定义配置，从而指向JindoFS服务的Endpoint。设置**Key**为fs.oss.bucket.XXX.endpoint，**Value**为cn-***.oss-dls.aliyuncs.com。 ○ 方式三：配置全局默认Endpoint。 如果使用 `oss://<Bucket>/<Object>` 格式的访问路径，而且访问路径中未设置Bucket级别的Endpoint，则会用全局Endpoint访问。 您可以在HDFS服务的**core-site**页签添加自定义配置，从而指向JindoFS服务的Endpoint。设置**Key**为fs.oss.endpoint，**Value**为cn-***.oss-dls.aliyuncs.com。 **说明** `XXX` 为OSS-HDFS服务Bucket的名称。 配置完成后，JindoSDK会根据访问路径中的Endpoint访问对应的OSS-HDFS服务接口。	
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
启用缓存加速功能	fs.jindofsx.data.cache.enable	数据缓存开关： ○ false（默认值）：禁用数据缓存。 ○ true：启用数据缓存。 **说明** 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。
	fs.jindofsx.meta.cache.enable	元数据缓存开关： ○ false（默认值）：禁用元数据缓存。 ○ true：启用元数据缓存。
	fs.jindofsx.slice.cache.enable	小文件缓存优化开关： ○ false（默认值）：禁用小文件缓存。 ○ true：启用小文件缓存。
	fs.jindofsx.ram.cache.enable	内存缓存开关： ○ false（默认值）：禁用内存缓存。 ○ true：启用内存缓存。
	fs.jindofsx.short.circuit.enable	短路读开关： ○ true（默认值）：打开短路读开关。 ○ false：关闭短路读开关。

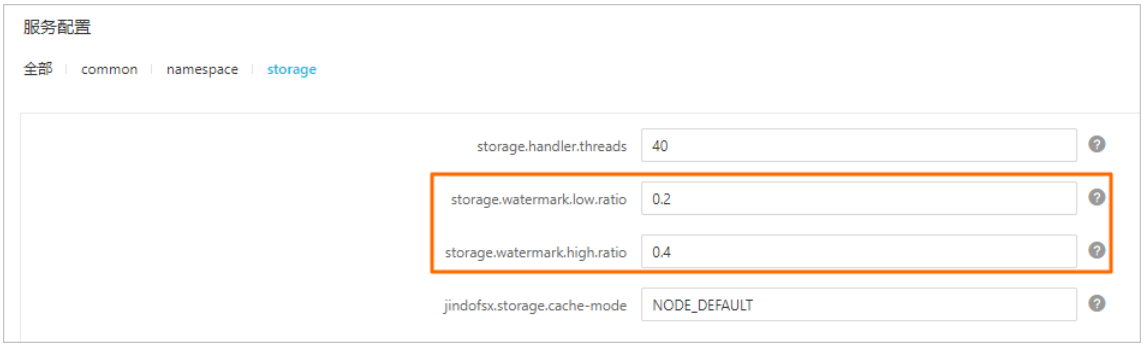
说明 完成以上配置，作业读取JindoFS上的数据后，会自动缓存到JindoFSx存储加速系统中，作业访问JindoFS的方式无需做任何修改，后续访问相同的数据就能够命中缓存。

步骤三：磁盘空间水位控制

缓存启用后，JindoFSx服务会自动管理本地缓存备份，通过水位清理本地缓存，请您根据需求配置一定的比例用于缓存。JindoFSx后端基于OSS，可以提供海量的存储，但是本地盘的容量是有限的，因此JindoFSx会自动淘汰本地较冷的数据备份。您可以通过修改**storage.watermark.high.ratio**和**storage.watermark.low.ratio**两个参数来调节本地存储的使用容量，值均为0～1的小数，表示使用磁盘空间的比例。

1. 修改磁盘水位配置。

   在JindoData服务**配置**页面的**服务配置**区域，单击**storage**页签，修改以下参数。

参数	描述
storage.watermark.low.ratio	表示使用量的下水位比例，触发清理后会自动清理冷数据，将JindoFS数据目录占用空间清理到下水位。默认值：0.2。
storage.watermark.high.ratio	表示磁盘使用量的上水位比例，每块数据盘的JindoFS数据目录占用的磁盘空间到达上水位即会触发清理。默认值：0.4。

   > **说明** 修改该参数时，下水位必须小于上水位，设置合理的值即可。

2. 保存配置。
   i. 单击**服务配置**区域的**保存**。
   ii. 在**确认修改**对话框中，输入执行原因，开启**自动更新配置**，单击**确定**。
3. 重启服务。
   i. 在JindoData服务页面，选择右上角的**操作** > **重启All Components**。
   ii. 在**执行集群操作**对话框中，输入执行原因（其他参数保持默认），单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

步骤四：挂载OSS-HDFS服务

挂载OSS-HDFS服务目录的语法如下。

```
jindo admin -mount <path> <realpath>
```

例如，执行以下命令挂载OSS-HDFS服务目录。

```
jindo admin -mount /jindodls oss://<yourBucketName>.<yourBucketEndpoint>/
```

> **说明** 执行上述命令后，则 `/jindodls` 目录真正挂载的文件路径是 `oss://<yourBucketName>.<yourBucketEndpoint>/` 。

执行以下命令，返回信息为 `jindo://emr-header-1:8101/jindodls` 。

```
hdfs dfs -ls jindo://emr-header-1:8101/
```

即访问 `jindo://emr-header-1:8101/jindodls/` 相当于访问 `oss://<yourBucketName>/<yourBucketObject>` 。

步骤五：访问阿里云OSS

您通过 `jindo://` 前缀读取OSS-HDFS服务上的数据后，在数据缓存开关打开时，会自动缓存到JindoFSx存储加速系统中，后续通过 `jindo://` 访问相同的数据就能够命中缓存。

6.4.2.2.5. Apache HDFS透明缓存加速

Apache HDFS透明缓存加速可以利用计算集群的闲置存储资源对远端HDFS集群进行数据缓存，避免了计算集群或服务占用核心集群过多带宽。适用于在HDFS集群和计算集群分离，HDFS集群访问性能不及预期时，您可以通过在计算集群或靠近计算集群的地方缓存数据来进行加速。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> **说明** 本文以EMR-3.40.0版本为例介绍。

操作流程

1. 步骤一：配置服务端
2. 步骤二：配置JindoSDK
3. 步骤三：访问HDFS

步骤一：配置服务端

1. 进入新增配置项页面。
   i. 进入JindoData服务页面。
      a. 登录阿里云E-MapReduce控制台。
      b. 在顶部菜单栏处，根据实际情况选择地域和资源组。
      c. 单击上方的**集群管理**页签。
      d. 在**集群管理**页面，单击相应集群所在行的**详情**。
      e. 在左侧导航栏，选择**集群服务 > JindoData**。
   ii. 在JindoData服务页面，单击**配置**页签。
   iii. 在**服务配置**区域，单击**common**页签。
2. 新增配置。
   i. 单击**自定义配置**。
   ii. 在**新增配置项**对话框中，新增以下配置项。
   新增配置项的具体操作，请参见添加组件参数。

集群类型	参数	描述
普通集群	jindofsx.hdfs.user	Storage Service访问HDFS使用的用户名。
HA集群	jindofsx.hdfs.XXX.dfs.ha.namenodes	填写格式为 `nn1,nn2` 。
	jindofsx.hdfs.XXX.dfs.namenode.rpc-address.nn1	填写格式为 `nn1-host1:nn1-rpc-port` 。
	jindofsx.hdfs.XXX.dfs.namenode.rpc-address.nn2	填写格式为 `nn2-host1:nn2-rpc-port` 。

> **说明** 根据您集群的类型，新增相应的配置项。 `XXX` 为集群中配置的dfs.nameservices参数值。

   iii. 单击**确定**。
3. 重启服务。
   i. 在JindoData服务页面，选择右上角的**操作 > 重启All Components**。
   ii. 在**执行集群操作**对话框中，输入执行原因（其他参数保持默认），单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

步骤二：配置JindoSDK

> **注意** 此配置为客户端配置，无需重启JindoData服务。

1. 进入配置页面。

i. 进入HDFS服务页面。
   a. 登录EMR on ECS控制台。
   b. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   c. 单击上方的**集群管理**页签。
   d. 在**集群管理**页面，单击相应集群所在行的**详情**。
   e. 在左侧导航栏，选择**集群服务 > HDFS**。
ii. 在HDFS服务页面，单击**配置**页签。
iii. 在**服务配置**区域，单击**core-site**页签。

2. 新增和修改配置项。

   新增配置项的具体操作，请参见添加组件参数。修改配置项的具体操作，请参见修改组件参数。

内容	参数	描述
配置统一名字空间使用的实现类	fs.hdfs.impl	固定值为com.aliyun.jindodata.hdfs.JindoHdfsFileSystem。
	fs.AbstractFileSystem.hdfs.impl	固定值为com.aliyun.jindodata.hdfs.HDFS。
	fs.xengine	固定值为jindofsx。
（可选）配置HA Namenodes **说明** 如果为HA集群，则需要配置该类参数。	fs.jindofsx.hdfs.XXX.dfs.ha.namenodes	格式为 `nn1,nn2` 。
	fs.jindofsx.hdfs.XXX.dfs.namenode.rpc-address.nn1	格式为 `nn1-host1:nn1-rpc-port` 。
	fs.jindofsx.hdfs.XXX.dfs.namenode.rpc-address.nn2	格式为 `nn2-host1:nn2-rpc-port` 。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
开启缓存加速 **说明** 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为禁用，即所有OSS都可直接访问OSS上的数据。	fs.jindofsx.data.cache.enable	数据缓存开关： ○ false（默认值）：禁用数据缓存。 ○ true：启用数据缓存。
	fs.jindofsx.meta.cache.enable	元数据缓存开关： ○ false（默认值）：禁用元数据缓存。 ○ true：启用元数据缓存。
	fs.jindofsx.slice.cache.enable	小文件缓存优化开关： ○ false（默认值）：禁用小文件缓存。 ○ true：启用小文件缓存。
	fs.jindofsx.ram.cache.enable	内存缓存开关： ○ false（默认值）：禁用内存缓存。 ○ true：启用内存缓存。
	fs.jindofsx.short.circuit.enable	短路读开关： ○ true（默认值）：打开。 ○ false：关闭。

步骤三：访问HDFS

您通过 `hdfs://` 前缀读取HDFS上的数据后，在数据缓存开关打开时，会自动缓存到JindoFSx存储加速系统中，后续通过 `hdfs://` 访问相同的数据就能够命中缓存。

6.4.2.2.6. Apache HDFS统一挂载缓存加速

Apache HDFS统一挂载缓存加速可以利用计算集群的闲置存储资源进行数据缓存来加速计算服务，避免了计算集群或服务占用核心集群过多带宽。在计算集群与待访问的HDFS集群网络带宽有限的情况下，可以通过缓存加速提升计算集群访问HDFS集群的速度。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> **说明** 本文以EMR-3.40.0版本为例介绍。

操作流程

1. 步骤一：配置服务端
2. 步骤二：配置JindoSDK
3. 步骤三：挂载HDFS
4. 步骤四：访问HDFS

步骤一：配置服务端

1. 进入新增配置项页面。
   i. 进入JindoData服务页面。
      a. 登录阿里云E-MapReduce控制台。
      b. 在顶部菜单栏处，根据实际情况选择地域和资源组。
      c. 单击上方的**集群管理**页签。
      d. 在**集群管理**页面，单击相应集群所在行的**详情**。
      e. 在左侧导航栏，选择**集群服务 > JindoData**。
   ii. 在JindoData服务页面，单击**配置**页签。
   iii. 在**服务配置**区域，单击**common**页签。
2. 新增配置。
   i. 单击**自定义配置**。
   ii. 在**新增配置项**对话框中，新增以下配置项。
   新增配置项的具体操作，请参见添加组件参数。

集群类型	参数	描述
普通集群	jindofsx.hdfs.user	Storage Service访问HDFS使用的用户名。
HA集群	jindofsx.hdfs.XXX.dfs.ha.namenodes	填写格式为 `nn1,nn2` 。
	jindofsx.hdfs.XXX.dfs.namenode.rpc-address.nn1	填写格式为 `nn1-host1:nn1-rpc-port` 。
	jindofsx.hdfs.XXX.dfs.namenode.rpc-address.nn2	填写格式为 `nn2-host1:nn2-rpc-port` 。

> **说明** 根据您集群的类型，新增相应的配置项。`XXX` 为集群中配置的dfs.nameservices参数值。

   iii. 单击**确定**。
3. 重启服务。
   i. 在JindoData服务页面，选择右上角的**操作 > 重启All Components**。
   ii. 在**执行集群操作**对话框中，输入执行原因（其他参数保持默认），单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

步骤二：配置JindoSDK

> **注意** 此配置为客户端配置，无需重启JindoData服务。

1. 进入配置页面。
   i. 进入HDFS服务页面。
      a. 登录EMR on ECS控制台。
      b. 在顶部菜单栏处，根据实际情况选择地域和资源组。
      c. 单击上方的**集群管理**页签。
      d. 在**集群管理**页面，单击相应集群所在行的**详情**。
      e. 在左侧导航栏，选择**集群服务 > HDFS**。

ii. 在HDFS服务页面，单击**配置**页签。

iii. 在**服务配置**区域，单击**core-site**页签。

2. 新增和修改配置项。

新增配置项的具体操作，请参见添加组件参数。修改配置项的具体操作，请参见修改组件参数。

内容	参数	描述
配置统一名字空间使用的实现类	fs.AbstractFileSystem.jindo.impl	固定值为com.aliyun.jindodata.jindo.JINDO。
	fs.jindo.impl	固定值为com.aliyun.jindodata.jindo.JindoFileSystem。
	fs.xengine	固定值为jindofsx。
（可选）配置HA Namenodes **说明** 如果为HA集群，则需要配置该类参数。	fs.jindofsx.hdfs.XXX.dfs.ha.namenodes	格式为 `nn1,nn2` 。
	fs.jindofsx.hdfs.XXX.dfs.namenode.rpc-address.nn1	格式为 `nn1-host1:nn1-rpc-port` 。
	fs.jindofsx.hdfs.XXX.dfs.namenode.rpc-address.nn2	格式为 `nn2-host1:nn2-rpc-port` 。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
开启缓存加速 **说明** 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为禁用，即所有OSS都可直接访问OSS上的数据。	fs.jindofsx.data.cache.enable	数据缓存开关： ○ false（默认值）：禁用数据缓存。 ○ true：启用数据缓存。
	fs.jindofsx.meta.cache.enable	元数据缓存开关： ○ false（默认值）：禁用元数据缓存。 ○ true：启用元数据缓存。
	fs.jindofsx.slice.cache.enable	小文件缓存优化开关： ○ false（默认值）：禁用小文件缓存。 ○ true：启用小文件缓存。
	fs.jindofsx.ram.cache.enable	内存缓存开关： ○ false（默认值）：禁用内存缓存。 ○ true：启用内存缓存。
	fs.jindofsx.short.circuit.enable	短路读开关： ○ true（默认值）：打开。 ○ false：关闭。

步骤三：挂载HDFS

- 语法

```
jindo admin -mount <path> <realpath>
```

- 示例

执行以下命令，挂载HDFS目录。

```
jindo admin -mount /jindohdfs hdfs://emr-header-1:9000/
```

执行如上命令后，则 `jindo:///jindohdfs` 目录下真正挂载的文件路径是 `hdfs://emr-header-1:9000/` 。

步骤四：访问HDFS

您通过 `jindo://` 前缀读取HDFS上的数据后，在数据缓存开关打开时会自动缓存到JindoFSx存储加速系统中，后续通过 `jindo://` 访问相同的数据就能够命中缓存。

6.4.2.2.7. 阿里云文件存储NAS统一挂载缓存加速

阿里云文件存储NAS（Apsara File Storage NAS）是一个可大规模共享访问，弹性扩展的高性能云原生分布式文件系统。支持智能冷热数据分层，有效降低数据存储成本。本文主要为您介绍JindoFSx支持阿里云文件存储NAS统一挂载缓存加速的使用方式。

背景信息

文件存储NAS的详细信息，请参见什么是文件存储NAS。

前提条件

- 已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

  > **说明** 本文以EMR-3.40.0版本为例介绍。

- 集群已挂载NFS文件系统，详情请参见Linux系统挂载NFS文件系统。

操作流程

1. 步骤一：配置JindoSDK
2. 步骤二：挂载NAS文件系统目录
3. 步骤三：访问NAS文件系统目录

步骤一：配置JindoSDK

1. 进入配置页面。
   i. 进入HDFS服务页面。
      a. 登录EMR on ECS控制台。
      b. 在顶部菜单栏处，根据实际情况选择地域和资源组。
      c. 单击上方的**集群管理**页签。
      d. 在**集群管理**页面，单击相应集群所在行的**详情**。
      e. 在左侧导航栏，选择**集群服务 > HDFS**。
   ii. 在HDFS服务页面，单击**配置**页签。
   iii. 在**服务配置**区域，单击**core-site**页签。
2. 新增和修改配置项。

   新增配置项的具体操作，请参见添加组件参数。修改配置项的具体操作，请参见修改组件参数。

内容	参数	描述
配置统一名字空间使用的实现类	fs.jindo.impl	固定值为com.aliyun.jindodata.jindo.JindoFileSystem。
	fs.AbstractFileSystem.jindo.impl	固定值为com.aliyun.jindodata.jindo.JINDO。
	fs.xengine	固定值为jindofsx。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
	fs.jindofsx.data.cache.enable	数据缓存开关： ○ false（默认值）：禁用数据缓存。 ○ true：启用数据缓存。 **说明** 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。
	fs.jindofsx.meta.cache.enable	元数据缓存开关： ○ false（默认值）：禁用元数据缓存。 ○ true：启用元数据缓存。

启用缓存加速功能 内容	参数	描述
	fs.jindofsx.slice.cache.enable	小文件缓存优化开关： ○ false（默认值）：禁用小文件缓存。 ○ true：启用小文件缓存。
	fs.jindofsx.ram.cache.enable	内存缓存开关： ○ false（默认值）：禁用内存缓存。 ○ true：启用内存缓存。
	fs.jindofsx.short.circuit.enable	短路读开关： ○ true（默认值）：打开短路读开关。 ○ false：关闭短路读开关。

步骤二：挂载NAS文件系统目录

- 语法

```
jindo admin -mount <path> <realpath>
```

- 示例

  执行以下命令，挂载NAS文件系统目录。

```
jindo admin -mount /jindonas /mnt/nas
```

  假设NAS文件系统挂载在服务器上的本地路径为 `/mnt/nas` 。执行如上命令后，则 `/jindonas` 目录下真正挂载的文件路径是 `/mnt/nas` 。

  执行 `hdfs dfs -ls jindo://emr-header-1:8101/` 命令，返回如下信息，即访问 `jindo://emr-header-1:8101` 等价于访问 `/mnt/nas/` 。

```
----------   1          0 1970-01-01 08:00 jindo://emr-header-1:8101/jindonas
```

步骤三：访问NAS文件系统目录

您通过 `jindo://` 前缀读取NAS上的数据后，在数据缓存开关打开时，会自动缓存到JindoFSx存储加速系统中，后续通过 `jindo://` 访问相同的数据就能够命中缓存。

6.4.2.2.8. P2P分布式下载缓存

JindoFSx客户端P2P可以被视作一种本地缓存（LocalCache）。与原有的LocalCache相比，P2P缓存中的本地数据块会优先从其他持有该数据的客户端拉取，只有无法向其他客户端请求时，才会从STS或远端读取。本文为您介绍P2P分布式下载缓存的使用方法。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> **说明** 本文以EMR-3.40.0版本为例介绍。

操作流程

1. 进入新增配置项页面。
   i. 进入JindoData服务页面。
      a. 登录阿里云E-MapReduce控制台。
      b. 在顶部菜单栏处，根据实际情况选择地域和资源组。
      c. 单击上方的**集群管理**页签。
      d. 在**集群管理**页面，单击相应集群所在行的**详情**。
      e. 在左侧导航栏，选择**集群服务 > JindoData**。
   ii. 在JindoData服务页面，单击**配置**页签。
   iii. 在**服务配置**区域，单击**common**页签。
2. 新增配置。
   i. 单击**自定义配置**。

ii. 在**新增配置项**对话框中，新增以下配置项。

新增配置项的具体操作，请参见添加组件参数。

配置项	参数	描述
服务端配置	jindofsx.p2p.tracker.thread.number	TrackerService的处理线程数。 如果要开启P2P功能，则该参数值必须设置大于1。如果小于等于1，则不会创建TrackerService，也不会开启P2P功能。
	jindofsx.p2p.file.prefix	使用P2P下载的前缀列表。当包含多个文件路径时，使用半角逗号（,）隔开，文件路径只有匹配到其中任一个前缀，才会以P2P方式下载。在应用层使用统一挂载路径进行下载时，此处仍应配置为真实的对象路径。 例如，`oss://bucket1/data-dir1/,oss://bucket2/data-dir2/`。
客户端配置	fs.jindofsx.p2p.cache.capacity.limit	P2P下载最大占用的缓存大小，单位为字节，默认为5 GB，最小值为1 GB。 例如，取值为5 * 1024 * 1024 * 1024。
	fs.jindofsx.p2p.download.parallelism.per.file	P2P下载单个文件使用的并发数。 例如，取值为5。
	fs.jindofsx.p2p.download.thread.pool.size	P2P下载使用的线程池总大小。 例如，取值为5。

iii. 单击**确定**。

3. 重启服务。

i. 在JindoData服务页面，选择右上角的**操作 > 重启All Components**。

ii. 在**执行集群操作**对话框中，输入执行原因（其他参数保持默认），单击**确定**。

iii. 在**确认**对话框中，单击**确定**。

6.4.2.2.9. JindoShell CLI支持JindoFSx使用说明

JindoShell CLI支持操作JindoFSx数据缓存、元数据缓存和统一命名空间等命令。

背景信息

本文为您介绍以下内容：

- 数据缓存命令
- 元数据缓存命令
- 清理缓存命令
- 统一命名空间命令
- 其他命令

数据缓存命令

数据缓存命令可以备份对应路径的数据至本集群的磁盘，以便于后续可以读取本地数据，无需读取OSS等后端上的数据。

```
jindo fsx -load -data <options> <path>
```

参数	描述
`<options>`	各种可选参数： • `-s`：表示缓存过程同步执行。即缓存完成前命令不退出，日志直接打印在控制台上。推荐开启。 • `-replica`：缓存副本数量，默认缓存1个副本。 • `-R`：递归缓存文件，当`<path>`是文件夹时需开启。 • `-cachelist`：接收本地文件，文件内容为cache列表。
`<path>`	数据缓存路径。

推荐使用以下组合命令。

```
jindo fsx -load -data -s -R <path>
```

元数据缓存命令

元数据缓存命令可以备份远端文件的元数据信息，从而后续无需从OSS等后端读取文件元数据信息。

```
jindo fsx -load -meta <options> <path>
```

参数	描述
`<options>`	各种可选参数： • `-s`：表示缓存过程同步执行，即缓存完成前命令不退出，日志直接打印在控制台上。推荐开启。 • `-R`：递归缓存文件，当`<path>`是文件夹时需开启。
`<path>`	元数据缓存路径。

推荐使用以下组合命令。

```
jindo fsx -load -meta -s -R <path>
```

数据缓存和元数据缓存可以组合使用，当需要同时进行二者缓存时，可以搭配可选参数使用。推荐使用以下组合命令。

```
jindo fsx -load -meta -data -s -R <path>
```

清理缓存命令

uncache命令可以删除本地集群中的本地备份，只存储数据在OSS标准存储上，以便于后续读取OSS上的数据。

```
jindo fsx -uncache <path>
```

统一命名空间命令

- 添加一个挂载点

  ```
  jindo fsxadmin -mount <path> <realpath>
  ```

- 移除一个挂载点

  ```
  jindo fsxadmin -unmount <path>
  ```

其他命令

执行以下命令，输出当前缓存系统的信息，例如缓存大小，缓存容量等。

```
jindo fsx -report
```

输出信息如下。

```
Namespace Address: 127.0.0.1:8101
Rpc Port: 8101
Started: Mon Jan 10 15:23:51 2022
Version: 4.1.0
Live Nodes: 2
Decommission Nodes: 0
Total Disk Capacity: 438.17GB
Used Disk Capacity: 5120.00MB
Total MEM Capacity: 4096.00MB
Used MEM Capacity: 0B
```

6.4.2.3. 大数据生态

6.4.2.3.1. Hadoop访问阿里云OSS+JindoFSx透明加速

JindoFSx存储加速系统提供了透明缓存的使用方式，兼容原生OSS存储方式，文件以对象的形式存储在OSS上，每个文件根据实际访问情况会在本地进行缓存，提升访问OSS的效率，同时兼容了原有OSS文件形式，数据访问上能够与其他OSS客户端完全兼容，作业访问OSS的方式无需做任何修改。本文为您介绍Hadoop如何使用JindoSDK处理阿里云OSS上的数据。

背景信息

JindoSDK包含一些高级调优参数，配置方式以及配置项请参见JindoSDK高级参数配置。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> **说明** 本文以EMR-3.40.0版本为例介绍。

操作流程

1. 步骤一：配置AccessKey
2. 步骤二：配置JindoSDK
3. 步骤三：使用JindoSDK访问OSS

步骤一：配置AccessKey

1. 进入新增配置项页面。
   i. 进入JindoData服务页面。
      a. 登录阿里云E-MapReduce控制台。
      b. 在顶部菜单栏处，根据实际情况选择地域和资源组。
      c. 单击上方的**集群管理**页签。
      d. 在**集群管理**页面，单击相应集群所在行的**详情**。
      e. 在左侧导航栏，选择**集群服务 > JindoData**。
   ii. 在JindoData服务页面，单击**配置**页签。
   iii. 在**服务配置**区域，单击**common**页签。
2. 新增配置。
   i. 单击**自定义配置**。
   ii. 在**新增配置项**对话框中，新增以下配置项。

   新增配置项的具体操作，请参见添加组件参数。

   - 全局方式配置（所有Bucket使用同一种方式）

参数	描述
jindofsx.oss.accessKeyId	OSS的AccessKey ID。
jindofsx.oss.accessKeySecret	OSS的AccessKey Secret。
jindofsx.oss.endpoint	OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。

   - 按照Bucket配置

参数	描述
jindofsx.oss.bucket.XXX.accessKeyId	`XXX`的Bucket的AccessKey ID。
jindofsx.oss.bucket.XXX.accessKeySecret	`XXX`的Bucket的AccessKey Secret。
jindofsx.oss.bucket.XXX.endpoint	`XXX`的Bucket的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
jindofsx.oss.bucket.YYY.accessKeyId	`YYY`的Bucket的AccessKey ID。
jindofsx.oss.bucket.YYY.accessKeySecret	`YYY`的Bucket的AccessKey Secret。
jindofsx.oss.bucket.YYY.endpoint	`YYY`的Bucket的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。

   > **说明** `XXX`和`YYY`为OSS Bucket的名称。

   iii. 单击**确定**。
3. 重启服务。
   i. 在JindoData服务页面，选择右上角的**操作 > 重启All Components**。

ii. 在**执行集群操作**对话框中，输入执行原因（其他参数保持默认），单击**确定**。

iii. 在**确认**对话框中，单击**确定**。

步骤二：配置JindoSDK

1. 进入配置页面。
   i. 进入HDFS服务页面。
      a. 登录EMR on ECS控制台。
      b. 在顶部菜单栏处，根据实际情况选择地域和资源组。
      c. 单击上方的**集群管理**页签。
      d. 在**集群管理**页面，单击相应集群所在行的**详情**。
      e. 在左侧导航栏，选择**集群服务 > HDFS**。
   ii. 在HDFS服务页面，单击**配置**页签。
   iii. 在**服务配置**区域，单击**core-site**页签。
2. 新增和修改配置项。

   新增配置项的具体操作，请参见添加组件参数。修改配置项的具体操作，请参见修改组件参数。

内容	参数	描述
配置JindoSDK OSS实现类	fs.AbstractFileSystem.oss.impl	固定值为com.aliyun.jindodata.oss.OSS。
	fs.oss.impl	固定值为com.aliyun.jindodata.oss.JindoOssFileSystem。
	fs.xengine	固定值为jindofsx。
配置Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
开启缓存加速功能	fs.jindofsx.data.cache.enable	数据缓存开关： false（默认值）：禁用数据缓存。 true：启用数据缓存。 **说明** 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。

步骤三：使用JindoSDK访问OSS

您可以使用Hadoop Shell访问OSS，常用命令如下：

- put操作

```
hadoop fs -put <path> oss://<BucketName>/
```

- ls操作

```
hadoop fs -ls oss://<BucketName>/
```

- mkdir操作

```
hadoop fs -mkdir oss://<BucketName>/<path>
```

- rm操作

```
hadoop fs -rm oss://<BucketName>/<path>
```

6.4.2.3.2. Hadoop访问阿里云OSS-HDFS服务+JindoFSx透明加速

JindoFSx存储加速系统提供了透明缓存的使用方式，兼容原生OSS存储方式，文件以对象的形式存储在OSS上，每个文件根据实际访问情况会在本地进行缓存，提升访问OSS的效率，同时兼容了原有OSS文件形式，数据访问上能够与其他OSS客户端完全兼容，作业访问OSS的方式无需做任何修改。本文为您介绍Hadoop如何使用JindoSDK查询OSS-HDFS服务（JindoFS服务）中的数据。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> **说明** 本文以EMR-3.40.0版本为例介绍。

操作流程

1. 步骤一：配置AccessKey
2. 步骤二：配置JindoSDK
3. 步骤三：使用JindoSDK访问OSS-HDFS服务

步骤一：配置AccessKey

1. 进入新增配置项页面。
   i. 进入JindoData服务页面。
      a. 登录阿里云E-MapReduce控制台。
      b. 在顶部菜单栏处，根据实际情况选择地域和资源组。
      c. 单击上方的**集群管理**页签。
      d. 在**集群管理**页面，单击相应集群所在行的**详情**。
      e. 在左侧导航栏，选择**集群服务 > JindoData**。
   ii. 在JindoData服务页面，单击**配置**页签。
   iii. 在**服务配置**区域，单击**common**页签。
2. 新增配置。
   i. 单击**自定义配置**。
   ii. 在**新增配置项**对话框中，新增以下配置项。

   新增配置项的具体操作，请参见添加组件参数。

   - 全局方式配置（所有Bucket使用同一种方式）

参数	描述
jindofsx.oss.accessKeyId	OSS的AccessKey ID。
jindofsx.oss.accessKeySecret	OSS的AccessKey Secret。
jindofsx.oss.endpoint	OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。

   - 按照Bucket配置

参数	描述
jindofsx.oss.bucket.XXX.accessKeyId	`XXX`的Bucket的AccessKey ID。
jindofsx.oss.bucket.XXX.accessKeySecret	`XXX`的Bucket的AccessKey Secret。
jindofsx.oss.bucket.XXX.endpoint	`XXX`的Bucket的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
jindofsx.oss.bucket.YYY.accessKeyId	`YYY`的Bucket的AccessKey ID。
jindofsx.oss.bucket.YYY.accessKeySecret	`YYY`的Bucket的AccessKey Secret。
jindofsx.oss.bucket.YYY.endpoint	`YYY`的Bucket的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。

   > **说明** `XXX`和`YYY`为OSS Bucket的名称。

   iii. 单击**确定**。
3. 重启服务。
   i. 在JindoData服务页面，选择右上角的**操作 > 重启All Components**。

ii. 在**执行集群操作**对话框中，输入执行原因（其他参数保持默认），单击**确定**。

iii. 在**确认**对话框中，单击**确定**。

步骤二：配置JindoSDK

1. 进入配置页面。
   i. 进入HDFS服务页面。
      a. 登录EMR on ECS控制台。
      b. 在顶部菜单栏处，根据实际情况选择地域和资源组。
      c. 单击上方的**集群管理**页签。
      d. 在**集群管理**页面，单击相应集群所在行的**详情**。
      e. 在左侧导航栏，选择**集群服务 > HDFS**。
   ii. 在HDFS服务页面，单击**配置**页签。
   iii. 在**服务配置**区域，单击**core-site**页签。
2. 新增和修改配置项。

   新增配置项的具体操作，请参见添加组件参数。修改配置项的具体操作，请参见修改组件参数。

内容	参数	描述
配置JindoSDK OSS实现类	fs.AbstractFileSystem.oss.impl	固定值为com.aliyun.jindodata.oss.OSS。
	fs.oss.impl	固定值为com.aliyun.jindodata.oss.JindoOssFileSystem。
	fs.xengine	固定值为jindofsx。
配置Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	OSS-HDFS服务的Endpoint。例如，cn-***.oss-dls.aliyuncs.com。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
开启JindoFSx透明加速缓存功能	fs.jindofsx.data.cache.enable	数据缓存开关： false（默认值）：禁用数据缓存。 true：启用数据缓存。 **说明** 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。

3. 重启HDFS服务，具体操作请参见重启服务。

步骤三：使用JindoSDK访问OSS-HDFS服务

您可以使用Hadoop Shell访问JindoFS，常用命令如下：

- put操作

```
hadoop fs -put <path> oss://<BucketName>.<Endpoint>/
```

- ls操作

```
hadoop fs -mkdir oss://<BucketName>.<Endpoint>/<path>
```

- mkdir操作

```
hadoop fs -rm oss://<BucketName>.<Endpoint>/<path>
```

- rm操作

```
hadoop fs -rm oss://<BucketName>/<path>
```

6.4.2.3.3. Hadoop访问JindoFSx统一挂载的数据

JindoSDK为JindoFSx存储加速系统提供了Apache Hadoop支持。本文介绍Hadoop如何处理JindoFSx统一挂载的数据。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> **说明** 本文以EMR-3.40.0版本为例介绍。

操作步骤

1. （可选）如果想使用缓存功能，请先配置JindoData服务的AccessKey信息，具体操作请参见步骤一：配置AccessKey。
2. 进入HDFS服务的core-site页签。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务 > HDFS**。
   vi. 在HDFS服务页面，单击**配置**页签。
   vii. 在**服务配置**区域，单击**core-site**页签。
3. 新增和修改配置项。

   新增配置项的具体操作，请参见添加组件参数。修改配置项的具体操作，请参见修改组件参数。

内容	参数	描述
配置实现类	fs.AbstractFileSystem.jindo.impl	固定值为com.aliyun.jindodata.jindo.JINDO。
	fs.jindo.impl	固定值为com.aliyun.jindodata.jindo.JindoFileSystem。
	fs.xengine	固定值为jindofsx。
配置OSS Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	○ OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。 ○ OSS-HDFS服务的Endpoint。例如，cn-***.oss-dls.aliyuncs.com。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
缓存加速功能（可选） **注意** 如果使用缓存功能，请执行步骤1。	fs.jindofsx.data.cache.enable	数据缓存开关： ○ false（默认值）：禁用数据缓存。 ○ true：启用数据缓存。 **说明** 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。
	fs.jindofsx.meta.cache.enable	元数据缓存开关： ○ false（默认值）：禁用元数据缓存。 ○ true：启用元数据缓存。
	fs.jindofsx.slice.cache.enable	小文件缓存优化开关： ○ false（默认值）：禁用小文件缓存。 ○ true：启用小文件缓存。

内容	参数	描述
	fs.jindofsx.ram.cache.enable	内存缓存开关： ○ false（默认值）：禁用内存缓存。 ○ true：启用内存缓存。
	fs.jindofsx.short.circuit.enable	短路读开关： ○ true（默认值）：打开。 ○ false：关闭。

4. 重启Presto服务，具体操作请参见重启服务。
5. 挂载OSS或OSS-HDFS服务目录。

   挂载OSS-HDFS服务目录语法如下。

   ```
   jindo admin -mount <path> <realpath>
   ```

   例如，执行以下命令挂载OSS-HDFS服务目录。

   ```
   jindo admin -mount /jindodls oss://<yourBucketName>.<yourBucketEndpoint>/
   ```

   > **说明** 执行上述命令后，则 `/jindodls` 目录真正挂载的文件路径是 `oss://<yourBucketName>.<yourBucketEndpoint>/` 。

6. 访问OSS或OSS-HDFS服务。

   您可以使用Hadoop Shell访问OSS或OSS-HDFS服务，常用命令如下：

   - put操作

     ```
     hadoop fs -put <path> jindo://emr-header-1:8101/jindooss/
     ```

   - ls操作

     ```
     hadoop fs -ls jindo://emr-header-1:8101/jindooss/
     ```

   - mkdir操作

     ```
     hadoop fs -mkdir jindo://emr-header-1:8101/jindooss/<path>
     ```

   - rm操作

     ```
     hadoop fs -rm jindo://emr-header-1:8101/jindooss/<path>
     ```

6.4.2.3.4. Spark处理阿里云OSS上的数据+JindoFSx透明加速

JindoSDK是一个简单易用面向Hadoop和Spark生态的OSS客户端，为阿里云OSS提供高度优化的Hadoop FileSystem实现，Spark使用JindoSDK相对于使用Hadoop社区OSS客户端，可以获得更好的性能，同时还能获得阿里云E-MapReduce产品的技术支持。JindoFSx存储加速系统提供了透明缓存的使用方式，兼容原生OSS存储方式，文件以对象的形式存储在OSS上，每个文件根据实际访问情况会在本地进行缓存，提升访问OSS的效率，同时兼容了原有OSS文件形式，数据访问上能够与其他OSS客户端完全兼容，作业访问OSS的方式无需做任何修改。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> **说明** 本文以EMR-3.40.0版本为例介绍。

操作步骤

1. （可选）如果想使用缓存功能，请先配置JindoData服务的AccessKey信息，具体操作请参见步骤一：配置AccessKey。
2. 进入HDFS服务的core-site页签。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务** > **HDFS**。
   vi. 在HDFS服务页面，单击**配置**页签。

vii. 在**服务配置**区域，单击**core-site**页签。

3. 配置JindoSDK。

   新增配置项的具体操作，请参见添加组件参数。修改配置项的具体操作，请参见修改组件参数。

内容	参数	描述
配置JindoSDK OSS实现类	fs.AbstractFileSystem.oss.impl	固定值为com.aliyun.jindodata.oss.OSS。
	fs.oss.impl	固定值为com.aliyun.jindodata.oss.JindoOssFileSystem。
	fs.xengine	固定值为jindofsx。
配置Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
开启透明缓存加速（可选） **注意** 如果使用缓存功能，请执行步骤1。	fs.jindofsx.data.cache.enable	数据缓存开关： false（默认值）：禁用数据缓存。 true：启用数据缓存。 **说明** 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。

4. 使用Spark访问OSS。
   - 创建表

     ```
     create table test_oss (c1 string) location "oss://<Bucket>/<path>";
     ```

   - 插入数据

     ```
     insert into table test_oss values ("testdata");
     ```

   - 查询OSS表

     ```
     select * from test_oss;
     ```

6.4.2.3.5. Spark处理阿里云OSS-HDFS服务上的数据+JindoFSx透明加速

本文为您介绍Spark如何使用JindoSDK处理OSS-HDFS服务（JindoFS服务）中的数据。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

说明 本文以EMR-3.40.0版本为例介绍。

操作步骤

1. （可选）如果想使用缓存功能，请先配置JindoData服务的AccessKey信息，具体操作请参见步骤一：配置AccessKey。
2. 进入HDFS服务的core-site页签。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务 > HDFS**。
   vi. 在HDFS服务页面，单击**配置**页签。
   vii. 在**服务配置**区域，单击**core-site**页签。

3. 新增和修改配置项。

全局配置

新增配置项的具体操作，请参见添加组件参数。修改配置项的具体操作，请参见修改组件参数。

内容	参数	描述
配置JindoSDK OSS-HDFS服务实现类	fs.AbstractFileSystem.oss.impl	固定值为com.aliyun.jindodata.oss.OSS。
	fs.oss.impl	固定值为com.aliyun.jindodata.oss.JindoOssFileSystem。
	fs.xengine	固定值为jindofsx。
配置OSS Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	OSS-HDFS服务的Endpoint。例如，cn-***.oss-dls.aliyuncs.com。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
开启透明缓存加速（可选） **注意** 如果使用缓存功能，请执行步骤1。	fs.jindofsx.data.cache.enable	数据缓存开关： • false（默认值）：禁用数据缓存。 • true：启用数据缓存。 **说明** 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。

任务级别配置

使用参数在提交Spark任务的时候设置JindoSDK，示例如下。

```
spark-submit --conf spark.hadoop.fs.AbstractFileSystem.oss.impl=com.aliyun.jindodata.oss.OSS --conf spark.hadoop.fs.oss.impl=com.aliyun.jindodata.oss.JindoOssFileSystem --conf spark.hadoop.fs.oss.accessKeyId=xxx  --conf spark.hadoop.fs.oss.accessKeySecret=xxx --conf spark.hadoop.fs.jindofsx.namespace.rpc.address=hostname:port --conf spark.hadoop.fs.jindofsx.data.cache.enable=true
```

4. 使用Spark访问OSS-HDFS服务。
   - 创建表

     ```
     create table test_oss (c1 string) location "oss://<Bucket>/<path>";
     ```

   - 插入数据

     ```
     insert into table test_oss values ("testdata");
     ```

   - 查询OSS表

     ```
     select * from test_oss;
     ```

6.4.2.3.6. Spark处理JindoFSx统一挂载的数据

本文为您介绍Spark如何处理JindoFSx统一挂载的数据。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

说明 本文以EMR-3.40.0版本为例介绍。

操作步骤

1. （可选）如果想使用缓存功能，请先配置JindoData服务的AccessKey信息，具体操作请参见步骤一：配置AccessKey。
2. 进入HDFS服务的core-site页签。

i. 登录阿里云E-MapReduce控制台。
ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
iii. 单击上方的**集群管理**页签。
iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
v. 在左侧导航栏，选择**集群服务 > HDFS**。
vi. 在HDFS服务页面，单击**配置**页签。
vii. 在**服务配置**区域，单击**core-site**页签。

3. 新增和修改配置项。

全局配置

i. 新增和修改配置项。

新增配置项的具体操作，请参见添加组件参数。修改配置项的具体操作，请参见修改组件参数。

内容	参数	描述
配置实现类	fs.AbstractFileSystem.jindo.impl	固定值为com.aliyun.jindodata.jindo.JINDO。
	fs.jindo.impl	固定值为com.aliyun.jindodata.jindo.JindoFileSystem。
	fs.xengine	固定值为jindofsx。
配置Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	■ OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。 ■ OSS-HDFS服务的Endpoint。例如，cn-***.oss-dls.aliyuncs.com。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 说明 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
缓存加速功能（可选） 注意 如果使用缓存功能，请执行步骤1。	fs.jindofsx.data.cache.enable	数据缓存开关： ■ false（默认值）：禁用数据缓存。 ■ true：启用数据缓存。 说明 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。
	fs.jindofsx.meta.cache.enable	元数据缓存开关： ■ false（默认值）：禁用元数据缓存。 ■ true：启用元数据缓存。
	fs.jindofsx.slice.cache.enable	小文件缓存优化开关： ■ false（默认值）：禁用小文件缓存。 ■ true：启用小文件缓存。
	fs.jindofsx.ram.cache.enable	内存缓存开关： ■ false（默认值）：禁用内存缓存。 ■ true：启用内存缓存。
	fs.jindofsx.short.circuit.enable	短路读开关： ■ true（默认值）：打开。 ■ false：关闭。

任务级别配置

使用参数在提交Spark任务的时候设置JindoSDK，示例如下。

```
spark-submit --conf spark.hadoop.fs.AbstractFileSystem.oss.impl=com.aliyun.jindodata.oss.OSS --conf spark.hadoop.fs
.oss.impl=com.aliyun.jindodata.oss.JindoOssFileSystem --conf spark.hadoop.fs.oss.accessKeyId=xxx  --conf spark.hado
op.fs.oss.accessKeySecret=xxx --conf spark.hadoop.fs.oss.endpoint=oss-cn-xxx-internal.aliyuncs.com --conf spark.had
oop.fs.jindofsx.namespace.rpc.address=hostname:port --conf spark.hadoop.fs.jindofsx.data.cache.enable=true
```

4. 挂载OSS或OSS-HDFS服务目录。

挂载语法如下。

```
jindo admin -mount <path> <realpath>
```

例如，执行以下命令挂载OSS目录。

```
jindo admin -mount /jindooss oss://<yourBucketName>.<yourBucketEndpoint>/
```

> **说明** 执行上述命令后，则 `/jindooss` 目录真正挂载的文件路径是 `oss://<yourBucketName>.<yourBucketEndpoint>/` 。

5. 使用Spark访问OSS。
   - 创建表

     ```
     create table test_oss (c1 string) location "jindo://emr-header-1:8101/jindooss/<path>";
     ```

   - 插入数据

     ```
     insert into table test_oss values ("testdata");
     ```

   - 查询OSS表

     ```
     select * from test_oss;
     ```

6.4.2.3.7. Hive处理阿里云OSS上的数据+JindoFSx透明加速

JindoFSx存储加速系统提供了透明缓存的使用方式，兼容原生OSS存储方式，文件以对象的形式存储在OSS上，每个文件根据实际访问情况会在本地进行缓存，提升访问OSS的效率，同时兼容了原有OSS文件形式，数据访问上能够与其他OSS客户端完全兼容，作业访问OSS的方式无需做任何修改。本文为您介绍Hive如何处理阿里云OSS上的数据。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> **说明** 本文以EMR-3.40.0版本为例介绍。

操作步骤

1. 配置JindoData服务的AccessKey信息，具体操作请参见步骤一：配置AccessKey。
2. 进入HDFS服务的core-site页签。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务 > HDFS**。
   vi. 在HDFS服务页面，单击**配置**页签。
   vii. 在**服务配置**区域，单击**core-site**页签。
3. 新增和修改配置项。

   新增配置项的具体操作，请参见添加组件参数。修改配置项的具体操作，请参见修改组件参数。

内容	参数	描述
配置OSS实现类	fs.AbstractFileSystem.oss.impl	固定值为com.aliyun.jindodata.oss.OSS。
	fs.jindo.impl	固定值为com.aliyun.jindodata.oss.JindoOssFileSystem。
	fs.xengine	固定值为jindofsx。
配置OSS Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。

内容	参数	描述
开启透明缓存加速（可选） 注意 如果使用缓存功能，请执行步骤1。	fs.jindofsx.data.cache.enable	数据缓存开关： false（默认值）：禁用数据缓存。 true：启用数据缓存。 说明 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。

4. 重启Hive服务，具体操作请参见重启服务。
5. 指定OSS存储。
   - 在创建数据库和表时，可以指定OSS路径，把数据库或表的数据默认保存到OSS上，示例如下。

     ```
     CREATE DATABASE db_on_oss1 LOCATION 'oss://bucketname/path/to/db1';
     CREATE TABLE db2.table_on_oss ... LOCATION 'oss://bucketname/path/to/db2/tablepath';
     ```

   - 在Hive Metastore的*hive-site.xml*配置中设置参数hive.metastore.warehouse.dir到OSS路径，并重启Hive Metastore，则后续创建的数据库和这些数据库下的表都会默认存储于OSS。

     ```
     <configuration>
         <property>
             <name>hive.metastore.warehouse.dir</name>
             <value>oss://bucketname/path/to/warehouse</value>
         </property>
     </configuration>
     ```

   - 给已有表添加OSS的分区。

     ```
     ALTER TABLE existed_table ADD PARTITION (dt='2021-03-01', country='cn') LOCATION 'oss://bucketname/path/to/us/part210
     301cn';
     ```

6.4.2.3.8. Hive处理阿里云OSS-HDFS上的数据+JindoFSx透明加速

JindoFSx存储加速系统提供了透明缓存的使用方式，兼容原生OSS存储方式，文件以对象的形式存储在OSS-HDFS上，每个文件根据实际访问情况会在本地进行缓存，提升访问OSS-HDFS的效率，同时兼容了原有OSS文件形式。本文为您介绍Hive如何处理OSS-HDFS服务（JindoFS服务）中的数据。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

说明 本文以EMR-3.40.0版本为例介绍。

操作步骤

1. （可选）如果想使用缓存功能，请先配置JindoData服务的AccessKey信息，具体操作请参见步骤一：配置AccessKey。
2. 进入HDFS服务的core-site页签。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务 > HDFS**。
   vi. 在HDFS服务页面，单击**配置**页签。
   vii. 在**服务配置**区域，单击**core-site**页签。
3. 新增和修改配置项。

   新增配置项的具体操作，请参见添加组件参数。修改配置项的具体操作，请参见修改组件参数。

内容	参数	描述
配置实现类	fs.AbstractFileSystem.oss.impl	固定值为com.aliyun.jindodata.oss.OSS。
	fs.oss.impl	固定值为com.aliyun.jindodata.oss.JindoOssFileSystem。

内容	参数	描述
	fs.xengine	固定值为jindofsx。
配置OSS Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	OSS-HDFS服务的Endpoint。例如，cn-***.oss-dls.aliyuncs.com。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
开启透明缓存加速（可选） **注意** 如果使用缓存功能，请执行步骤1。	fs.jindofsx.data.cache.enable	数据缓存开关： false（默认值）：禁用数据缓存。 true：启用数据缓存。 **说明** 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。

4. 重启Hive服务，具体操作请参见重启服务。
5. 指定OSS-HDFS服务路径。
   - 在创建数据库和表时，可以指定OSS-HDFS服务的路径，把数据库或表的数据默认保存到OSS-HDFS服务上，示例如下。

     ```
     CREATE DATABASE db_on_oss1 LOCATION 'oss://bucket_name/path/to/db1';
     CREATE TABLE db2.table_on_oss ... LOCATION 'oss://bucket_name/path/to/db2/tablepath';
     ```

   - 在Hive Metastore的*hive-site.xml*配置中设置参数hive.metastore.warehouse.dir到OSS-HDFS服务路径，并重启Hive Metastore，则后续创建的数据库和这些数据库下的表都会默认存储于OSS-HDFS服务。

     ```
     <configuration>
         <property>
             <name>hive.metastore.warehouse.dir</name>
             <value>oss://bucket_name/path/to/warehouse</value>
         </property>
     </configuration>
     ```

   - 给已有表添加OSS的分区。

     ```
     ALTER TABLE existed_table ADD PARTITION (dt='2021-03-01', country='cn') LOCATION 'oss://bucket_name/path/to/us/part210301cn';
     ```

6.4.2.3.9. Hive处理JindoFSx统一挂载的数据

Hive是大数据的常用工具之一，使用Hive可以搭建离线数仓。随着数据量不断增长，传统的基于HDFS存储的数仓可能无法以较低成本满足用户的需求，常见的可以结合对象存储等云存储使用Hive。本文为您介绍Hive如何处理JindoFSx统一挂载的数据。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

说明 本文以EMR-3.40.0版本为例介绍。

操作步骤

1. （可选）如果想使用缓存功能，请先配置JindoData服务的AccessKey信息，具体操作请参见步骤一：配置AccessKey。
2. 进入HDFS服务的core-site页签。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务** > **HDFS**。

vi. 在HDFS服务页面，单击**配置**页签。

vii. 在**服务配置**区域，单击**core-site**页签。

3. 新增和修改配置项。

新增配置项的具体操作，请参见添加组件参数。修改配置项的具体操作，请参见修改组件参数。

内容	参数	描述
配置实现类	fs.AbstractFileSystem.jindo.impl	固定值为com.aliyun.jindodata.jindo.JINDO。
	fs.jindo.impl	固定值为com.aliyun.jindodata.jindo.JindoFileSystem。
	fs.xengine	固定值为jindofsx。
配置Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	○ OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。 ○ OSS-HDFS服务的Endpoint。例如，cn-***.oss-dls.aliyuncs.com。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
缓存加速功能（可选） **注意** 如果使用缓存功能，请执行步骤1。	fs.jindofsx.data.cache.enable	数据缓存开关： ○ false（默认值）：禁用数据缓存。 ○ true：启用数据缓存。 **说明** 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。
	fs.jindofsx.meta.cache.enable	元数据缓存开关： ○ false（默认值）：禁用元数据缓存。 ○ true：启用元数据缓存。
	fs.jindofsx.slice.cache.enable	小文件缓存优化开关： ○ false（默认值）：禁用小文件缓存。 ○ true：启用小文件缓存。
	fs.jindofsx.ram.cache.enable	内存缓存开关： ○ false（默认值）：禁用内存缓存。 ○ true：启用内存缓存。
	fs.jindofsx.short.circuit.enable	短路读开关： ○ true（默认值）：打开。 ○ false：关闭。

4. 重启Hive服务，具体操作请参见重启服务。
5. 挂载OSS或OSS-HDFS服务目录。

挂载语法如下。

```
jindo admin -mount <path> <realpath>
```

例如，执行以下命令挂载OSS-HDFS服务目录。

```
jindo admin -mount /jindodls oss://<yourBucketName>.<yourBucketEndpoint>/
```

说明 执行上述命令后，则 `/jindodls` 目录真正挂载的文件路径是 `oss://<yourBucketName>.<yourBucketEndpoint>/` 。

6. 指定存储路径。
   - 在创建数据库和表时，可以指定OSS路径，把数据库或表的数据默认保存到OSS上，示例如下。

     ```
     CREATE DATABASE db_on_oss1 LOCATION 'jindo://headerhost:8101/jindooss/path/to/db1';
     CREATE TABLE db2.table_on_oss ... LOCATION 'jindo://headerhost:8101/jindooss/path/to/db2/tablepath';
     ```

   - 在Hive Metastore的*hive-site.xml*配置中设置参数hive.metastore.warehouse.dir到OSS路径，并重启Hive Metastore，则后续创建的数据库和这些数据库下的表都会默认存储于OSS。

     ```
     <configuration>
         <property>
             <name>hive.metastore.warehouse.dir</name>
             <value>oss://bucket_name/path/to/warehouse</value>
         </property>
     </configuration>
     ```

   - 给已有表添加OSS的分区。

     ```
     ALTER TABLE existed_table ADD PARTITION (dt='2021-03-01', country='cn') LOCATION 'jindo://headerhost:8101/jindooss/path/to/us/part210301cn';
     ```

6.4.2.3.10. Presto查询阿里云OSS上数据+JindoFSx透明加速

Presto是一个开源的分布式SQL查询引擎，适用于交互式分析查询。本文为您介绍Presto如何使用JindoSDK查询阿里云OSS数据湖存储。

背景信息

JindoSDK包含一些高级调优参数，配置方式以及配置项请参见JindoSDK高级参数配置。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> **说明** 本文以EMR-3.40.0版本为例介绍。

操作步骤

1. 配置JindoData服务的AccessKey信息，具体操作请参见步骤一：配置AccessKey。
2. 进入HDFS服务的**core-site**页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 在**集群服务**页签，单击HDFS服务区域的**配置**。
   v. 在HDFS服务的**服务配置**区域，单击**core-site**页签。
3. 新增和修改配置项。

   新增配置项的具体操作，请参见添加组件参数。修改配置项的具体操作，请参见修改组件参数。

内容	参数	描述
配置JindoSDK OSS实现类	fs.AbstractFileSystem.oss.impl	固定值为com.aliyun.jindodata.oss.OSS。
	fs.oss.impl	固定值为com.aliyun.jindodata.oss.JindoOssFileSystem。
	fs.xengine	固定值为jindofsx。
配置OSS AccessKey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。

内容	参数	描述
开启JindoFSx透明加速缓存功能	fs.jindofsx.data.cache.enable	数据缓存开关： ○ false（默认值）：禁用数据缓存。 ○ true：启用数据缓存。 **说明** 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。

4. 重启Presto服务，具体操作请参见重启服务。

使用示例

以下以最常用的Hive Catalog为例，使用Presto创建一个OSS上的Schema，并执行一些简单的SQL示例。

1. 执行以下命令，启动Presto客户端。

```
presto --server emr-header-1:9090 --catalog hive
```

2. 执行以下命令，创建并使用一个Location位于OSS上的Schema。

```
create schema testDB with (location='oss://<bucket>/<schema_dir>');
use testDB;
```

3. 执行以下命令，创建表。

```
create table tbl (key int, val int);
```

4. 执行以下命令，向创建的表中插入数据。

```
insert into tbl values (1,666);
```

5. 执行以下命令，查询表数据。

```
select * from tbl;
```

6.4.2.3.11. Presto使用JindoSDK查询阿里云OSS-HDFS服务上的数据

Presto是一个开源的分布式SQL查询引擎，适用于交互式分析查询。本文为您介绍Presto如何使用JindoSDK查询OSS-HDFS服务（JindoFS服务）中的数据。

背景信息

JindoSDK包含一些高级调优参数，配置方式以及配置项请参见JindoSDK高级参数配置。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

说明 本文以EMR-3.40.0版本为例介绍。

步骤一：配置AccessKey

1. 进入新增配置项页面。
   i. 进入JindoData服务页面。
      a. 登录阿里云E-MapReduce控制台。
      b. 在顶部菜单栏处，根据实际情况选择地域和资源组。
      c. 单击上方的**集群管理**页签。
      d. 在**集群管理**页面，单击相应集群所在行的**详情**。
      e. 在左侧导航栏，选择**集群服务 > JindoData**。
   ii. 在JindoData服务页面，单击**配置**页签。
   iii. 在**服务配置**区域，单击**common**页签。
2. 新增配置。
   i. 单击**自定义配置**。

ii. 在**新增配置项**对话框中，新增以下配置项。

新增配置项的具体操作，请参见添加组件参数。

- 全局方式配置（所有Bucket使用同一种方式）

参数	描述
jindofsx.oss.accessKeyId	OSS的AccessKey ID。
jindofsx.oss.accessKeySecret	OSS的AccessKey Secret。
jindofsx.oss.endpoint	OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。

- 按照Bucket配置

参数	描述
jindofsx.oss.bucket.XXX.accessKeyId	`XXX`的Bucket的AccessKey ID。
jindofsx.oss.bucket.XXX.accessKeySecret	`XXX`的Bucket的AccessKey Secret。
jindofsx.oss.bucket.XXX.endpoint	`XXX`的Bucket的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
jindofsx.oss.bucket.YYY.accessKeyId	`YYY`的Bucket的AccessKey ID。
jindofsx.oss.bucket.YYY.accessKeySecret	`YYY`的Bucket的AccessKey Secret。
jindofsx.oss.bucket.YYY.endpoint	`YYY`的Bucket的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。

> **说明** `XXX`和`YYY`为OSS Bucket的名称。

iii. 单击**确定**。

3. 重启服务。
   i. 在JindoData服务页面，选择右上角的**操作 > 重启All Components**。
   ii. 在**执行集群操作**对话框中，输入执行原因（其他参数保持默认），单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

步骤二：配置JindoSDK

1. 进入HDFS服务的**core-site**页签。
   i. 登录EMR on ECS控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击目标集群操作列的**集群服务**。
   iv. 在**集群服务**页签，单击HDFS服务区域的**配置**。
   v. 在HDFS服务的**服务配置**区域，单击**core-site**页签。
2. 新增和修改配置项。

   新增配置项的具体操作，请参见添加组件参数。修改配置项的具体操作，请参见修改组件参数。

内容	参数	描述
配置实现类	fs.AbstractFileSystem.oss.impl	固定值为com.aliyun.jindodata.oss.OSS。
	fs.oss.impl	固定值为com.aliyun.jindodata.oss.JindoOssFileSystem。
	fs.xengine	固定值为jindofsx。
配置AccessKey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	OSS-HDFS服务的Endpoint。例如，cn-***.oss-dls.aliyuncs.com。

内容	参数	描述
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 说明 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
开启缓存加速	fs.jindofsx.data.cache.enable	数据缓存开关： false（默认值）：禁用数据缓存。 true：启用数据缓存。 说明 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。
	fs.jindofsx.meta.cache.enable	元数据缓存开关： false（默认值）：禁用元数据缓存。 true：启用元数据缓存。
	fs.jindofsx.slice.cache.enable	小文件缓存优化开关： false（默认值）：禁用小文件缓存。 true：启用小文件缓存。
	fs.jindofsx.ram.cache.enable	内存缓存开关： false（默认值）：禁用内存缓存。 true：启用内存缓存。
	fs.jindofsx.short.circuit.enable	短路读开关： true（默认值）：打开。 false：关闭。

3. 重启Presto服务，具体操作请参见重启服务。

使用示例

以下以最常用的Hive Catalog为例，使用Presto创建一个OSS上的Schema，并执行一些简单的SQL示例。

1. 执行以下命令，启动Presto客户端。

```
presto --server emr-header-1:9090 --catalog hive
```

2. 执行以下命令，创建并使用一个Location位于OSS上的Schema。

```
create schema testDB with (location='oss://<bucket>/<schema_dir>');
use testDB;
```

3. 执行以下命令，创建表。

```
create table tbl (key int, val int);
```

4. 执行以下命令，向创建的表中插入数据。

```
insert into tbl values (1,666);
```

5. 执行以下命令，查询表数据。

```
select * from tbl;
```

6.4.2.3.12. Presto处理JindoFSx统一挂载的数据

Presto是一个开源的分布式SQL查询引擎，适用于交互式分析查询。本文为您介绍Presto如何处理JindoFSx统一挂载的数据。

背景信息

JindoSDK包含一些高级调优参数，配置方式以及配置项请参见JindoSDK高级参数配置。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> **说明** 本文以EMR-3.40.0版本为例介绍。

操作步骤

1. （可选）如果想使用缓存功能，请先配置JindoData服务的AccessKey信息，具体操作请参见步骤一：配置AccessKey。
2. 进入HDFS服务的core-site页签。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务 > HDFS**。
   vi. 在HDFS服务页面，单击**配置**页签。
   vii. 在**服务配置**区域，单击**core-site**页签。
3. 新增和修改配置项。

   新增配置项的具体操作，请参见添加组件参数。修改配置项的具体操作，请参见修改组件参数。

内容	参数	描述
配置实现类	fs.AbstractFileSystem.jindo.impl	固定值为com.aliyun.jindodata.jindo.JINDO。
	fs.jindo.impl	固定值为com.aliyun.jindodata.jindo.JindoFileSystem。
	fs.xengine	固定值为jindofsx。
配置Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	○ OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。 ○ OSS-HDFS服务的Endpoint。例如，cn-***.oss-dls.aliyuncs.com。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
缓存加速功能（可选） **注意** 如果使用缓存功能，请执行步骤1。	fs.jindofsx.data.cache.enable	数据缓存开关： ○ false（默认值）：禁用数据缓存。 ○ true：启用数据缓存。 **说明** 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。
	fs.jindofsx.meta.cache.enable	元数据缓存开关： ○ false（默认值）：禁用元数据缓存。 ○ true：启用元数据缓存。
	fs.jindofsx.slice.cache.enable	小文件缓存优化开关： ○ false（默认值）：禁用小文件缓存。 ○ true：启用小文件缓存。
	fs.jindofsx.ram.cache.enable	内存缓存开关： ○ false（默认值）：禁用内存缓存。 ○ true：启用内存缓存。

内容	参数	描述
	fs.jindofsx.short.circuit.enable	短路读开关： ○ true（默认值）：打开。 ○ false：关闭。

4. 重启Presto服务，具体操作请参见重启服务。
5. 挂载OSS或OSS-HDFS服务目录。

   挂载语法如下。

   ```
   jindo admin -mount <path> <realpath>
   ```

   例如，执行以下命令挂载OSS-HDFS服务目录。

   ```
   jindo admin -mount /jindodls oss://<yourBucketName>.<yourBucketEndpoint>/
   ```

   > **说明** 执行上述命令后，则 `/jindodls` 目录真正挂载的文件路径是 `oss://<yourBucketName>.<yourBucketEndpoint>/` 。

使用示例

以下以最常用的Hive Catalog为例，使用Presto创建一个OSS上的Schema，并执行一些简单的SQL示例。

1. 执行以下命令，启动Presto客户端。

   ```
   presto --server emr-header-1:9090 --catalog hive
   ```

2. 执行以下命令，创建并使用一个Location位于OSS上的Schema。

   ```
   create schema testDB with (location='oss://<bucket>/<schema_dir>');
   use testDB;
   ```

3. 执行以下命令，创建表。

   ```
   create table tbl (key int, val int);
   ```

4. 执行以下命令，向创建的表中插入数据。

   ```
   insert into tbl values (1,666);
   ```

5. 执行以下命令，查询表数据。

   ```
   select * from tbl;
   ```

6.4.2.3.13. Impala处理阿里云OSS上的数据+JindoFSx透明加速

JindoFSx存储加速系统提供了透明缓存的使用方式，兼容原生OSS存储方式，文件以对象的形式存储在OSS上，每个文件根据实际访问情况会在本地进行缓存，提升访问OSS的效率，同时兼容了原有OSS文件形式，数据访问上能够与其他OSS客户端完全兼容，作业访问OSS的方式无需做任何修改。本文为您介绍Impala如何处理阿里云OSS上的数据。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> **说明** 本文以EMR-3.40.0版本为例介绍。

操作步骤

1. （可选）如果想使用缓存功能，请先配置JindoData服务的AccessKey信息，具体操作请参见步骤一：配置AccessKey。
2. 进入HDFS服务的core-site页签。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务** > **HDFS**。
   vi. 在HDFS服务页面，单击**配置**页签。
   vii. 在**服务配置**区域，单击**core-site**页签。

3. 配置JindoSDK。

   新增配置项的具体操作，请参见添加组件参数。修改配置项的具体操作，请参见修改组件参数。

内容	参数	描述
配置JindoSDK OSS实现类	fs.AbstractFileSystem.oss.impl	固定值为com.aliyun.jindodata.oss.OSS。
	fs.oss.impl	固定值为com.aliyun.jindodata.oss.JindoOssFileSystem。
	fs.xengine	固定值为jindofsx。
配置OSS Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 说明 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
开启透明缓存加速（可选） 注意 如果使用缓存功能，请执行步骤1。	fs.jindofsx.data.cache.enable	数据缓存开关： false（默认值）：禁用数据缓存。 true：启用数据缓存。 说明 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。

4. 使用Impala访问OSS。
   - 创建表

```
CREATE EXTERNAL TABLE customer_demographics (
 `cd_demo_sk` INT,
 `cd_gender` STRING,
 `cd_marital_status` STRING,
 `cd_education_status` STRING,
 `cd_purchase_estimate` INT,
 `cd_credit_rating` STRING,
 `cd_dep_count` INT,
 `cd_dep_employed_count` INT,
 `cd_dep_college_count` INT)
STORED AS PARQUET
LOCATION 'oss://bucket.endpoint/dir';
```

   说明 示例中的LOCATION需要替换为OSS的实际路径。

   - 查询OSS表

```
select * from customer_demographics;
```

6.4.2.3.14. Impala处理JindoFSx统一挂载的数据

本文为您介绍Impala如何处理JindoFSx统一挂载的数据。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

说明 本文以EMR-3.40.0版本为例介绍。

操作步骤

1. （可选）如果想使用缓存功能，请先配置JindoData服务的AccessKey信息，具体操作请参见步骤一：配置AccessKey。
2. 进入HDFS服务的core-site页签。

i. 登录阿里云E-MapReduce控制台。
ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
iii. 单击上方的**集群管理**页签。
iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
v. 在左侧导航栏，选择**集群服务 > HDFS**。
vi. 在HDFS服务页面，单击**配置**页签。
vii. 在**服务配置**区域，单击**core-site**页签。

3. 配置JindoSDK。

新增配置项的具体操作，请参见添加组件参数。修改配置项的具体操作，请参见修改组件参数。

内容	参数	描述
配置实现类	fs.AbstractFileSystem.jindo.impl	固定值为com.aliyun.jindodata.jindo.JINDO。
	fs.jindo.impl	固定值为com.aliyun.jindodata.jindo.JindoFileSystem。
	fs.xengine	固定值为jindofsx。
配置Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	○ OSS的Endpoint。例如，oss-cn-***-internal.aliyuncs.com。 ○ OSS-HDFS服务的Endpoint。例如，cn-***.oss-dls.aliyuncs.com。
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
缓存加速功能（可选） **注意** 如果使用缓存功能，请执行步骤1。	fs.jindofsx.data.cache.enable	数据缓存开关： ○ false（默认值）：禁用数据缓存。 ○ true：启用数据缓存。 **说明** 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。
	fs.jindofsx.meta.cache.enable	元数据缓存开关： ○ false（默认值）：禁用元数据缓存。 ○ true：启用元数据缓存。
	fs.jindofsx.slice.cache.enable	小文件缓存优化开关： ○ false（默认值）：禁用小文件缓存。 ○ true：启用小文件缓存。
	fs.jindofsx.ram.cache.enable	内存缓存开关： ○ false（默认值）：禁用内存缓存。 ○ true：启用内存缓存。
	fs.jindofsx.short.circuit.enable	短路读开关： ○ true（默认值）：打开。 ○ false：关闭。

4. 挂载OSS或OSS-HDFS服务目录。

挂载语法如下。

```
jindo admin -mount <path> <realpath>
```

例如，执行以下命令挂载OSS目录。

```
jindo admin -mount /jindooss oss://<yourBucketName>.<yourBucketEndpoint>/
```

> **说明** 执行上述命令后，则 `/jindooss` 目录真正挂载的文件路径是 `oss://<yourBucketName>.<yourBucketEndpoint>/` 。

5. 使用Impala访问OSS。
   - 创建表

```
CREATE EXTERNAL TABLE customer_demographics (
 `cd_demo_sk` INT,
 `cd_gender` STRING,
 `cd_marital_status` STRING,
 `cd_education_status` STRING,
 `cd_purchase_estimate` INT,
 `cd_credit_rating` STRING,
 `cd_dep_count` INT,
 `cd_dep_employed_count` INT,
 `cd_dep_college_count` INT)
STORED AS PARQUET
LOCATION 'jindo://headerhost:8101/jindooss/dir';
```

   - 查询OSS表

```
select * from customer_demographics;
```

6.4.2.3.15. Impala处理阿里云OSS-HDFS上的数据+JindoFSx透明加速

JindoFSx存储加速系统提供了透明缓存的使用方式，兼容原生OSS存储方式，文件以对象的形式存储在OSS-HDFS上，每个文件根据实际访问情况会在本地进行缓存，提升访问OSS-HDFS的效率，同时兼容了原有OSS文件形式。本文为您介绍Impala如何使用JindoSDK处理阿里云OSS-HDFS（JindoFS服务）上的数据。

前提条件

已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

> **说明** 本文以EMR-3.40.0版本为例介绍。

操作步骤

1. （可选）如果想使用缓存功能，请先配置JindoData服务的AccessKey信息，具体操作请参见步骤一：配置AccessKey。
2. 进入HDFS服务的core-site页签。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 单击上方的**集群管理**页签。
   iv. 在**集群管理**页面，单击相应集群所在行的**详情**。
   v. 在左侧导航栏，选择**集群服务 > HDFS**。
   vi. 在HDFS服务页面，单击**配置**页签。
   vii. 在**服务配置**区域，单击**core-site**页签。
3. 新增和修改配置项。

   新增配置项的具体操作，请参见添加组件参数。修改配置项的具体操作，请参见修改组件参数。

内容	参数	描述
配置实现类	fs.AbstractFileSystem.oss.impl	固定值为com.aliyun.jindodata.oss.OSS。
	fs.oss.impl	固定值为com.aliyun.jindodata.oss.JindoOssFileSystem。
	fs.xengine	固定值为jindofsx。
配置OSS Accesskey	fs.oss.accessKeyId	OSS的AccessKey ID。
	fs.oss.accessKeySecret	OSS的AccessKey Secret。
	fs.oss.endpoint	OSS-HDFS服务的Endpoint。例如，cn-***.oss-dls.aliyuncs.com。

内容	参数	描述
配置JindoFSx Namespace服务地址	fs.jindofsx.namespace.rpc.address	格式为${headerhost}:8101。例如，emr-header-1:8101。 **说明** 如果使用高可用NameSpace，配置详情请参见高可用JindoFSx Namespace配置和使用。
开启透明缓存加速（可选） **注意** 如果使用缓存功能，请执行步骤1。	fs.jindofsx.data.cache.enable	数据缓存开关： false（默认值）：禁用数据缓存。 true：启用数据缓存。 **说明** 启用缓存会利用本地磁盘对访问的热数据块进行缓存，默认状态为false，即可以直接访问OSS上的数据。

4. 使用Impala访问OSS。
   - 创建表

```
CREATE EXTERNAL TABLE customer_demographics (
 `cd_demo_sk` INT,
 `cd_gender` STRING,
 `cd_marital_status` STRING,
 `cd_education_status` STRING,
 `cd_purchase_estimate` INT,
 `cd_credit_rating` STRING,
 `cd_dep_count` INT,
 `cd_dep_employed_count` INT,
 `cd_dep_college_count` INT)
STORED AS PARQUET
LOCATION 'oss://bucket.endpoint/dir';
```

   - 查询OSS表

```
select * from customer_demographics;
```

6.4.2.3.16. 切换为Hadoop原生的JobCommitter

E-MapReduce（简称EMR）集群默认使用JindoCommitter加速大数据作业，解决OSS等对象存储在Spark、MapReduce等作业使用原生Hadoop JobCommitter时遇到的性能和一致性等问题。如果您不想使用默认的JindoCommitter，则可以参照本文切换为Hadoop原生的JobCommitter。本文为您介绍如何切换为Hadoop原生的JobCommitter。

前提条件

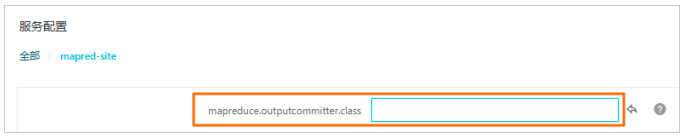
已在E-MapReduce上创建EMR-3.40.0及后续版本，EMR-5.6.0及后续版本的集群，具体操作请参见创建集群。

说明 本文以EMR-3.40.0版本为例介绍。

步骤一：修改YARN配置

1. 进入YARN服务的**mapred-site**页签。
   i. 登录阿里云E-MapReduce控制台。
   ii. 在顶部菜单栏处，根据实际情况选择地域和资源组。
   iii. 在**集群管理**页面，单击相应集群所在行的**详情**。
   iv. 在左侧导航栏，选择**集群服务 > YARN**。
   v. 在YARN服务页面，单击**配置**页签。
   vi. 在**服务配置**区域，单击**mapred-site**页签。
2. 修改配置项。

版本	参数	描述

版本	参数	描述
EMR Hadoop 2.x版本	mapreduce.outputcommitter.class	删除参数值，将参数值置为空。 例如，搜索mapreduce.outputcommitter.class配置，删除参数值。 
EMR Hadoop 3.x版本	mapreduce.outputcommitter.factory.class	删除参数值，将参数值置为空。

3. 保存配置。
   i. 单击右上角的**保存**。
   ii. 在**确认修改**对话框中，输入**执行原因**，开启**自动更新配置**。
   iii. 单击**确定**。
4. 重启YARN服务，具体操作请参见重启服务。

步骤二：修改Spark配置

1. 进入Spark服务的**spark-defaults**页签。
   i. 在左侧导航栏，选择**集群服务 > Spark**。
   ii. 在Spark服务页面，单击**配置**页签。
   iii. 在**服务配置**区域，单击**spark-defaults**页签。
2. 修改配置项。

参数	描述
spark.sql.sources.outputCommitterClass	删除参数值，将参数值置为空。
spark.sql.parquet.output.committer.class	固定值为org.apache.parquet.hadoop.ParquetOutputCommitter。

3. （可选）如果您集群的Spark是2.x版本，则还需在**spark-defaults**页签新增以下配置项。
   i. 在Spark**服务配置**区域，单击右侧的**自定义配置**。
   ii. 新增参数为spark.sql.hive.commitProtocolClass，参数值为org.apache.spark.sql.execution.datasources.SQLHadoopMapReduceCommitProtocol的配置项。
   iii. 单击**确定**。
4. 保存配置。
   i. 单击右上角的**保存**。
   ii. 在**确认修改**对话框中，输入**执行原因**，开启**自动更新配置**。
   iii. 单击**确定**。
5. 重启Spark服务，具体操作请参见重启服务。

6.4.3. JindoData 4.0.0版本简介

JindoData 4.x是阿里云E-MapReduce产品SmartData自研组件（SmartData 3.8.0版本）架构升级之后首次发布的版本，重点对接和支持了阿里云OSS存储产品和阿里云OSS-HDFS服务（JindoFS服务）。本文为您介绍JindoData 4.0.0版本支持的功能。

使用限制

- JindoSDK暂不支持OSS上超大文件写入（大于80 GB）。
- JindoSDK暂不支持OSS append方式写入。
- JindoSDK暂不支持OSS客户端加密。
- JindoSDK暂不支持老版本JindoFS Block模式和Cache模式。
- 阿里云OSS-HDFS服务（JindoFS服务）暂不支持老版本JindoFS Block模式系统升级。需要用户通过JindoDistCp迁移工具把数据从老系统迁移到新服务。
- JindoData 4.0.0版本暂未发布JindoFSx缓存系统。

JindoSDK Hadoop

- 为阿里云OSS提供了Java Hadoop SDK，完全兼容Hadoop OSS Connector，性能上大幅领先。
- 支持多种Credential Provider设置方式，包括配置、ECS Role和EMR免密机制。
- 支持写入时归档，包括归档和深冷归档。

JindoShell CLI

- 对Hadoop或HDFS Shell提供额外的命令扩展，为阿里云OSS提供面向Hadoop用户使用的操作方式。
- 支持ls2扩展命令，在标准ls命令的基础上可以额外显示文件或对象在OSS上的存储状态。例如，标准、低频还是归档。
- 支持archive命令，允许指定目录进行转归档操作。
- 支持restore命令，允许指定目录进行解冻操作。

JindoFuse POSIX

为阿里云OSS提供优化后的Fuse客户端，受益于完全Native代码的开发实现，性能在业界大幅领先。

JindoDistCp数据迁移

支持将自建HDFS集群数据迁移到阿里云OSS，针对大文件和大量小文件场景优化。

OSS-HDFS服务

- JindoSDK内置支持访问阿里OSS-HDFS服务（JindoFS服务），提供全面的HDFS接口访问和使用体验。
- 提供Hadoop或HDFS Shell额外的命令扩展，为阿里云OSS-HDFS服务（JindoFS服务）提供面向Hadoop用户使用的操作方式。
- 支持通过HDFS命令和JindoShell扩展命令使用HDFS快照功能，详情请参见管理OSS-HDFS服务快照。
- 支持使用命令导入（UserGroupsMapping），设定用户组信息。
- 支持使用命令设定Hadoop Proxy User规则。

JindoSDK

- 使用JindoSDK访问OSS，详情请参见Hadoop使用JindoSDK访问OSS。
- 使用JindoSDK访问OSS-HDFS服务，详情请参见OSS-HDFS服务快速入门。

7.EMR on ACK

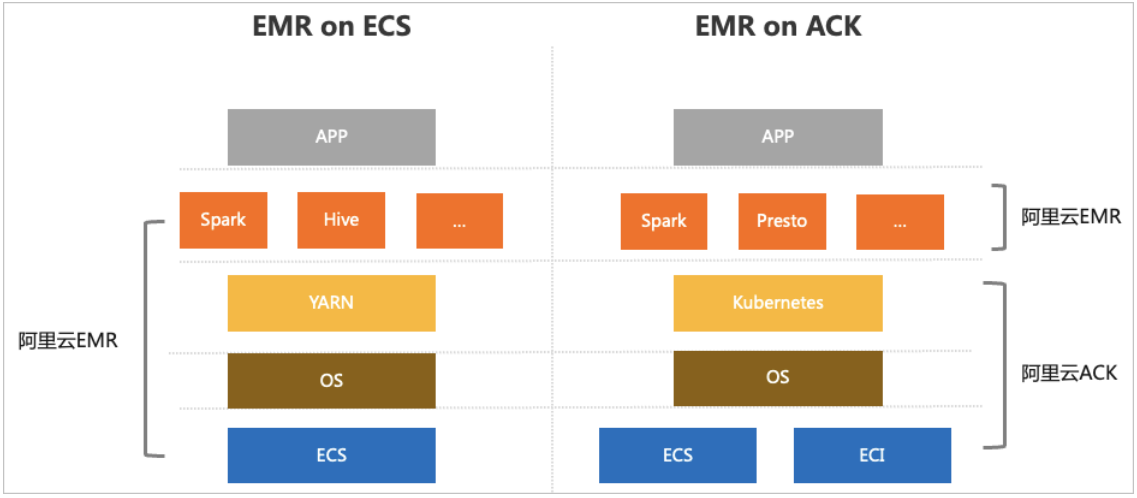
7.1. EMR on ACK概述

阿里云E-MapReduce（简称EMR） on ACK提供了全新构建大数据平台的方式。您可以将开源大数据服务部署在阿里云容器服务Kubernetes版（ACK）之上，利用ACK在服务部署和容器应用管理的优势，减少对底层集群资源的运维投入，以便于您可以更加专注大数据任务本身。

形态对比

阿里云EMR提供on ECS和on ACK两种方式，以满足不同用户的需求。

- 对于正在使用EMR on ECS的用户，可以将Spark和Presto任务运行在ACK集群上，与其他应用共享一个ACK集群，可以实现计算资源跨可用区共享。
- 对于已经将大数据任务（例如，Spark和Presto等）执行在ACK集群上的用户，EMR on ACK提供了自动部署和管理集群的能力。EMR on ACK与EMR Shuffle Service相结合，可以显著提升Spark任务的性能。



形态	描述
EMR on ECS	EMR负责将开源Hadoop生态的组件安装部署在ECS上，并启动相应的服务。您可以在EMR控制台完成对集群ECS及服务的运维操作。 您需要将其大数据任务提交至EMR集群。
EMR on ACK	您需要先完成ACK集群的安装部署。当ACK集群准备就绪后，EMR将基于ACK的资源安装部署大数据服务组件，并在容器内运行。

EMR on ACK优势

优势	描述
节省成本	您无需为大数据服务单独购买ACK集群，通过简单的配置即可在已有的ACK集群上执行大数据作业，成本低廉。 复用现有ACK集群的空闲资源，一键执行EMR Spark和Presto等任务，轻松上手。大数据和在线应用程序可以共享集群资源。 离在线混部（在线任务和离线任务）场景下，资源可以充分利用。大数据和在线应用程序共享集群资源，达到削峰填谷的效果。
简化运维	一套运维体系，一套集群管理，全面覆盖大数据和在线等多种业务，简化运维。
优化体验	一套EMR平台，同时支持ECS和ACK两套IaaS资源模型，您可以无缝切换。 利用ACK和弹性容器实例ECI的资源快速交付能力，弹性计算资源的获取时间更短，充分应对计算高峰期。 支持针对作业级别调整Spark版本，便于快速尝试新特性，以满足不同业务对版本的需求。
深度集成	完全采用云原生数据湖架构，计算使用阿里云ACK，计算资源可以无限扩展；存储使用阿里云OSS，存储计算分离；元数据使用数据湖构建DLF，助力数据湖构建。

7.2. 准备工作

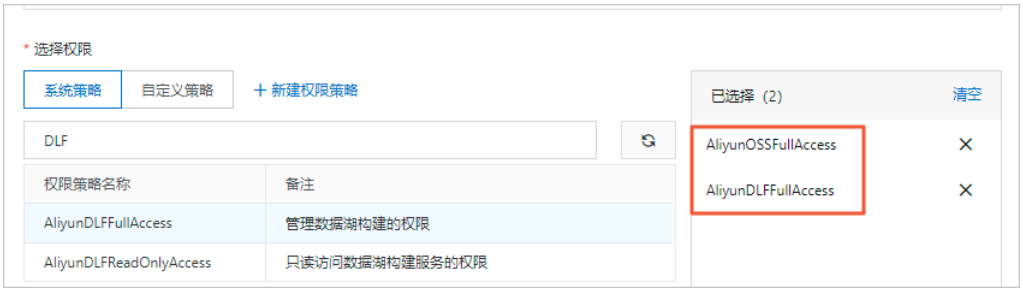
7.2.1. 角色授权

本文为您介绍添加AliyunOSSFullAccess和AliyunDLFFullAccess权限，以便于您可以使用DLF服务，以及授权组件免AccessKey访问OSS。

操作步骤

1. 登录容器服务管理控制台。
2. 在**集群列表**页面，单击集群**操作**列的**详情**。
3. 在**集群信息**页面，单击**集群资源**页签。
4. 在**集群资源**页面，单击**Worker RAM角色**所在行的链接。

5. 添加权限。
   i. 在**角色**页面，单击**添加权限**。
   ii. 在**添加权限**页面，选择并添加系统策略**AliyunOSSFullAccess**和**AliyunDLFFullAccess**。



   iii. 单击**确定**。
   iv. 单击**完成**。

7.3. 快速入门

本文为您介绍如何通过阿里云账号登录E-MapReduce控制台，基于Kubernetes创建E-MapReduce（简称EMR）集群并执行作业。

背景信息

节点池相关的信息，请参见节点池概述。

前提条件

- 已完成添加AliyunOSSFullAccess和AliyunDLFFullAccess权限，详情请参见角色授权。
- 已创建Kubernetes集群，详情请参见创建Kubernetes专有版集群或创建Kubernetes托管版集群。
- 已创建节点池，详情请参见创建节点池。
- 已开通对象存储OSS，详情请参见开通OSS服务。

使用限制

Flink集群类型目前是白名单模式，如果您想使用Flink集群请提交工单申请。

操作流程

1. 步骤一：创建集群

   基于Kubernetes创建一个EMR集群。

2. 步骤二：提交作业

   集群创建成功后，您可以提交作业，本文以通过CRD方式提交Spark作业为例介绍。

3. （可选）步骤三：释放集群

   如果不再使用该集群，可以释放集群以节约成本。

步骤一：创建集群

1. 登录阿里云E-MapReduce on ACK控制台。
2. 在**集群管理**页面，单击**创建集群**。
3. 在**E-MapReduce on ACK**页面，完成集群相关配置。

参数	示例	描述
地域	华东1（杭州）	创建的集群将会在对应的地域内，一旦创建不能修改。
集群类型	Spark	支持以下集群类型： • **Spark**：是通用的分布式大数据处理引擎，提供了ETL、离线批处理和数据建模等能力。 **注意** 创建Spark集群后，如果您需要关联集群，则所选产品版本的大版本号需要和关联的Shuffle Service集群大版本号一致。例如，EMR-5.x-ack版本的Spark集群只能关联EMR-5.x-ack版本的Shuffle Service集群。 • **Shuffle Service**：是阿里云EMR在优化计算引擎的Shuffle操作上，推出的扩展组件。Shuffle Service通过提供远端Shuffle服务，使得Spark作业可以运行在无本地盘的节点上，并完美支持了动态资源，非常适合ACK环境下的Spark集群，详情请参见ESS。 **注意** 当创建Shuffle Service集群类型时，新建的Kubernetes集群或者选择的已有的Kubernetes集群的实例规格必须是大数据型或者本地SSD，否则部署RSS失败。  • **Presto**：是基于内存的分布式SQL交互式查询引擎。 支持多种数据源，适合PB级海量数据的复杂分析，以及跨数据源的查询。 • **Flink**：是一个在有界或无界数据流上进行有状态计算分布式处理的计算引擎。Flink on Ack基于EMR on Ack底座与社区Flink Kubernetes Operator 1.0.0版本开发，默认使用Flink官方团队推出的企业版内核，旨在为用户提供开箱即用的Flink on K8s体验。
产品版本	EMR-4.6.0-ack	默认最新的软件版本。
组件版本	SPARK (2.4.7-schemav2-ack-1.1)	所选集群类型下的组件及组件版本信息。
ACK集群	Emr-ack	选择已有的ACK集群，或者在容器服务ACK控制台新建ACK集群。 单击**配置专属节点**，可以配置EMR专属节点。配置专属节点可以对节点池或节点打上EMR专属的污点和标签，被配置的节点池或节点只能用于EMR。 **说明** 推荐您使用节点池的方式来配置专属节点，如果没有节点池，请创建节点池，详情请参见创建节点池。

参数	示例	描述
OSS Bucket	oss-spark-test	选择已有的Bucket，或者在对象存储OSS控制台新建Bucket。
集群名称	Emr-Spark	集群的名字，长度限制为1~64个字符，仅可使用中文、字母、数字、中划线（-）和下划线（_）。

4. 单击**创建**。

   当集群状态显示为**运行中**时，表示集群创建成功。

步骤二：提交作业

本文以通过CRD方式提交Spark作业为例介绍。各组件提交作业的详细信息请参见：

- 提交Spark作业
- 提交Presto作业
- 提交Flink作业

1. 通过kubectl连接Kubernetes集群，详情请参见通过kubectl工具连接集群。
2. 新建*spark-pi.yaml*文件，文件内容如下。

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: spark-pi-simple
spec:
  type: Scala
  sparkVersion: 3.2.1
  mainClass: org.apache.spark.examples.SparkPi
  mainApplicationFile: "local:///opt/spark/examples/spark-examples.jar"
  arguments:
    - "1000"
  driver:
    cores: 1
    coreLimit: 1000m
    memory: 4g
  executor:
    cores: 1
    coreLimit: 1000m
    memory: 8g
    memoryOverhead: 1g
    instances: 1
```

   本文示例中的参数描述，请参见spark-on-k8s-operator。

   > **说明**
   > - 文件名您可以自定义，本文以*spark-pi.yaml*为例介绍。
   > - 本文以Spark 3.2.1（EMR-5.6.0）版本为例，其他版本时请修改sparkVersion的配置。

3. 执行如下命令，提交作业。

```
kubectl apply -f spark-pi.yaml --namespace <集群对应的namespace>
```

   本文示例代码中的`<集群对应的namespace>`，需要替换为集群的命名空间，您可以登录E-MapReduce on ACK控制台，在**集群详情**页面查看。

   返回如下信息。

```
sparkapplication.sparkoperator.k8s.io/spark-pi-simple created
```

   > **说明** `spark-pi-simple`为本示例提交任务后的作业名。

4. （可选）您可以在**集群详情**页面的**作业**区域，查看已创建的作业信息。

（可选）步骤三：释放集群

如果您创建的集群不再使用时，可以释放集群节约成本。

1. 在集群管理页面，单击目标集群操作列的**释放**。
2. 在**释放集群**对话框中，单击**确定**。

相关文档

- 查看您账号下拥有的集群概况：查看集群信息。
- 查看您集群下的作业信息：查看作业列表。

7.4. 集群管理

7.4.1. 资源管理

7.4.1.1. 查看集群信息

本文为您介绍如何查看您账号下拥有的集群概况。

前提条件

已在E-MapReduce上创建on ACK的集群，创建详情请参见快速入门。

查看集群列表

1. 登录阿里云E-MapReduce on ACK控制台。
2. 在**集群管理**页面，您可以查看集群的列表信息。

参数	描述
集群ID/名称	集群的ID和名称。
集群类型	当前集群的类型，包括Spark和Shuffle Service。
状态	集群当前的状态： ○ 初始化中：集群正在构建。 ○ 运行中：集群处于正常运行状态。 ○ 释放中：单击集群状态列表的释放按钮可达到此状态，此状态表示集群正在努力释放。 ○ 初始化失败：集群初始化失败。保留集群基本配置信息，包括集群名称和集群ID，但是集群不可用。 说明 配置信息在集群列表中保留7天。 ○ 释放失败：集群释放失败。 ○ 已释放：集群已释放。仅保留集群配置信息，不保存集群作业信息。
集群资源使用量	显示集群的CPU和内存信息。

参数	描述
所属ACK集群	ACK集群的名称。
命名空间	ACK集群的命名空间。
创建时间	显示集群创建的时间。
操作	集群支持的操作： ○ 配置：管理服务配置，详情请参见管理配置项。 ○ 释放：释放当前集群，详情请参见释放集群。

查看集群详情

1. 登录阿里云E-MapReduce on ACK控制台。
2. 在**集群管理**页面，单击待查看集群的**名称**。
3. 在**集群详情**页面，您可以查看集群的详情。

参数	描述
集群状态	集群当前的状态： ○ 初始化中：集群正在构建。 ○ 运行中：集群处于正常运行状态。 ○ 释放中：单击集群状态列表的释放按钮可达到此状态，此状态表示集群正在努力释放。 ○ 初始化失败：集群初始化失败。保留集群基本配置信息，包括集群名称和集群ID，但是集群不可用。 说明 配置信息在集群列表中保留7天。 ○ 释放失败：集群释放失败。 ○ 已释放：集群已释放。仅保留集群配置信息，不保存集群作业信息。
集群名称	在E-MapReduce上创建的集群名称。
集群ID	在E-MapReduce上创建的集群ID。
所属ACK集群	ACK集群的名称。
命名空间	ACK集群的命名空间。
OSS Bucket	OSS控制台创建的Bucket。
集群类型	当前集群的类型，包括Spark和Shuffle Service。
产品版本	产品的版本信息。
VPC	ACK集群的VPC信息。
创建时间	显示集群创建的时间。
集群资源使用量	显示集群的CPU和内存信息。
最大资源使用上限	显示EMR集群对应的ACK命名空间的ResourceQuota属性值。
关联ShuffleService	显示ShuffleService是否关联状态。默认是未关联。 如果需要关联，可以单击**前往关联**。
数据湖构建（DLF）	显示DLF是否启用的状态。默认是未启用。 如果需要启用，可以单击**点击启用**。

7.4.1.2. 释放集群

当集群不再使用时，您可以释放集群以删除对应的NamSpace以及该Namespace下的所有软件服务，但不会释放实际的物理资源。本文为您介绍如何释放集群。

前提条件

请确保待释放集群的状态是创建中、运行中或空闲中。

操作步骤

1. 登录阿里云E-MapReduce on ACK控制台。
2. 在**集群管理**页面，单击待释放集群**操作**列的**释放**。
3. 在弹出的对话框中，单击**确定**。

7.4.2. 服务管理

7.4.2.1. 重启服务

修改配置项后，需要重启对应的服务使配置生效。本文为您介绍如何重启服务。

前提条件

已在E-MapReduce上创建on ACK的集群，创建详情请参见快速入门。

操作步骤

1. 进入服务详情页面。
   i. 登录阿里云E-MapReduce on ACK控制台。
   ii. 在**集群管理**页面，单击目标集群操作列的**配置**。
   iii. 单击上方的**服务详情**。
2. 在**服务详情**页面，单击待重启组件操作列的**重启**。
3. 在弹出的对话框中，输入**执行原因**，单击**确定**。
4. 在**确认**对话框中，单击**确定**。

7.4.2.2. 访问Web UI

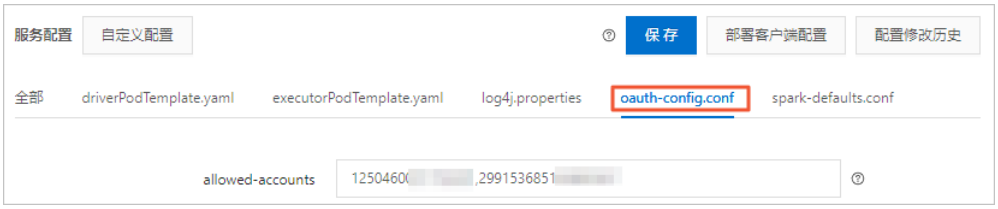
访问Web UI页面需要进行用户身份认证，添加EMR用户后可以访问相应的UI页面。集群创建后，默认仅支持集群创建者对应的阿里云账号访问开源组件的Web UI，如果其他阿里云账号（主账号）或RAM用户（子账号）需要访问当前集群的Web UI，请按照本文操作。

前提条件

- 已在E-MapReduce上创建on ACK的集群，创建详情请参见快速入门。
- 已获取其他阿里云账号或RAM用户的账号ID。

操作步骤

1. 进入配置页面。
   i. 登录阿里云E-MapReduce on ACK控制台。
   ii. 在**集群管理**页面，单击目标集群操作列的**配置**。
   iii. 单击上方的**配置**页签。
2. 添加用户。
   i. 在**服务配置**区域，单击**oauth-config.conf**页签。
   ii. 设置**allowed-accounts**参数的参数值为阿里云账号的账号ID。
   添加多个账号时，使用英文逗号（,）隔开。



3. 保存配置。
   i. 单击上方的**保存**。
   ii. 在确认修改对话框中，输入**执行原因**，开启**自动更新配置**。
   iii. 单击**确定**。
4. 部署配置。

i. 单击上方的**部署客户端配置**。
ii. 在弹出的对话框中，输入**执行原因**，单击**确定**。
iii. 在**确认**对话框中，单击**确定**。

5. 访问开源组件或作业的Web UI。
   i. 单击上方的**集群详情**页签。
   ii. 在**集群详情**页面的**服务信息**或**作业**区域，单击服务所在行的链接，即可正常的访问Web UI页面。

7.4.2.3. 管理配置项

本文为您介绍如何修改和添加配置项。

前提条件

已在E-MapReduce上创建on ACK的集群，创建详情请参见快速入门。

修改配置项

1. 进入配置页面。
   i. 登录阿里云E-MapReduce on ACK控制台。
   ii. 在**集群管理**页面，单击目标集群操作列的**配置**。
   iii. 单击上方的**配置**页签。
2. 修改配置。
   i. 在配置搜索中，输入待修改的配置项，单击 图标。
   ii. 找到您要修改的参数，修改对应的值。
3. 保存配置。
   i. 单击**保存**。
   ii. 在**修改信息**对话框中，输入**执行原因**，开启**自动配置更新**，单击**保存**。
4. 部署配置。
   i. 单击**部署客户端配置**。
   ii. 在弹出的对话框中，输入**执行原因**，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

> **说明** Spark集群中的配置是作为Spark提交作业的默认参数使用，会被您提交的YAML作业中指定的参数覆盖，默认配置生效后，之前提交的作业配置是不能更新的，只有后续提交的任务才能使用新的默认配置；Shuffle Service集群的配置生效后，需要重启组件才能使新配置生效，重启详情请参见重启服务。

添加配置项

1. 进入配置页面。
   i. 登录阿里云E-MapReduce on ACK控制台。
   ii. 在**集群管理**页面，单击目标集群操作列的**配置**。
   iii. 单击上方的**配置**页签。
2. 添加配置项。
   i. 在目标服务的配置页面的**服务配置**区域，单击待操作的页签。
   ii. 单击上方的**新增配置项**。
   iii. 根据您的实际情况，添加配置项信息，单击**确定**。
   一次可以添加多个配置项。

配置项	描述
Key	参数名。
Value	参数值。
描述	参数描述。
操作	支持删除配置项。

   iv. 在**修改信息**对话框中，输入执行原因，打开**自动配置更新**开关，单击**保存**。

3. 部署配置。
   i. 单击**部署客户端配置**。
   ii. 在弹出的对话框中，输入**执行原因**，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

7.4.3. 作业管理

7.4.3.1. 提交作业

7.4.3.1.1. 提交Spark作业

EMR支持CRD、spark-submit和控制台终端三种方式提交作业。本文为您介绍如何提交Spark作业。

前提条件

已在E-MapReduce on ACK控制台创建Spark集群，详情请参见快速入门。

方式一：使用CRD方式提交作业（推荐）

1. 通过kubectl连接Kubernetes集群，详情请参见通过kubectl工具连接集群。
2. 新建*spark-pi.yaml*文件，文件内容如下。

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: spark-pi-simple
spec:
  type: Scala
  sparkVersion: 3.2.1
  mainClass: org.apache.spark.examples.SparkPi
  mainApplicationFile: "local:///opt/spark/examples/spark-examples.jar"
  arguments:
    - "1000"
  driver:
    cores: 1
    coreLimit: 1000m
    memory: 4g
  executor:
    cores: 1
    coreLimit: 1000m
    memory: 8g
    memoryOverhead: 1g
    instances: 1
```

本文示例中的参数描述，请参见spark-on-k8s-operator。

> **说明**
> - 文件名您可以自定义，本文以*spark-pi.yaml*为例介绍。
> - 本文以Spark 3.2.1（EMR-5.6.0）版本为例，其他版本时请修改sparkVersion的配置。

3. 执行如下命令，提交作业。

```
kubectl apply -f spark-pi.yaml --namespace <集群对应的namespace>
```

本文示例代码中的`<集群对应的namespace>`，需要替换为集群的命名空间，您可以登录E-MapReduce on ACK控制台，在**集群详情**页面查看。

返回如下信息。

```
sparkapplication.sparkoperator.k8s.io/spark-pi-simple created
```

> **说明** `spark-pi-simple`为本示例提交任务后的作业名。

4. （可选）您可以在**集群详情**页面的**作业**区域，查看已创建的作业信息。

方式二：使用spark-submit方式提交作业

1. 通过kubectl连接Kubernetes集群，详情请参见通过kubectl工具连接集群。
2. 执行以下命令，安装阿里云EMR提供的emr-spark-ack工具并授权。

```
wget https://emr-on-ack.oss-cn-beijing.aliyuncs.com/util/emr-spark-ack
chmod 755 emr-spark-ack
```

3. 使用emr-spark-ack工具提交作业。

   提交作业的语法如下。

```
./emr-spark-ack -n <集群对应的namespace> <spark命令>
```

> **说明** 语法中的 `<spark命令>` 支持spark-submit、spark-sql、spark-shell和pyspark四种，语法和Spark本身完全一致。

   - Cluster模式示例

     通过spark-submit提交spark-pi作业。

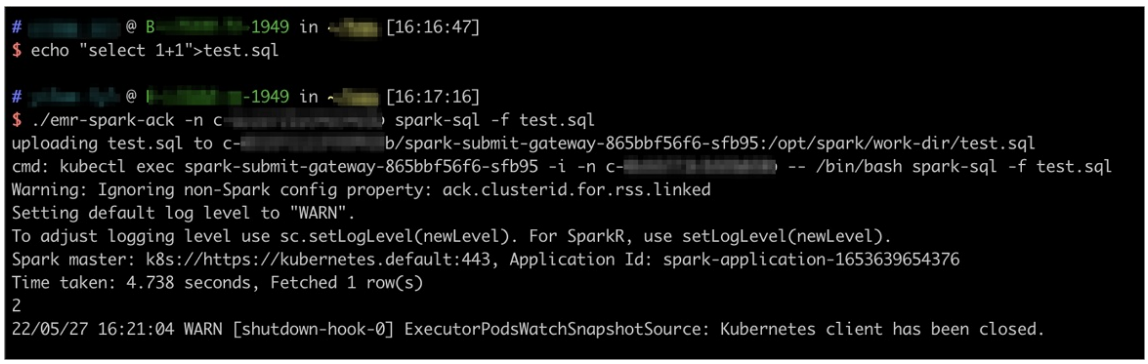
```
./emr-spark-ack -n <集群对应的namespace> spark-submit \
    --name spark-pi-submit \
    --deploy-mode cluster \
    --class org.apache.spark.examples.SparkPi \
    local:///opt/spark/examples/spark-examples.jar \
    1000
```

   - Client模式示例
     - spark-sql命令方式

```
# 本地准备sql文件
echo "select 1+1">test.sql
# 提交作业
./emr-spark-ack -n <集群对应的namespace> spark-sql -f test.sql
```

       在Spark 3及以上集群版本（EMR-5.X版本）中，emr-spark-ack工具支持本地依赖自动上传，提交命令里面的本地文件依赖，包括 `--jars` ， `--files` 和 `-f` 等参数中传入的本地文件，会自动上传到EMR on ACK集群内，用于K8s环境的作业提交。

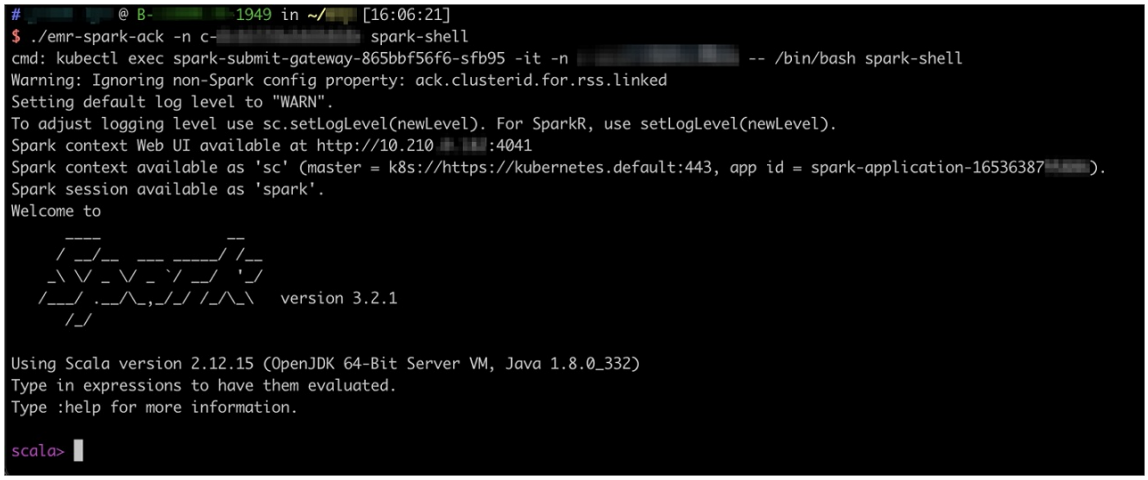
       代码示例及返回信息见下图。



- spark-shell命令方式

```
./emr-spark-ack -n <集群对应的namespace> spark-shell
```

代码示例及返回信息见下图。



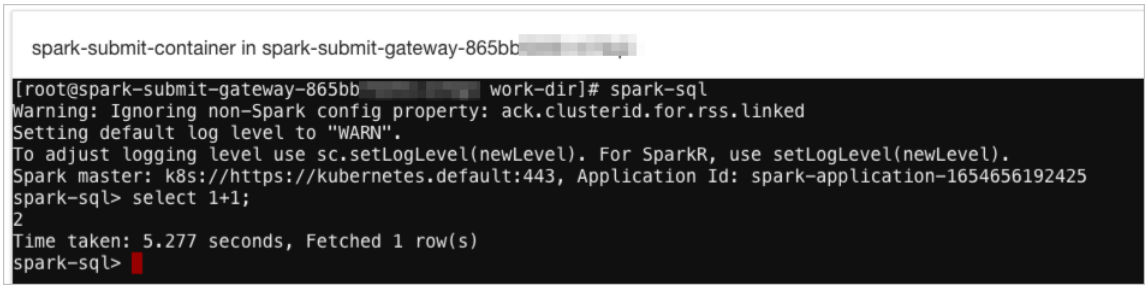
4. （可选）您可以在**集群详情**页面，查看已创建的作业信息。

方式三：使用控制台终端方式提交作业

1. 进入访问链接与端口页面。
   i. 登录阿里云E-MapReduce on ACK控制台。
   ii. 在**集群管理**页面，单击目标集群的集群名。
   iii. 单击上方的**访问链接与端口**页签。
2. 在**访问链接与端口**页面，单击**SparkSubmit Gateway UI**对应的链接。
   即可进入Shell终端。
3. 在Shell终端中，可以通过以下两种方式运行Spark命令。
   - spark-sql命令方式

```
spark-sql
```

进入spark-sql后，您可以直接运行Spark命令进行交互式查询。



   - spark-submit命令方式

```
spark-submit \
    --name spark-pi-submit \
    --deploy-mode cluster \
    --class org.apache.spark.examples.SparkPi \
    local:///opt/spark/examples/spark-examples.jar \
    1000
```

4. （可选）您可以在**集群详情**页面的**作业**区域，查看已创建的作业信息。

相关文档

- 通过kubectl管理Spark作业详情，请参见使用kubectl管理作业。
- 通过阿里云日志服务收集Spark作业的日志详情，请参见使用日志服务收集Spark作业日志。
- 在EMR on ACK中设置Spark集群的元数据详情，请参见为Spark集群设置元数据。
- 使用ECI弹性调度Spark作业详情，请参见使用ECI弹性调度Spark作业。

7.4.3.1.2. 提交Presto作业

本文为您介绍如何提交Presto作业。

前提条件

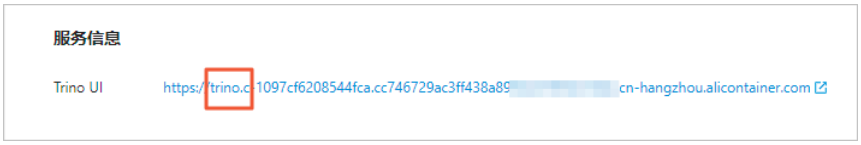
- 已在E-MapReduce on ACK控制台创建Presto集群，详情请参见快速入门。
- 已添加AliyunOSSFullAccess和AliyunDLFFullAccess权限，详情请参见角色授权。

操作步骤

1. 下载trino至本地目录。
2. 在本地执行以下命令，修改文件权限。

   ```
   sudo chmod +x trino
   ```

3. 启动Trino。
   i. 登录阿里云E-MapReduce on ACK控制台。
   ii. 在**集群管理**页面，单击目标集群的集群名称。
   iii. 在**集群详情**页签的**服务信息**区域，获取Trino UI的链接。

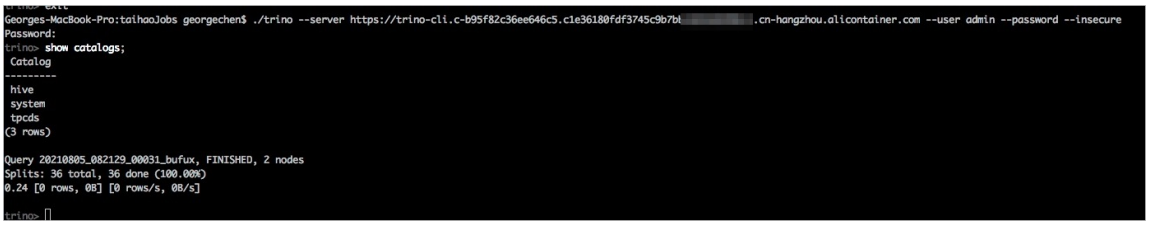
   iv. 在本地执行以下命令，启动Trino。

   ```
   ./trino --server https://trino-cli.c-b95f82c36ee646c5.c1e36180fdf3745c9b7bbc8d3e682****.cn-hangzhou.alicontainer.com --user admin --password --insecure
   ```

   > 注意 `https://trino-cli.c-b95f82c36ee646c5.c1e36180fdf3745c9b7bbc8d3e682****.cn-hangzhou.alicontainer.com` 为您在前一步骤中获取到的Trino UI的链接，需要修改链接中的 `trino.c` 为 `trino-cli.c` 。

   根据提示输入密码，登录后您可以执行以下命令查询catalogs。

   ```
   show catalogs;
   ```

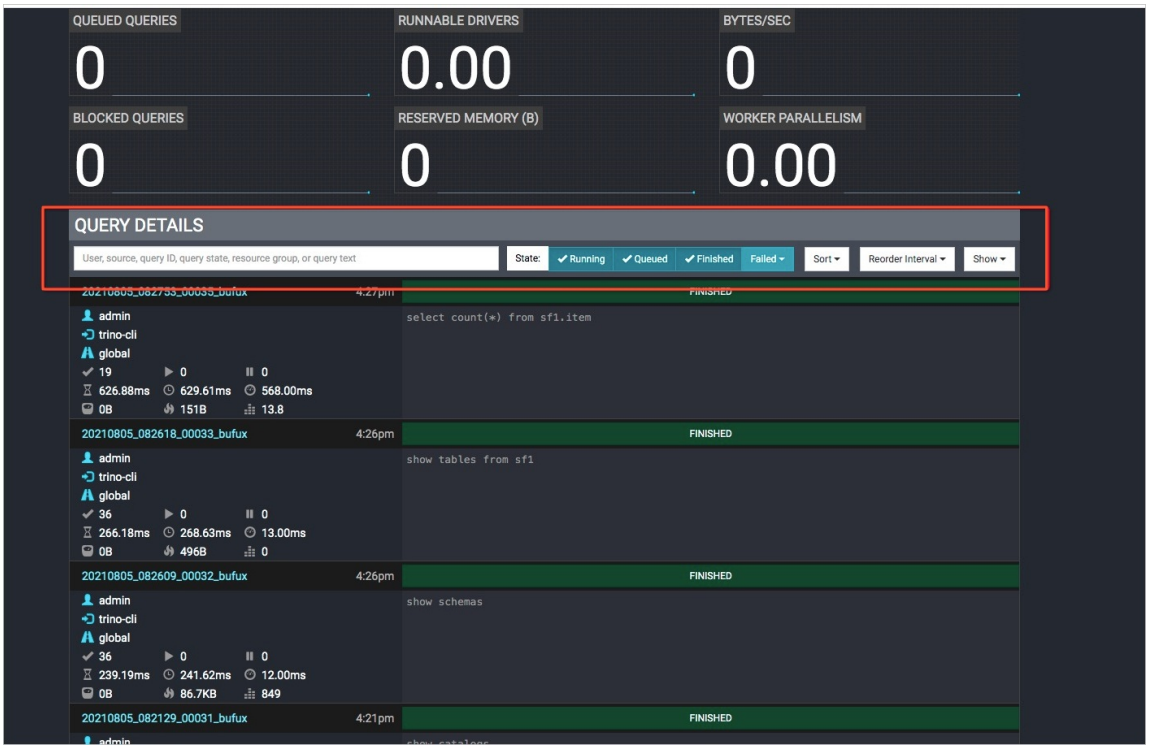
   

   更多SQL语句用法，请参见SQL statement syntax。

4. 查看作业详情。
   i. 在**集群详情**页签的**服务信息**区域，单击Trino UI的链接。

ii. 输入默认用户admin及密码aliyunEMR!，单击**Log In**。

登录后，即可查看作业执行的详细信息。



7.4.3.1.3. 提交Flink作业

本文为您介绍如何提交Flink作业。

前提条件

已在E-MapReduce on ACK控制台创建Flink集群，详情请参见快速入门。

使用限制

Flink集群类型目前是白名单模式，如果您想使用Flink集群请提交工单申请。

方式一：通过ACK控制台提交作业

1. 登录阿里云E-MapReduce on ACK控制台。
2. 在**集群管理**页面，单击目标集群所在行**所属ACK集群**列的链接。
3. 在**容器组**页面，单击右上角的**使用YAML创建资源**。
4. 在**创建**页面，从**示例模板**列表中，选择**自定义**，模板内容请复制以下内容，然后单击**创建**。

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: basic-emr-example
spec:
  flinkVersion: v1_13
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
    state.savepoints.dir: file:///flink-data/flink-savepoints
    state.checkpoints.dir: file:///flink-data/flink-checkpoints
  serviceAccount: flink
  podTemplate:
    spec:
      serviceAccount: flink
      containers:
        - name: flink-main-container
          volumeMounts:
            - mountPath: /flink-data
              name: flink-volume
      volumes:
        - name: flink-volume
          emptyDir: {}
  jobManager:
    replicas: 1
    resource:
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
  job:
    jarURI: local:///opt/flink/examples/streaming/StateMachineExample.jar
    parallelism: 2
    upgradeMode: stateless
```

说明 本文以Flink 1.13版本为例，其他版本时请修改flinkVersion的配置，具体版本以控制台为准。

方式二：通过kubectl工具提交作业

1. 通过kubectl连接Kubernetes集群，详情请参见通过kubectl工具连接集群。
   您也可以通过API等方式连接Kubernetes集群，详情请参见使用Kubernetes API。
2. 新建*basic-emr-example.yaml*文件，文件内容如下。

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: basic-emr-example
spec:
  flinkVersion: v1_13
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
    state.savepoints.dir: file:///flink-data/flink-savepoints
    state.checkpoints.dir: file:///flink-data/flink-checkpoints
  serviceAccount: flink
  podTemplate:
    spec:
      serviceAccount: flink
      containers:
        - name: flink-main-container
          volumeMounts:
            - mountPath: /flink-data
              name: flink-volume
      volumes:
        - name: flink-volume
          emptyDir: {}
  jobManager:
    replicas: 1
    resource:
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
  job:
    jarURI: local:///opt/flink/examples/streaming/StateMachineExample.jar
    parallelism: 2
    upgradeMode: stateless
```

说明

- 文件名您可以自定义，本文以*basic-emr-example.yaml*为例介绍。
- 本文以Flink 1.13版本为例，其他版本时请修改flinkVersion的配置。

3. 执行以下命令，提交作业。

```
kubectl apply -f basic-emr-example.yaml -namespace <集群对应的namespace>
```

说明 本文示例代码中的 `<集群对应的namespace>` ，需要替换为集群的命名空间，您可以登录E-MapReduce on ACK控制台，在集群管理页面查看。

相关文档

- 查看Flink作业日志和访问Flink Web UI

7.4.3.2. 查看作业列表

本文为您介绍如何查看您集群下的作业信息。

前提条件

已在E-MapReduce控制台创建on ACK的集群，详情请参见快速入门。

操作步骤

1. 进入集群详情页面。
   i. 登录阿里云E-MapReduce on ACK控制台。
   ii. 在**集群管理**页面，单击待查看集群的**名称**。
2. 在**集群详情**页面的**作业**区域，您可以查看作业列表信息。

参数	描述
作业ID	作业的ID。
作业名	提交作业的名称。
类型	组件的类型。
状态	作业当前的状态。状态包括已提交、执行中、结束、失败、完成中、提交失败、任务无效、等待重试和失败中。
创建时间	作业开始创建的时间。
结束时间	作业创建结束的时间。
执行时长	作业运行的时间。
跟踪URL	Spark或Historyserver。 运行中时是Spark的UI链接，运行结束是Historyserver的UI链接。

7.4.3.3. 查看Flink作业日志和访问Flink Web UI

本文为您介绍，如何查看运行中的Flink作业的日志和访问对应作业的Web UI。

前提条件

已在E-MapReduce on ACK控制台创建Flink集群，详情请参见快速入门。

查看Flink作业日志

1. 登录阿里云E-MapReduce on ACK控制台。
2. 在**集群管理**页面，单击目标集群所在行**所属ACK集群**列的链接。
3. 在**容器组**页面，单击目标Pod操作列的**日志**。
   则可查看对应JobManager或者TaskManager的日志。

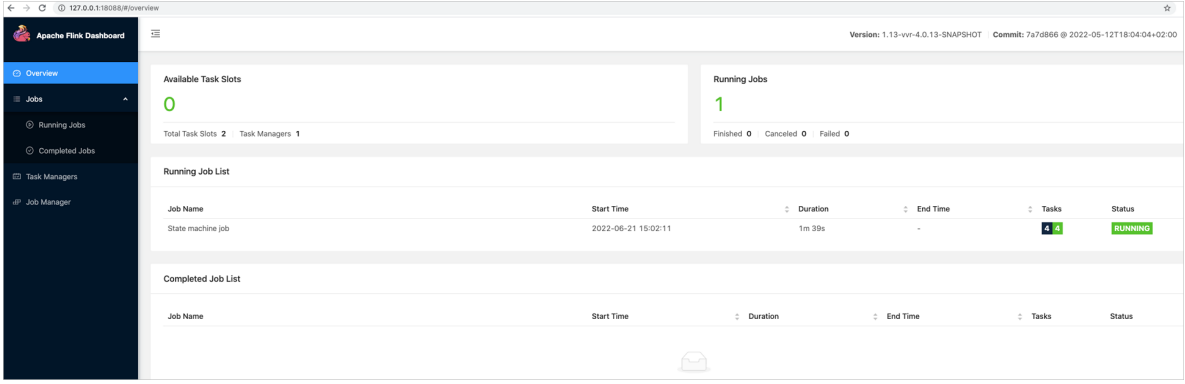
访问Flink Web UI

1. 通过kubectl连接Kubernetes集群，详情请参见通过kubectl工具连接集群。
   您也可以通过API等方式连接Kubernetes集群，详情请参见使用Kubernetes API。
2. 执行以下命令，转发Flink作业JobManager的服务到本地进行访问。

```
kubectl port-forward -n <yourClusterId> svc/basic-emr-example-rest 18088:8081
```

> **说明** 示例代码中需替换以下参数：
> - `<yourClusterId>`：集群ID，您可以登录E-MapReduce on ACK控制台，在集群管理页面获取。
> - `svc/basic-emr-example-rest`：您运行的Flink作业的服务名称，您可以通过命令 `kubectl get svc -n <yourClusterId>` 查看所有的K8s服务。请选择<flink_deployment_name>-rest服务进行转发。

3. 在浏览器中访问*http://127.0.0.1:18088/#/overview*，查看作业运行情况。



7.5. 组件操作指南

7.5.1. Presto

7.5.1.1. 配置连接器

本文为您介绍EMR on ACK的Presto提供的内置连接器，以及如何修改连接器。暂不支持增加自定义连接器。

前提条件

已在EMR on ACK控制台上创建Presto集群，详情请参见创建集群。

EMR Presto内置连接器

EMR Presto默认提供了开箱即用的内置连接器，详情信息如下表。

连接器	功能	对应文档
hive	使用Hive连接器可以查询存储在Hive数据仓库中的数据。	Hive连接器
kudu	使用Kudu连接器可以查询、插入和删除存储在Kudu里的数据。	Kudu连接器
iceberg	使用Iceberg连接器可以查询Iceberg格式的数据文件。	Iceberg连接器
mysql	使用MySQL连接器可以在外部MySQL实例中查询和创建表。	MySQL连接器
hudi	使用Hudi连接器可以查询COW和MOR表。	Hudi连接器
phoenix	使用Phoenix连接器可以查询存储在HBase中的数据。	无
tpcds	使用TPCDS连接器可用于测试Presto的功能和查询语法，而无需配置对外部数据源的访问。	无

修改内置连接器

您可以在EMR on ACK控制台的Presto集群的**配置**页面，修改各个连接器对应的配置文件。连接器与配置文件对应关系如下表。

连接器	配置文件
hive	**catelog-hive.properties**
kudu	**catelog-kudu.properties**
iceberg	**catelog-iceberg.properties**
mysql	**catelog-mysql.properties**
hudi	**catelog-hudi.properties**
phoenix	**catelog-phoenix.properties**
tpcds	**catelog-tpcds.properties**

示例：修改Hive连接器

1. 在EMR on ACK控制台的Presto集群的**配置**页面，在**服务配置**区域，单击**catelog-hive.properties**页签。
2. 根据实际情况修改各配置项。
3. 保存配置时，在**确认修改配置**对话框中开启**自动更新配置**。
4. 保存完配置后，在服务详情页签下，重启Presto服务。

> **说明** 服务重启完成后，即可正常使用连接器。

7.5.1.2. 使用Hive连接器读取DLF数据表

修改Hive连接器的配置后，可以正常读取DLF（Data Lake Formation）的数据表。本文为您介绍如何使用Hive连接器读取DLF数据表。

前提条件

已在EMR on ACK控制台上创建Presto集群，详情请参见创建集群。

操作步骤

1. 进入catelog-hive.properties页签。
   i. 登录阿里云E-MapReduce on ACK控制台。
   ii. 在**集群管理**页面，单击目标集群所在行的**配置**。
   iii. 在**服务配置**区域，选择... > **catelog-hive.properties**页签。
2. 新增配置项。
   i. 在**服务配置**区域，单击**自定义配置**。
   ii. 在**自定义配置**对话框中，新增以下配置信息。

参数	描述
hive.metastore	固定值dlf。
dlf.catalog.proxyMode	固定值DLF_ONLY。
dlf.catalog.akMode	固定值DLF_AUTO。
dlf.catalog.uid	阿里云账号的账号ID。 登录账号信息，请通过用户信息页面获取。 
dlf.catalog.endpoint	DLF服务的Endpoint。详情请参见已开通的地域和访问域名。 推荐您设置为DLF的VPC Endpoint。例如，如果您选择的地域为cn-hangzhou地域，则参数值需要配置为dlf-vpc.cn-hangzhou.aliyuncs.com。 **说明** 您也可以使用DLF的公网Endpoint，如果您选择的地域为cn-hangzhou地域，则参数值需要配置为dlf.cn-hangzhou.aliyuncs.com。
dlf.catalog.region	DLF服务的地域名。例如，cn-hangzhou。

   iii. 单击**确定**。
   iv. 在**确认修改配置**对话框中，输入**执行原因**，打开**自动更新配置**开关，单击**确定**。
3. 重启Presto服务。
   i. 单击上方的**服务详情**页签。

ii. 在**组件列表**区域，单击TrinoCoordinator和TrinoWorker操作列的**重启**。

iii. 在弹出的对话框中，输入**执行原因**，单击**确定**。

iv. 在**确认**对话框中，单击**确定**。
服务重启完成后，即可正常读取DLF数据表。

7.5.1.3. 使用日志服务收集Presto作业日志

本文为您介绍如何通过阿里云日志服务SLS收集Presto作业的日志。

背景信息

由于EMR on ACK使用虚拟容器运行作业，作业运行完成后，相关的容器会被销毁以释放集群资源。对于弹性伸缩或分时调度，相关的容器资源也会被销毁。因此，当EMR on ACK上的作业或者服务运行有异常时，如果容器运行时的日志没有被及时保留和持久化，容器资源释放后，对异常根因定位相关的工作会造成很大的困难，因此EMR on ACK推荐启用日志服务SLS对容器日志进行保留。

前提条件

- 已在E-MapReduce on ACK控制台创建Presto集群，详情请参见快速入门。
- 已开通阿里云日志服务SLS，详情请参见快速入门。

操作步骤

EMR on ACK推荐启用日志服务SLS对容器日志进行保留，详情请参见通过日志服务采集Kubernetes容器日志。

> 说明
>
> 涉及的日志路径如下：
>
> - 标准日志输出路径：*stdout*。
> - core dump日志输出路径：*/data/trino/hs_err_pid1.log*。

7.5.1.4. 配置hosts

Presto on ACK提供了自定义hosts功能，当Presto on ACK集群读取EMR on ECS集群的Hive数据时，该功能可以提供正确的域名解析配置。本文为您介绍如何配置hosts。

背景信息

如果没有正确配置hosts，则可能遇到以下报错提示。

```
java.net.UnknownHostException: emr-header-1.cluster-xxxx
```

前提条件

已在EMR on ACK控制台上创建Presto集群，详情请参见创建集群。

操作步骤

1. 进入hosts.properties页签。
   i. 登录阿里云E-MapReduce on ACK控制台。
   ii. 在**集群管理**页面，单击目标集群所在行的**配置**。
   iii. 在**服务配置**区域，选择... > **hosts.properties**页签。
2. 新增配置项。
   i. 在**服务配置**区域，单击**新增配置项**。

ii. 在**新增配置项**对话框中，**Key**设置为内网IP地址，**Value**设置为hostname。

通常情况下，只需要配置header节点所在机器的内网IP地址和hostname，即可访问EMR on ECS集群的Hive数据。

> **说明**
> - 内网IP地址：旧版控制台，您可以在集群管理中的主机列表页面查看。新版控制台，您可以在集群的节点管理页面查看。
> - hostname：您可以在emr-header-1节点上，执行 `hostname` 命令获取。

iii. 单击**确定**。

iv. 在**确认修改**对话框中，输入**执行原因**，打开**自动配置更新**开关，单击**保存**。

3. 重启Presto服务。

i. 单击上方的**服务详情**页签。

ii. 在**组件列表**区域，单击**TrinoCoordinator**和**TrinoWorker**操作列的**重启**。

iii. 在弹出的对话框中，输入**执行原因**，单击**确定**。

iv. 在**确认**对话框中，单击**确定**。

服务重启完成后，即可正常访问Hive数据。

7.5.2. Spark

7.5.2.1. 使用kubectl管理作业

您可以在EMR on ACK控制台管理您的作业，也可以通过Kubernetes工具或API直接管理您的作业。本文为您介绍如何通过kubectl管理Spark作业。

前提条件

已在E-MapReduce on ACK控制台创建Spark集群，详情请参见快速入门。

操作步骤

1. 通过kubectl连接Kubernetes集群，详情请参见通过kubectl工具连接集群。

   您也可以通过API等方式连接Kubernetes集群，详情请参见使用Kubernetes API。

2. 执行以下命令，管理作业。

   - 您可以执行以下命令，查看作业状态。

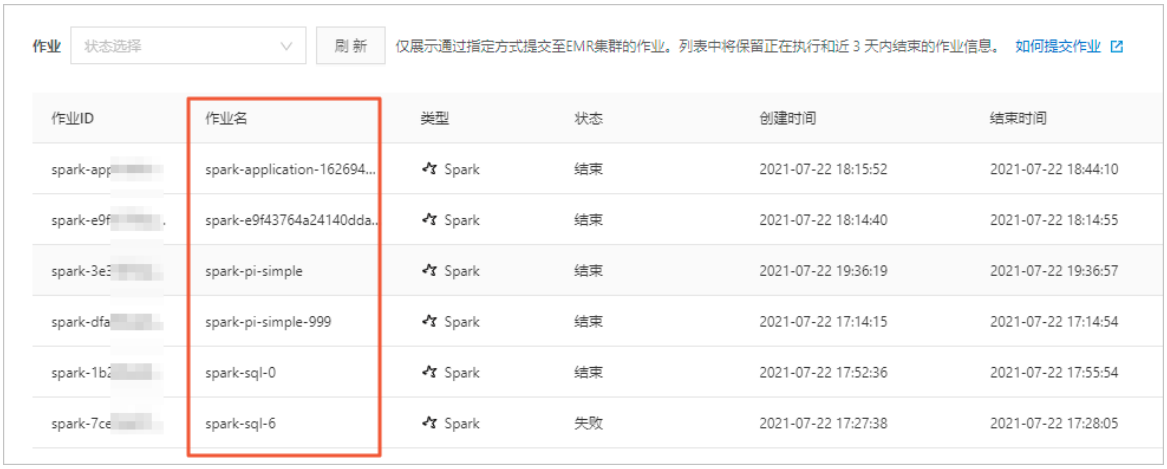
     ```
     kubectl describe SparkApplication <作业名> --namespace <集群对应的namespace>
     ```

     返回信息如下所示。

```
Name:         spark-pi-simple
Namespace:    c-48e779e0d9ad****
Labels:       <none>
Annotations:  <none>
API Version:  sparkoperator.k8s.io/v1beta2
Kind:         SparkApplication
Metadata:
  Creation Timestamp:  2021-07-22T06:25:33Z
  Generation:          1
  Resource Version:  7503740
  UID:               930874ad-bb17-47f1-a556-55118c1d****
Spec:
  Arguments:
    1000
  Driver:
    Core Limit:  1000m
    Cores:       1
    Memory:      4g
  Executor:
    Core Limit:           1000m
    Cores:                1
    Instances:            1
    Memory:               8g
    Memory Overhead:      1g
  Image:                  registry-vpc.cn-hangzhou.aliyuncs.com/emr/spark:emr-2.4.5-1.0.0
  Main Application File:  local:///opt/spark/examples/target/scala-2.11/jars/spark-examples_2.11-2.4.5.jar
  Main Class:             org.apache.spark.examples.SparkPi
  Spark Version:          2.4.5
  Type:                   Scala
Status:
  Application State:
    State:  RUNNING
  Driver Info:
    Pod Name:                spark-pi-simple-driver
    Web UI Address:          172.16.230.240:4040
    Web UI Ingress Address:  spark-pi-simple.c-48e779e0d9ad4bfd.c7f6b768c34764c27ab740bdb1fc2a3ff.cn-hangzhou.alicont
ainer.com
    Web UI Ingress Name:     spark-pi-simple-ui-ingress
    Web UI Port:             4040
    Web UI Service Name:     spark-pi-simple-ui-svc
  Execution Attempts:        1
  Executor State:
    spark-pi-1626935142670-exec-1:  RUNNING
  Last Submission Attempt Time:     2021-07-22T06:25:33Z
  Spark Application Id:             spark-15b44f956ecc40b1ae59a27ca18d****
  Submission Attempts:              1
  Submission ID:                    d71f30e2-9bf8-4da1-8412-b585fd45****
  Termination Time:                 <nil>
Events:
  Type    Reason                     Age   From            Message
  ----    ------                     ----  ----            -------
  Normal  SparkApplicationAdded      17s   spark-operator  SparkApplication spark-pi-simple was added, enqueuing it f
or submission
  Normal  SparkApplicationSubmitted  14s   spark-operator  SparkApplication spark-pi-simple was submitted successfull
y
  Normal  SparkDriverRunning         13s   spark-operator  Driver spark-pi-simple-driver is running
  Normal  SparkExecutorPending       7s    spark-operator  Executor spark-pi-1626935142670-exec-1 is pending
  Normal  SparkExecutorRunning       6s    spark-operator  Executor spark-pi-1626935142670-exec-1 is running
```

本文示例代码中的 `<集群对应的namespace>` ，需要替换为集群的命名空间，您可以登录E-MapReduce on ACK控制台，在**集群详情**页面查看。

本文示例代码中的 <作业名> ，您可以登录E-MapReduce on ACK控制台，在**集群详情**页面的**作业**区域，查看已创建的作业名。



- 您可以执行以下命令，停止并删除作业。

  ```
  kubectl delete SparkApplication <作业名> -n <集群对应的namespace>
  ```

  返回信息如下所示。

  ```
  sparkapplication.sparkoperator.k8s.io "spark-pi-simple" deleted
  ```

- 您可以执行以下命令，查看作业日志。

  ```
  kubectl logs <作业名-driver> -n <集群对应的namespace>
  ```

  > **说明** 例如，作业名为spark-pi-simple，集群对应的namespace为c-d2232227b95145d3，则对应的命令为 `kubectl logs spark-pi-simple-driver -n c-d2232227b95145d3` 。

  返回如下类似信息。

  ```
  ......
  Pi is roughly 3.141488791414888
  21/07/22 14:37:57 INFO SparkContext: Successfully stopped SparkContext
  21/07/22 14:37:57 INFO ShutdownHookManager: Shutdown hook called
  21/07/22 14:37:57 INFO ShutdownHookManager: Deleting directory /var/data/spark-b6a43b55-a354-44d7-ae5e-45b8b1493edb/spark-56aae0d1-37b9-4a7d-9c99-4e4ca12deb4b
  21/07/22 14:37:57 INFO ShutdownHookManager: Deleting directory /tmp/spark-e2500491-6ed7-48d7-b94e-a9ebeb899320
  ```

7.5.2.2. 使用日志服务收集Spark作业日志

本文为您介绍如何通过阿里云日志服务收集Spark作业的日志。

前提条件

- 已在E-MapReduce on ACK控制台创建Spark集群，详情请参见快速入门。
- 已开通阿里云日志服务SLS，详情请参见快速入门。

操作步骤

1. 启用日志服务组件Logtail，详情请参见步骤一：启用日志服务组件Logtail。

   > **说明** 如果已启用日志服务组件Logtail，则直接执行步骤2。

2. 进入日志服务对应Project的控制台。
   i. 登录容器服务管理控制台。
   ii. 在控制台左侧导航栏中，单击**集群**。
   iii. 在**集群列表**页面，单击目标集群名称或者目标集群右侧**操作**列下的**详情**。
   iv. 在**集群信息**页面，单击上方的**集群资源**页签。
   v. 单击**日志服务Project**所在行的链接。
   进入日志服务对应Project的控制台。

3. 在**日志库**页签中，新建两个Logstore。

   本文示例新建的两个Logstore分别命名为spark-driver-log和spark-executor-log。创建Logstore详情，请参见步骤二：创建Project和Logstore。

4. 在spark-driver-log的Logstore中，执行以下操作。

   i. 新增Logtail配置，接入数据选择Kubernetes标准输出，在配置流程中选择已有的Kubernetes机器组。

   ii. 选择**数据接入 > logtail配置**下已有的Kubernetes机器组。

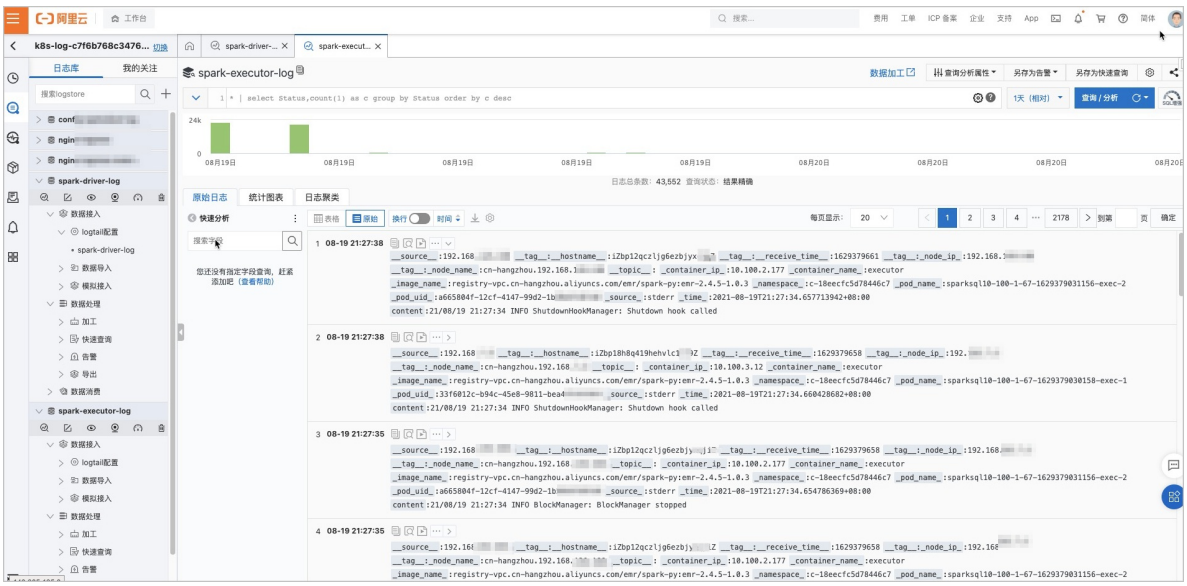
   iii. 在**插件配置**文本框内，输入以下内容。

```
{
    "inputs": [
        {
            "detail": {
                "IncludeEnv": {
                    "SPARKLOGENV": "spark-driver"
                },
                "Stderr": true,
                "Stdout": true,
                "BeginLineCheckLength": 10,
                "BeginLineRegex": "\\d+/\\d+/\\d+.*"
            },
            "type": "service_docker_stdout"
        }
    ]
}
```

5. 在spark-executor-log的Logstore中，参照步骤4在**插件配置**文本框中，输入以下内容。

```
{
    "inputs": [
        {
            "detail": {
                "IncludeEnv": {
                    "SPARKLOGENV": "spark-executor"
                },
                "Stderr": true,
                "Stdout": true,
                "BeginLineCheckLength": 10,
                "BeginLineRegex": "\\d+/\\d+/\\d+.*"
            },
            "type": "service_docker_stdout"
        }
    ]
}
```

6. 开启日志库Logstore索引，详情请参见配置索引。
   完成以上步骤，即可在SLS上查询到作业的日志了。



7.5.2.3. 为Spark集群关联RSS

RSS（EMR Remote Shuffle Service）是E-MapReduce（简称EMR）为了提升Shuffle稳定性和性能推出的扩展组件，优化了Spark原生的Shuffle。本文为您介绍EMR on ACK上的Spark集群如何关联RSS。

背景信息

目前在ACK的场景下，Spark Shuffle面临的问题：

- Spark Shuffle对本地存储有依赖，许多计算存储分离的机型、使用ECI的场景下没有自带本地盘，需要额外购买和挂载云盘，性价比和使用效率低。
- Spark2在ACK环境下不支持Dynamic Allocation，Spark3基于ShuffleTracking实现了Dynamic Allocation，但Executor回收效率低下。

目前Spark Shuffle方案缺点如下：

- Shuffle Write在大数据量场景下会溢出，导致写放大。
- Shuffle Read过程中存在大量的网络小包导致的Connection reset问题。
- Shuffle Read过程中存在大量小数据量的IO请求和随机读，对磁盘和CPU造成高负载。
- 对于M*N次的连接数，在M和N数千的规模下，作业基本无法完成。

EMR推出的RSS服务，可以优化上述Spark Shuffle方案的问题，完美支持ACK环境下的Dynamic Allocation。RSS详情请参见RSS。

前提条件

- 已在E-MapReduce on ACK控制台创建Spark集群，详情请参见步骤一：创建集群。
- 已在E-MapReduce on ACK控制台创建Shuffle Service集群，详情请参见步骤一：创建集群。

使用限制

Spark集群仅支持与同一ACK集群下的Shuffle Service集群进行关联。

操作步骤

1. 登录阿里云E-MapReduce on ACK控制台。
2. 关联RSS。
   i. 在**集群管理**页面，单击已创建Spark集群的集群名称。
   ii. 在**集群详情**页面的基础信息区域，单击**关联ShuffleService**后面的**前往关联**。
   iii. 在**关联集群**的区域内，单击**新增**。
   iv. 在**关联集群**对话框中，选择已创建的Shuffle Service集群，单击**关联**。
3. （可选）配置RSS参数，详情请参见配置项说明。

7.5.2.4. 为Spark集群设置元数据

EMR on ACK支持使用数据湖元数据DLF（Data Lake Formation）和自建Hive Metastore元数据两种方式，为Spark集群设置元数据。本文为您介绍如何在EMR on ACK中设置Spark集群的元数据。

背景信息

因为数据湖元数据DLF具有高可用和易维护的特点，所以以下场景适合使用数据湖元数据：

- 当您的EMR集群均为生产环境时，您无需维护独立的元数据库。
- 横向使用多种大数据计算引擎时，元数据可以集中管理。例如，MaxCompute、Hologres和机器学习PAI等。
- 多个EMR集群时，可以统一管理元数据。

前提条件

- 已在E-MapReduce on ACK控制台创建Spark集群，详情请参见步骤一：创建集群。
- 使用数据湖元数据DLF方式时，需要确保已开通数据湖构建DLF，详情请参见快速入门。
- 使用自建Hive Metastore元数据方式时，需要确保已自行创建Hive Metastore服务，并且和创建的ACK集群可以网络连通。

方式一：使用数据湖元数据DLF（推荐）

1. 进入集群详情页面。
   i. 登录阿里云E-MapReduce on ACK控制台。
   ii. 在**集群管理**页面，单击目标集群的名称。
2. 在**集群详情**页面的**基础信息**区域，单击**数据湖构建 (DLF)**后面的**点击启用**。
3. 在**启用DLF**对话框中，单击**确定**。

   完成上述配置后，向该Spark集群提交的任务，会自动连接DLF元数据。

方式二：使用自建Hive Metastore元数据

1. 进入集群的配置页面。
   i. 登录阿里云E-MapReduce on ACK控制台。
   ii. 在**集群管理**页面，单击目标集群的名称。
   iii. 在**集群详情**页面，单击**配置**页签。
2. 在**服务配置**区域，单击**spark-defaults.conf**页签。
3. 添加自定义配置。
   i. 单击上方的**新增配置项**。
   ii. 添加Key为spark.hadoop.hive.metastore.uris，Value为thrift://<自建Hive的IP地址>:9083的配置项。
   该参数表示Hive Metastore使用Thrift协议连接的URI。参数值请根据您实际情况修改。
   iii. 单击**确定**。
   iv. 在**修改信息**对话框中，输入执行原因，打开**自动配置更新**开关，单击**保存**。
4. 部署客户端配置。
   i. 单击**部署客户端配置**。
   ii. 在弹出的对话框中，输入执行原因，单击**确定**。
   iii. 在**确认**对话框中，单击**确定**。

   完成上述配置后，向该Spark集群提交的任务，会自动连接自建的Hive Metastore。

7.5.2.5. 使用ECI弹性调度Spark作业

使用阿里云弹性容器实例（Elastic Container Instance）调度Spark作业，可以不受限于ACK集群的节点计算容量，灵活动态地按需创建Pod（容器组），有效地降低计算成本。本文为您介绍如何使用ECI弹性调度Spark作业。

背景信息

如果您需要使用更多ECI的高级功能，可以通过设置更多的Annotation（注解）对ECI按需进行参数配置，详情请参见ECI Pod Annotation。

前提条件

- 已在E-MapReduce on ACK控制台创建Spark集群，详情请参见快速入门。
- 已开通弹性容器实例服务，详情请参见使用流程。

操作步骤

1. 在ACK集群中安装ECI所需的虚拟节点，详情请参见步骤一：在ACK集群中部署ack-virtual-node组件。
2. 在EMR on ACK上提交Spark作业时，可以通过设置Label（标签）、Annotation或者Spark Conf来实现ECI调度Spark作业。

   提交Spark作业详情，请参见提交Spark作业。

   > **说明** 本文示例中版本以Spark 3.1.1（EMR-5.2.1-ack）为例，其他版本时请修改sparkVersion和mainApplicationFile的配置。示例中的参数描述，请参见spark-on-k8s-operator。

   - 方式一：配置Pod Label。

     设置参数alibabacloud.com/eci为true，将指定Pod调度到ECI上运行，参考示例如下。

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: spark-pi-eci
spec:
  type: Scala
  sparkVersion: 3.1.1
  mainClass: org.apache.spark.examples.SparkPi
  mainApplicationFile: "local:////opt/spark/examples/jars/spark-examples_2.12-3.1.1.jar"
  arguments:
    - "1000000"
  driver:
    cores: 2
    coreLimit: 2000m
    memory: 4g
  executor:
    cores: 4
    coreLimit: 4000m
    memory: 8g
    instances: 10
    # 通过配置Label，所有Executor使用ECI。
    labels:
      alibabacloud.com/eci: "true"
    #（可选）配置ECI镜像缓存，提升性能
    annotations:
      k8s.aliyun.com/eci-image-cache: "true"
```

- 方式二：配置Pod Annotation。

  设置参数alibabacloud.com/burst-resource为eci，将指定Pod调度到ECI上运行，Annotation取值包含两种类型：

  - eci：当集群普通节点的资源不足时，使用ECI。
  - eci_only：只使用ECI。

  参考示例如下。

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: spark-pi-eci
spec:
  type: Scala
  sparkVersion: 3.1.1
  mainClass: org.apache.spark.examples.SparkPi
  mainApplicationFile: "local:////opt/spark/examples/jars/spark-examples_2.12-3.1.1.jar"
  arguments:
    - "1000000"
  driver:
    cores: 2
    coreLimit: 2000m
    memory: 4g
  executor:
    cores: 4
    coreLimit: 4000m
    memory: 8g
    instances: 10
    # 通过配置Annotation，当Executor节点资源不足时使用ECI。
    annotations:
      alibabacloud.com/burst-resource: "eci"
      #（可选）配置ECI镜像缓存，提升性能
      k8s.aliyun.com/eci-image-cache: "true"
```

- 方式三：配置Spark Conf。

  您也可以通过增加Spark Conf来配置Pod Annotation，从而实现ECI调度。Annotaion取值与方式二：配置Pod Annotation相同。

  a. 进入spark-defaults.conf页签。

  a. 登录阿里云E-MapReduce on ACK控制台。

  b. 在**集群管理**页面，单击目标集群所在行的**配置**。

  c. 在**服务配置**区域，单击**spark-defaults.conf**页签。

b. 配置Spark集群以启用ECI。
   a. 单击上方的**新增配置项**。
   b. 在**新增配置项**对话框中，添加以下配置。

参数	描述
spark.kubernetes.driver.annotation.alibabacloud.com/burst-resource	Spark Driver是否使用ECI，取值为eci或eci_only。
spark.kubernetes.driver.annotation.k8s.aliyun.com/eci-image-cache	Spark Driver是否使用ECI镜像缓存，建议填写为true。
spark.kubernetes.executor.annotation.alibabacloud.com/burst-resource	Spark Executor是否使用ECI，取值为eci或eci_only
spark.kubernetes.executor.annotation.k8s.aliyun.com/eci-image-cache	Spark Executor是否使用ECI镜像缓存，建议填写为true。

   c. 单击**确定**。
   d. 在**修改信息**对话框中，输入**执行原因**，打开**自动配置更新**开关，单击**保存**。
c. 生效配置。
   a. 在**服务配置**区域，单击**部署客户端配置**。
   b. 在**部署SPARK客户端配置**对话框中，输入**执行原因**，单击**确定**。

3. （可选）如果您的作业需要读写OSS数据，或者使用了DLF元数据，访问云服务时还需要额外授予ECI权限。授权方式如下：
   - 方式一：通过角色授权ECI以实现免密访问。
     a. 在RAM控制台，创建可信实体为阿里云服务的RAM角色，详情请参见创建普通服务角色。

        > **说明** 受信服务选择为云服务器。

     b. 为创建的RAM角色授权，权限策略为AliyunOSSFullAccess和AliyunDLFFullAccess。

        为RAM角色授权详情，请参见为RAM角色授权。

     c. 在Spark作业中，按照如下方式添加Annotations，指定创建好的RAM角色即可。

```
annotations:
  k8s.aliyun.com/eci-ram-role-name: <创建的角色名>
```

   - 方式二：配置OSS AccessKey或DLF AccessKey。
     - 如果您的作业需要读写OSS数据，则需要在hadoopConf中额外增加AccessKey的配置。配置信息如下。

```
hadoopConf:
  fs.jfs.cache.oss.accessKeyId: <yourAccessKeyId>
  fs.jfs.cache.oss.accessKeySecret: <yourAccessKeySecret>
```

     - 如果您的作业开启了DLF，则需要在hadoopConf中额外增加AccessKey的配置。配置信息如下。

```
hadoopConf:
  dlf.catalog.accessKeyId: <yourAccessKeyId>
  dlf.catalog.accessKeySecret: <yourAccessKeySecret>
  dlf.catalog.akMode: "MANUAL"
```

7.5.2.6. 使用JindoFS加速OSS文件访问

本文介绍如何在E-MapReduce（简称EMR）on ACK的Spark集群中，通过Fluid和JindoRuntime加速访问OSS文件。

背景信息

Fluid是一个开源的Kubernetes原生的分布式数据集编排和加速引擎，主要服务于云原生场景下的数据密集型应用，例如大数据应用和AI应用等。JindoRuntime来源于阿里云EMR团队JindoFS，是基于C++实现的支撑Dataset数据管理和缓存的执行引擎，支持OSS对象存储。

使用Fluid和JindoRuntime，可以加速EMR on ACK上的Spark作业读取OSS文件，达到节省带宽流量的效果。

前提条件

- 已在EMR on ACK控制台创建Spark集群，详情请参见快速入门。
- 已在本地安装Helm，安装详细步骤请参见Install Helm。

操作流程

1. 步骤一：在ACK集群中安装Fluid
2. 步骤二：创建Dataset和JindoRuntime
3. 步骤三：配置Spark集群的连接参数

步骤一：在ACK集群中安装Fluid

1. 单击下载地址，下载Fluid安装包。选择最新版本下载。
2. 通过kubectl连接Kubernetes集群，详情请参见通过kubectl工具连接集群。
3. 创建名称为fluid-system的命名空间。

```
kubectl create ns fluid-system
```

4. 使用Helm安装Fluid。

```
helm install --set runtime.jindo.enabled=true fluid fluid-<version>.tgz
```

> **说明** 代码中的 `fluid-<version>.tgz`，需要替换为下载的Fluid安装包文件名称。

步骤二：创建Dataset和JindoRuntime

1. 新建*resource.yaml*文件，文件内容如下。

```
apiVersion: data.fluid.io/v1alpha1
kind: Dataset
metadata:
  name: hadoop
spec:
  mounts:
    - mountPoint: oss://test-bucket/
      options:
        fs.oss.accessKeyId: <OSS_ACCESS_KEY_ID>
        fs.oss.accessKeySecret: <OSS_ACCESS_KEY_SECRET>
        fs.oss.endpoint: <OSS_ENDPOINT>
      name: hadoop
---
apiVersion: data.fluid.io/v1alpha1
kind: JindoRuntime
metadata:
  name: hadoop
spec:
  replicas: 2
  tieredstore:
    levels:
      - mediumtype: HDD
        path: /mnt/disk1
        quota: 100Gi
        high: "0.9"
        low: "0.8"
```

文件内容包含以下两部分：

- 第一部分是Dataset CRD对象，其中描述了数据集的来源。代码示例中的test-bucket，表示需要加速的OSS Bucket。
- 第二部分是JindoRuntime，相当于启动了一个JindoFS的集群来提供缓存服务。具体的参数说明参见JindoRuntime文档。

  其中path表示节点上缓存的存储路径，建议使用数据盘容量充足的节点，path设置为磁盘挂载的路径。

2. 文件内容修改完成后，执行以下命令，创建Dataset和JindoRuntime。

```
kubectl create -f resource.yaml -n fluid-system
```

步骤三：配置Spark集群的连接参数

1. 执行如下命令，获取JindoFS的连接地址。

```
kubectl get svc -n fluid-system | grep jindo
```

返回信息如下图所示，本示例获取的*hadoop-jindofs-master-0.fluid-system:18000*即为JindoFS的连接地址。

2. 进入*spark-defaults.conf*页签。
   i. 登录阿里云E-MapReduce on ACK控制台。
   ii. 在**集群管理**页面，单击已创建的Spark集群的集群名。
   iii. 单击上方的**配置**页签。
   iv. 在**服务配置**区域，单击**spark-defaults.conf**页签。
3. 配置Spark集群以连接到JindoFS。
   i. 单击上方的**新增配置项**。
   ii. 在**新增配置项**对话框中，添加以下配置。

参数	描述
spark.hadoop.fs.xengine	固定值为jindofsx。
spark.hadoop.fs.jindofsx.data.cache.enable	数据缓存开关。固定值为**true**。
spark.hadoop.fs.jindofsx.meta.cache.enable	元数据缓存开关： ■ false（默认值）：禁用元数据缓存。 ■ true：启用元数据缓存。
spark.hadoop.fs.jindofsx.client.metrics.enable	固定值**true**。
spark.hadoop.fs.jindofsx.storage.connect.enable	固定值**true**。
spark.hadoop.fs.jindofsx.namespace.rpc.address	步骤1获取到的JindoFS的连接地址。本示例为*hadoop-jindofs-master-0.fluid-system:18000*。

   iii. 单击**确定**。
   iv. 在**修改信息**对话框中，输入**执行原因**，打开**自动配置更新**开关，单击**保存**。
4. 生效配置。
   i. 在**服务配置**区域，单击**部署客户端配置**。
   ii. 在**配置SPARK服务**对话框中，输入执行原因，单击**确定**。

   后续正常提交的作业，读取OSS时就可以体验JindoFS的缓存加速效果了。

8.常见问题

当您使用阿里云E-MapReduce（简称EMR）时，可以根据本文查找对应的问题场景和解决方案。

- 集群管理常见问题
- 数据开发常见问题
- 元数据管理常见问题
- 组件常见问题：
  - Airflow常见问题
  - HDFS常见问题
  - YARN常见
  - Hudi常见问题
  - Alluxio常见问题
  - Kudu常见问题
  - Spark常见问题
  - Hive常见问题
  - Impala常见问题
  - Zookeeper常见问题
  - Kafka常见问题
  - DeltaLake常见问题
  - Flume常见问题
  - Druid常见问题
  - ClickHouse常见问题
  - SmartData常见问题