ALIBABA CLOUD

Alibaba Cloud

E-MapReduce

最佳实践

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.数据分析
1.1. SparkSQL自适应执行

阿里云E-MapReduce 3.13.0及后续版本的SparkSQL支持自适应执行功能，可以用来解决Reduce个数的动态调整、数据倾斜和执行计划的动态优化问题。

使用限制

本文针对SparkSQL自适应执行涉及到的参数适用于Spark 2.x。如果您使用的是Spark 3.x，请参见Adaptive Query Execution。

解决问题

SparkSQL自适应执行解决以下问题：

- Shuffle partition个数

  目前SparkSQL中reduce阶段的task个数取决于固定参数spark.sql.shuffle.partition（默认值200），一个作业一旦设置了该参数，运行过程中的所有阶段的reduce个数都是同一个值。

  而对于不同的作业，以及同一个作业内的不同reduce阶段，实际的数据量大小可能相差很大，例如reduce阶段要处理的数据可能是10 MB，也有可能是100 GB，如果使用同一个值对实际运行效率会产生很大影响，例如10 MB的数据一个task就可以解决，如果spark.sql.shuffle.partition使用默认值200的话，那么10 MB的数据就要被分成200个task处理，增加了调度开销，影响运行效率。

  SparkSQL自适应框架可以通过设置Shuffle partition的上下限区间，在这个区间内对不同作业不同阶段的reduce个数进行动态调整。

  通过区间的设置，一方面可以大大减少调优的成本（不需要找到一个固定值），另一方面同一个作业内部不同reduce阶段的reduce个数也能动态调整。

  涉及参数如下。

属性名称	默认值	描述
spark.sql.adaptive.enabled	false	自适应执行框架的开关。
spark.sql.adaptive.minNumPostShufflePartitions	1	reduce个数区间最小值。
spark.sql.adaptive.maxNumPostShufflePartitions	500	reduce个数区间最大值。
spark.sql.adaptive.shuffle.targetPostShuffleInputSize	67108864	动态调整reduce个数的partition大小依据，如果设置为64 MB，则reduce阶段每个task最少处理64 MB的数据。
spark.sql.adaptive.shuffle.targetPostShuffleRowCount	20000000	动态调整reduce个数的partition条数依据，如设置20000000则reduce阶段每个task最少处理20000000条的数据。

- 数据倾斜

JOIN中会经常碰到数据倾斜的场景，导致某些task处理的数据过多，出现很严重的长尾。目前SparkSQL没有对倾斜的数据进行相关的优化处理。

SparkSQL自适应框架可以根据预先的配置在作业运行过程中自动检测是否出现倾斜，并对检测到的倾斜进行优化处理。

优化的主要逻辑是对倾斜的partition进行拆分由多个task来进行处理，最后通过UNION进行结果合并。

支持的JOIN类型如下。

JOIN类型	描述
Inner	左或右表均可处理倾斜。
Cross	左或右表均可处理倾斜。
LeftSemi	只对左表处理倾斜。
LeftAnti	只对左表处理倾斜。
LeftOuter	只对左表处理倾斜。
RightOuter	只对右表处理倾斜。

涉及参数如下。

属性名称	默认值	备注
spark.sql.adaptive.enabled	false	自适应执行框架的开关。
spark.sql.adaptive.skewedJoin.enabled	false	倾斜处理开关。
spark.sql.adaptive.skewedPartitionFactor	10	当一个partition的size大于该值（所有parititon大小的中位数） 且大于spark.sql.adaptive.skewedPartitionSizeThreshold，或者parition的条数大于该值（所有parititon条数的中位数）且大于spark.sql.adaptive.skewedPartitionRowCountThreshold， 才会被当做倾斜的partition进行相应的处理。
spark.sql.adaptive.skewedPartitionSizeThreshold	67108864	倾斜的partition大小不能小于该值。
spark.sql.adaptive.skewedPartitionRowCountThreshold	10000000	倾斜的partition条数不能小于该值。
spark.shuffle.statistics.verbose	false	打开后MapStatus会采集每个partition条数的信息，用于倾斜处理。

- Runtime执行计划优化

SparkSQL的Catalyst优化器会将SQL语句转换成物理执行计划，然后真正运行物理执行计划。但是Catalyst转换物理执行计划的过程中，由于缺少Statistics统计信息，或者Statistics统计信息不准等原因，实际转换的物理执行计划可能并不是最优的，例如转换为SortMergeJoinExec，但实际BroadcastJoin更合适。

SparkSQL自适应执行框架会在物理执行计划真正运行的过程中，动态的根据shuffle阶段shuffle write的实际数据大小，来调整是否可以用 BroadcastJoin来代替SortMergeJoin，提高运行效率。

涉及参数如下。

属性名称	默认值	备注
spark.sql.adaptive.enabled	false	自适应执行框架的开关。
spark.sql.adaptive.join.enabled	true	开关。
spark.sql.adaptiveBroadcastJoinThreshold	为spark.sql.autoBroadcastJoinThreshold设置的参数值	运行过程中用于判断是否满足BroadcastJoin条件。

2.数据迁移和同步
2.1. 通过Presto查询RDS MySQL数据库

本文介绍在同一VPC下创建的E-MapReduce集群和RDS MySQL实例，如何通过E-MapReduce集群的Presto查询RDS MySQL数据库信息。

前提条件

- 已创建集群，并且选择了Presto服务。详情请参见创建集群。
- 已购买RDS，详情请参见创建RDS MySQL实例。

> 说明 本文以RDS为例介绍。建议类型选择MySQL的5.7；系列选择高可用版。

背景信息

Presto的相关概念，请参见概述。

步骤一：配置Connector

1. 进入Presto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 > Presto**。
2. 添加Connector信息。
   i. 在Presto页面，单击**配置**。
   ii. 在**服务配置**区域，单击**connector1.properties**页签。
   如果您需要连接多个RDS MySQL数据库时，可以配置多个*connector.properties*。
   iii. 修改**connector.name**为**mysql**。
   iv. 单击右上角的**自定义配置**。
   v. 在**新增配置项**对话框中，配置如下参数。

配置项	描述
connection-user	数据库的用户名。本文示例是hiveuser。
connection-password	数据库的密码。
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-2ze5ipacsu8265qxxxxxxxx.mysql.rds.aliyuncs.com:3306*。

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. 重启服务使配置生效。
   i. 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**。
   iii. 单击**确定**。
   iv. 在**确认**对话框中，单击**确定**。

步骤二：查看RDS的数据库

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，连接Presto客户端。

```
presto --server emr-header-1:9090 --catalog hive --schema default --user hadoop
```

   返回如下信息，表示Presto连接成功。

```
presto:default>
```

3. 执行如下命令，查看**connector1.properties**下的Schema。

```
show schemas from connector1;
```

> **说明** `connector1` 是您在步骤一：配置Connector中配置的properties的名称。

   返回如下类似信息。

```
       Schema
--------------------
 emruser
 information_schema
 performance_schema
 sys
(4 rows)
```

步骤三：查询表数据

> **说明** 本文示例中的 `hive.default.tbl_department` 是您在Hive上创建的表， `connector1.emruser.tbl_employee` 是您在MySQL上创建的表。

- 查询*emruser.tbl_employee*表的数据。

```
select * from connector1.emruser.tbl_employee;
```

  返回类似如下表数据。

```
 id |    name    | dept_id | salary
----+------------+---------+---------
  1 | Ming Li    |       1 | 10000.0
  2 | Eric Cai   |       1 | 11000.0
  3 | Bonnie Liu |       2 | 11000.0
(3 rows)
```

- 查询*tbl_department*表的数据。

```
select * from hive.default.tbl_department;
```

返回类似如下表数据。

```
dept_id | dept_name
--------+----------
      1 | IT
      2 | Finance
(2 rows)
```

- 交叉查询表数据。

执行如下命令，交叉查询表*default.tbl_department*和*emruser.tbl_employee*的数据。

```
select * from hive.default.tbl_department a, connector1.emruser.tbl_employee b where a.de
pt_id = b.dept_id;
```

返回结果信息如下表所示。

```
dept_id |dept_name | id |    name    | dept_id | salary
--------+----------+----+------------+---------+---------
      2 | Finance  |  3 | Bonnie Liu |       2 | 11000.0
      1 | IT       |  2 | Eric Cai   |       1 | 11000.0
      1 | IT       |  1 | Ming Li    |       1 | 10000.0
(3 rows)
```

2.2. E-MapReduce数据迁移方案

在开发过程中我们通常会碰到需要迁移数据的场景，本文介绍如何将自建集群数据迁移到E-MapReduce集群中。

背景信息

适用范围：

- 线下Hadoop到E-MapReduce迁移。
- 线上ECS自建Hadoop到E-MapReduce迁移。

迁移场景：HDFS增量上游数据源包括RDS增量数据和Flume。

新旧集群网络打通

- 线下IDC自建Hadoop

  现在自建Hadoop迁移到E-MapReduce可以通过OSS进行过度，或者使用阿里云高速通道产品建立线下IDC和线上E-MapReduce所在VPC网络的连通。

- 利用ECS自建Hadoop

  由于VPC实现用户专有网络之间的逻辑隔离，E-MapReduce建议使用VPC网络。

  - 经典网络与VPC网络打通

    如果ECS自建Hadoop，需要通过ECS的classiclink的方式将经典网络和VPC网络打通，详情请参见建立ClassicLink连接。

  - VPC网络之间连通

    数据迁移一般需要较高的网络带宽连通，建议新旧集群尽量处在同一个区域的同一个可用区内。

HDFS数据迁移

- Distcp工具同步数据

  HDFS数据迁移可以通过Hadoop社区标准的DistCp工具迁移，可以实现全量和增量的数据迁移。为减轻现有集群资源压力，建议在新旧集群网络连通后在新集群执行**distcp**命令。

  - 全量数据同步

    ```
    hadoop distcp -pbugpcax -m 1000 -bandwidth 30 hdfs://oldclusterip:8020/user/hive/wareho
    use /user/hive/warehouse
    ```

  - 增量数据同步

    ```
    hadoop distcp -pbugpcax -m 1000 -bandwidth 30  -update -delete hdfs://oldclusterip:8020
    /user/hive/warehouse /user/hive/warehouse
    ```

  参数说明：

  - `oldclusterip`：填写旧集群namenode ip，多个namenode情况填写当前状态为active的。
  - `-p`：默认副本数为3，如想保留原有副本数，`-p`后加r如`-prbugpcax`。如果不同步权限和ACL，`-p`后去掉p和a。
  - `-m`：指定map数，和集群规模、数据量有关。例如集群有2000核CPU，就可以指定2000个map。
  - `-bandwidth`：指定单个map的同步速度，是靠控制副本复制速度实现的，是大概值。
  - `-update`：校验源文件和目标文件的checksum和文件大小，如果不一致源文件会更新掉目标集群数据，新旧集群同步期间还有数据写入，可以通过`-update`做增量数据同步。
  - `-delete`：如果源集群数据不存在，新集群的数据也会被删掉。

  > **说明**
  > - 迁移整体速度受集群间带宽、集群规模影响。同时文件越多，checksum需要的时间越长。如果迁移数据量大，可以先试着同步几个目录评估一下整体时间。如果只能在指定时间段内同步，可以将目录切为几个小目录，依次同步。
  > - 一般全量数据同步，需要有个短暂的业务停写，以启用双写双算或直接将业务切换到新集群上。

- HDFS权限配置

  HDFS有权限设置，确定旧集群是否有ACL规则，是否要同步，检查新旧集群dfs.permissions.enabled和dfs.namenode.acls.enabled的配置是否一致，按照实际需要修改。

如果有ACL规则要同步，distcp参数后要加 `-p` 同步权限参数。如果distcp操作提示xx集群不支持 ACL，说明对应集群没配置ACL规则。新集群没配置ACL规则可以修改配置并重启namenode。旧集群不支持，说明旧集群根本就没有ACL方面的设置，也不需要同步。

Hive元数据同步

- 概述

  Hive元数据，一般存在MySQL里，与一般MySQL同步数据相比，要注意两点：

  - Location变化
  - Hive版本对齐

  E-MapReduce支持Hive Meta DB：

  - 统一元数据库，E-MapReduce管控RDS，每个用户一个Schema
  - 用户自建RDS
  - 用户ECS自建MySQL

  为了保证迁移之后新旧数据完全一致，最好是在迁移的时候将老的metastore服务停掉，等迁移过去之后，再把旧集群上的 metastore 服务打开，然后新集群开始提交业务作业。

- 操作步骤：

  i. 将新集群的元数据库删除，直接输出命令 `drop database xxx` 。

  ii. 将旧集群的元数据库的表结构和数据通过 `mysqldump` 命令全部导出。

  iii. 替换location、Hive元数据中分区等信息均带有location信息的，带dfs nameservices前缀的表，如*hdfs://mycluster:8020/*，而E-MapReduce集群的nameservices前缀是统一的E-MapReduce-cluster，所以需要订正。

  订正的最佳方式是先导出数据 。

  ```
  mysqldump --databases hivemeta --single-transaction -u root -p > hive_databases.sql
  ```

  用sed替换*hdfs://oldcluster:8020/*为*hdfs://E-MapReduce-cluster/*，再导入新db中。

  ```
  mysql hivemeta -p < hive_databases.sql
  ```

  iv. 在新集群的界面上，停止掉hivemetastore服务。

  v. 登录新的元数据库， `create database` 创建数据库。

  vi. 在新的元数据库中，导入替换location字段之后的老元数据库导出来的所有数据。

  vii. 版本对齐，E-MapReduce的Hive版本一般是当前社区最新的稳定版，自建集群Hive版本可能会更老，所以导入的旧版本数据可能不能直接使用。需要执行 Hive的升级脚本（期间会有表、字段已存在的问题可以忽略），请参见Hive升级脚本。例如Hive从1.2 升级到2.3.0，需要依次执行**upgrade-1.2.0-to-2.0.0.mysql.sql**、**upgrade-2.0.0-to-2.1.0.mysql.sql**、**upgrade-2.1.0-to-2.2.0.mysql.sql**、**upgrade-2.2.0-to-2.3.0.mysql.sql**。脚本主要是建表，加字段，改内容，如有表已存在，字段已存在的异常可以忽略。

  viii. Meta数据全部订正后，就可以重启metaserver了。命令行输入 `hive` ，查询库和表、查询数据、验证数据的正确性。

Flume数据迁移

- Flume双写配置

在新集群上也开启flume服务，并且将数据按照和老集群完全一致的规则写入到新集群中。

- Flume分区表写入

  Flume数据双写，双写时需控制开始的时机，要保证flume在开始一个新的时间分区的时候来进行新集群的同步。 如flume每小时整点会同步所有的表，那就要整点之前，开启flume同步服务，这样flume在一个新的小时内写入的数据，在旧集群和新集群上是完全一致的。而不完整的旧数据在distcp的时候，全量的同步会覆盖它。而开启双写时间点后的新数据，在数据同步的时候不进行同步。 这个新的写入的数据，我们在划分数据阶段，记得不要放到数据同步的目录里。

作业同步

Hadoop、Hive、Spark或MR等如果有较大的版本升级，可能涉及作业改造，要视具体情况而定。

常见问题：

- Gateway OOM

  修改 */etc/ecm/hive-conf/hive-env.sh*。

  export HADOOP_HEAPSIZE=512改成1024。

- 作业执行内存不足

  ```
  set mapreduce.map.java.opts=-Xmx3072m
  ```

  `mapreduce.map.java.opts` 调整的是启动JVM虚拟机时，传递给虚拟机的启动参数，而默认值 `-Xmx3072m` 表示这个Java程序可以使用的最大堆内存数，一旦超过这个大小，JVM就会抛出Out of Memory异常，并终止进程。

  ```
  set mapreduce.map.memory.mb=3840
  ```

  `mapreduce.map.memory.mb` 设置的是Container的内存上限，这个参数由NodeManager读取并进行控制，当Container的内存大小超过了这个参数值，NodeManager会负责终止Container。

数据校验

由客户自行抽检报表完成。

Presto集群迁移

如果有单独的Presto集群仅仅用来做数据查询，需要修改 Hive 中配置文件，请参见Presto文档。

需要修改hive.properties：

- connector.name=hive-hadoop2
- hive.metastore.uri=thrift://E-MapReduce-header-1.cluster-500148414:9083
- hive.config.resources=/etc/ecm/hadoop-conf/core-site.xml, /etc/ecm/hadoop-conf/hdfs-site.xml
- hive.allow-drop-table=true
- hive.allow-rename-table=true
- hive.recursive-directories=true

附录

Hive 1.2升级到2.3的版本对齐示例。

```
source /usr/lib/hive-current/scripts/metastore/upgrade/mysql/upgrade-1.2.0-to-2.0.0.mysql.s
ql
CREATE TABLE COMPACTION_QUEUE (
  CQ_ID bigint PRIMARY KEY,
```

```
  CQ_DATABASE varchar(128) NOT NULL,
  CQ_TABLE varchar(128) NOT NULL,
  CQ_PARTITION varchar(767),
  CQ_STATE char(1) NOT NULL,
  CQ_TYPE char(1) NOT NULL,
  CQ_WORKER_ID varchar(128),
  CQ_START bigint,
  CQ_RUN_AS varchar(128),
  CQ_HIGHEST_TXN_ID bigint,
  CQ_META_INFO varbinary(2048),
  CQ_HADOOP_JOB_ID varchar(32)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
CREATE TABLE TXNS (
  TXN_ID bigint PRIMARY KEY,
  TXN_STATE char(1) NOT NULL,
  TXN_STARTED bigint NOT NULL,
  TXN_LAST_HEARTBEAT bigint NOT NULL,
  TXN_USER varchar(128) NOT NULL,
  TXN_HOST varchar(128) NOT NULL,
  TXN_AGENT_INFO varchar(128),
  TXN_META_INFO varchar(128),
  TXN_HEARTBEAT_COUNT int
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
CREATE TABLE HIVE_LOCKS (
  HL_LOCK_EXT_ID bigint NOT NULL,
  HL_LOCK_INT_ID bigint NOT NULL,
  HL_TXNID bigint,
  HL_DB varchar(128) NOT NULL,
  HL_TABLE varchar(128),
  HL_PARTITION varchar(767),
  HL_LOCK_STATE char(1) not null,
  HL_LOCK_TYPE char(1) not null,
  HL_LAST_HEARTBEAT bigint NOT NULL,
  HL_ACQUIRED_AT bigint,
  HL_USER varchar(128) NOT NULL,
  HL_HOST varchar(128) NOT NULL,
  HL_HEARTBEAT_COUNT int,
  HL_AGENT_INFO varchar(128),
  HL_BLOCKEDBY_EXT_ID bigint,
  HL_BLOCKEDBY_INT_ID bigint,
  PRIMARY KEY(HL_LOCK_EXT_ID, HL_LOCK_INT_ID),
  KEY HIVE_LOCK_TXNID_INDEX (HL_TXNID)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
CREATE INDEX HL_TXNID_IDX ON HIVE_LOCKS (HL_TXNID);
source /usr/lib/hive-current/scripts/metastore/upgrade/mysql/upgrade-1.2.0-to-2.0.0.mysql.s
ql
source /usr/lib/hive-current/scripts/metastore/upgrade/mysql/upgrade-2.0.0-to-2.1.0.mysql.s
ql
CREATE TABLE TXN_COMPONENTS (
  TC_TXNID bigint,
  TC_DATABASE varchar(128) NOT NULL,
  TC_TABLE varchar(128),
  TC_PARTITION varchar(767),
  FOREIGN KEY (TC_TXNID) REFERENCES TXNS (TXN_ID)
```

```
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
source /usr/lib/hive-current/scripts/metastore/upgrade/mysql/upgrade-2.0.0-to-2.1.0.mysql.s
ql
source /usr/lib/hive-current/scripts/metastore/upgrade/mysql/upgrade-2.1.0-to-2.2.0.mysql.s
ql
CREATE TABLE IF NOT EXISTS `NOTIFICATION_LOG`
(
    `NL_ID` BIGINT(20) NOT NULL,
    `EVENT_ID` BIGINT(20) NOT NULL,
    `EVENT_TIME` INT(11) NOT NULL,
    `EVENT_TYPE` varchar(32) NOT NULL,
    `DB_NAME` varchar(128),
    `TBL_NAME` varchar(128),
    `MESSAGE` mediumtext,
    PRIMARY KEY (`NL_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
CREATE TABLE IF NOT EXISTS `PARTITION_EVENTS` (
  `PART_NAME_ID` bigint(20) NOT NULL,
  `DB_NAME` varchar(128) CHARACTER SET latin1 COLLATE latin1_bin DEFAULT NULL,
  `EVENT_TIME` bigint(20) NOT NULL,
  `EVENT_TYPE` int(11) NOT NULL,
  `PARTITION_NAME` varchar(767) CHARACTER SET latin1 COLLATE latin1_bin DEFAULT NULL,
  `TBL_NAME` varchar(128) CHARACTER SET latin1 COLLATE latin1_bin DEFAULT NULL,
  PRIMARY KEY (`PART_NAME_ID`),
  KEY `PARTITIONEVENTINDEX` (`PARTITION_NAME`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
CREATE TABLE COMPLETED_TXN_COMPONENTS (
  CTC_TXNID bigint NOT NULL,
  CTC_DATABASE varchar(128) NOT NULL,
  CTC_TABLE varchar(128),
  CTC_PARTITION varchar(767)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
 source /usr/lib/hive-current/scripts/metastore/upgrade/mysql/upgrade-2.1.0-to-2.2.0.mysql.
sql
 source /usr/lib/hive-current/scripts/metastore/upgrade/mysql/upgrade-2.2.0-to-2.3.0.mysql.
sql
  CREATE TABLE NEXT_TXN_ID (
  NTXN_NEXT bigint NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
INSERT INTO NEXT_TXN_ID VALUES(1);
CREATE TABLE NEXT_LOCK_ID (
  NL_NEXT bigint NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
INSERT INTO NEXT_LOCK_ID VALUES(1);
```

2.3. 通过Flink作业处理OSS数据

本文介绍如何在Dataflow集群中运行Flink作业来消费OSS数据。

背景信息

本文示例使用的是EMR数据开发功能，目前已不推荐使用，因为该功能在2022年2月21日21点停止更新。

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：准备测试数据
3. 步骤三：创建并运行Flink作业
4. （可选）步骤四：查看作业提交日志和作业信息

步骤一：准备环境

在E-MapReduce上创建Flink模式的Dataflow集群，详情请参见创建集群。

> **说明** 本文以EMR-3.39.1版本的集群为例。

步骤二：准备测试数据

在创建Flink作业前，您需要在OSS上传测试数据。本示例上传一个*test.txt*文件，文件内容为 `Nothing is impossible for a willing heart. While there is a life, there is a hope~` 。

1. 登录 OSS管理控制台。
2. 创建存储空间并上传测试数据文件，详情请参见创建存储空间和上传文件。

   测试数据的上传路径在后续步骤的代码中会使用，本示例的上传路径为*oss://vvr-test/test.txt*。

步骤三：创建并运行Flink作业

1. 进入数据开发的项目列表页面。
   i. 通过阿里云账号登录阿里云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**数据开发**页签。
2. 在**数据开发**页面，创建项目，详情请参见项目管理。
3. 新建Flink类型作业。
   i. 在页面左侧，在需要操作的文件夹上单击右键，选择**新建作业**。
   ii. 在**新建作业**对话框中，输入**作业名称**和**作业描述**，从**作业类型**下拉列表中选择**Flink**作业类型。
   iii. 单击**确定**。
4. 编辑作业内容。

   作业内容示例如下。

   ```
   run -m yarn-cluster -yjm 1024 -ytm 1024 -ynm flink-oss-sample /usr/lib/flink-current/examples/batch/WordCount.jar --input oss://vvr-test/test.txt
   ```

   示例代码中的关键参数说明如下：

   - */usr/lib/flink-current/examples/batch/WordCount.jar*：DataFlow集群内置的Flink WordCount作业，代码详细信息请参见官方代码仓库。

- *oss://vvr-test/test.txt*：上传到OSS的测试数据。

5. 作业配置完成后，单击右上方的**运行**。

   在**运行作业**对话框中，选择**执行集群**为新建的Flink模式的Dataflow集群。

6. 单击**确定**。

   作业成功运行后，即成功实现了在E-MapReduce集群上运行Flink作业处理OSS数据。日志中会打印如下信息。

```
(a,3)
(for,1)
(heart,1)
(hope,1)
(impossible,1)
(is,3)
(life,1)
(nothing,1)
(there,2)
(while,1)
(willing,1)
==========JOB OUTPUT END==========
```

（可选）步骤四：查看作业提交日志和作业信息

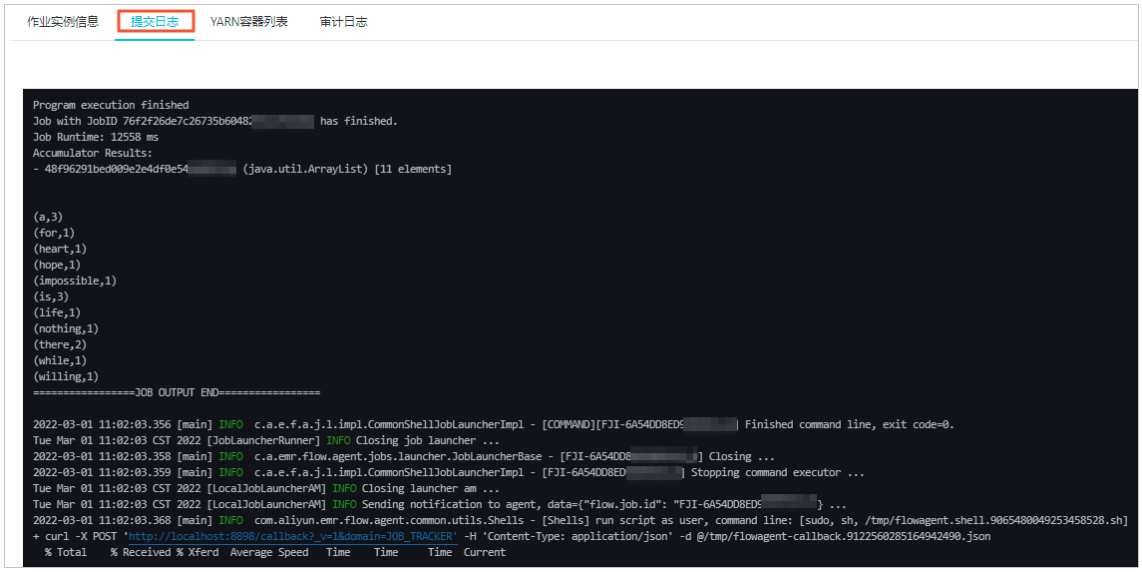
如果需要定位作业失败的原因或了解作业的详细信息，则您可以查看作业的日志和作业信息。

1. 查看作业提交日志。

   当前提交日志支持在E-MapReduce控制台查看，也支持在SSH客户端查看。

   - 提交作业后，您可以在E-MapReduce控制台的**运行记录**页签，单击待查看作业所在行的**详情**。

   单击**提交日志**页签，可以查看详细的日志信息。



   - 通过SSH客户端登录到header节点，查看提交的日志信息。

默认情况下，根据Flink的**log4j**配置（详情请参见*/etc/ecm/flink-conf/log4j-yarn-session.properties*），Flink客户端的提交日志会保存在*/mnt/disk1/log/flink/flink-{user}-client-{hostname}.log*。

其中，user为提交Flink作业的用户，hostname为提交作业所在的节点。以root用户在**emr-header-1**节点提交Flink作业为例，日志的路径为*/mnt/disk1/log/flink/flink-flink-historyserver-0-emr-header-1.cluster-126601.log*。

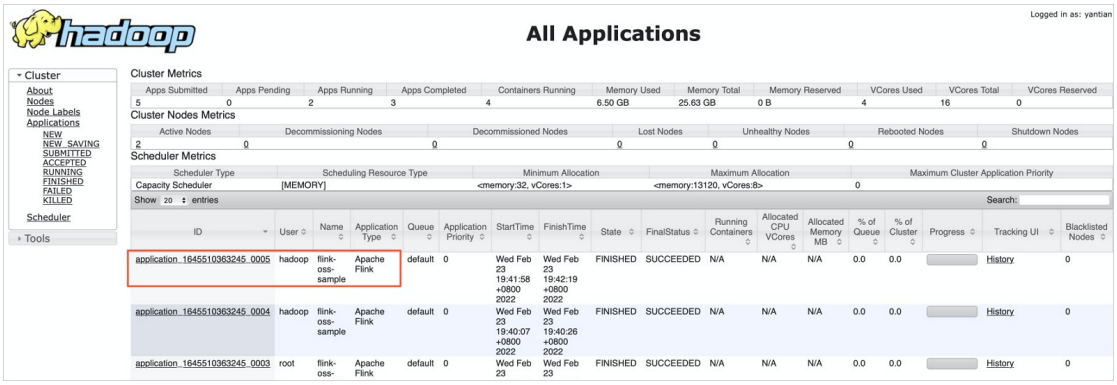
2. 查看作业信息。

通过**Yarn UI**可以查看Flink作业的信息。访问**Yarn UI**有SSH隧道和Knox两种方式，SSH隧道方式请参见通过SSH隧道方式访问开源组件Web UI，Knox方式请参见Knox和访问链接与端口。下面以Knox方式为例进行介绍。

i. 在E-MapReduce控制台的**访问链接与端口**页面中，单击**Yarn UI**后的链接。

ii. 在Hadoop控制台，单击作业的ID。

查看作业运行详情。



详细信息如下。

iii. 如果您需要查看运行中的Flink作业，则可以在作业详情页面，单击Tracking URL后面的链接，进入Flink Dashboard查看。

iv. 作业运行结束后，您可以查看所有已经完成的作业列表和日志。

详细信息，请参见如何访问DataFlow集群中的Flink HistoryServer? 和如何查看Flink作业的运行状态？。

2.4. 使用E-MapReduce进行MySQL Binlog日志准实时传输

本文介绍如何利用阿里云SLS插件功能和E-MapReduce集群进行MySQL Binlog的准实时传输。

前提条件

- 已在E-MapReduce上创建Hadoop集群，详情请参见创建集群。
- 已创建MySQL类型的数据库（例如RDS或DRDS）。MySQL必须开启Binlog，且Binlog必须为ROW模式。

  本文以RDS为例介绍，详情请参见创建RDS MySQL实例。

> **说明** RDS默认已开启Binlog功能。

操作步骤

1. 连接MySQL实例并添加用户权限。
   i. 使用命令方式连接MySQL实例，详情请参见通过客户端、命令行连接RDS MySQL。
   ii. 执行以下命令添加用户权限。

```
CREATE USER canal IDENTIFIED BY 'canal';
GRANT SELECT, REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'canal'@'%';
FLUSH PRIVILEGES;
```

2. 为SLS服务添加对应的配置文件，详情请参见采集MySQL Binlog。

> **说明** 本文创建的Project名称为canaltest，Logstore名称为canal。

   在SLS控制台查看日志数据是否上传成功，如果未上传成功请根据SLS的采集日志排查。

3. 编译JAR包并上传至OSS。
   i. 在本地打开Git复制示例代码。

```
git clone https://github.com/aliyun/aliyun-emapreduce-demo.git
```

   ii. 修改示例代码。

   示例代码中已经有LoghubSample类，该类主要用于从SLS采集数据并打印。以下示例为修改后的代码。

```
package com.aliyun.emr.example
import org.apache.spark.SparkConf
import org.apache.spark.storage.StorageLevel
import org.apache.spark.streaming.aliyun.logservice.LoghubUtils
import org.apache.spark.streaming.{Milliseconds, StreamingContext}
object LoghubSample {
def main(args: Array[String]): Unit = {
if (args.length < 7) {
 System.err.println(
   """Usage: bin/spark-submit --class LoghubSample examples-1.0-SNAPSHOT-shaded.jar
     |
     |
   """.stripMargin)
 System.exit(1)
}
val loghubProject = args(0)
val logStore = args(1)
val loghubGroupName = args(2)
val endpoint = args(3)
val accessKeyId = args(4)
val accessKeySecret = args(5)
val batchInterval = Milliseconds(args(6).toInt * 1000)
val conf = new SparkConf().setAppName("Mysql Sync")
//    conf.setMaster("local[4]");
val ssc = new StreamingContext(conf, batchInterval)
val loghubStream = LoghubUtils.createStream(
 ssc,
 loghubProject,
 logStore,
 loghubGroupName,
 endpoint,
 1,
 accessKeyId,
 accessKeySecret,
 StorageLevel.MEMORY_AND_DISK)
loghubStream.foreachRDD(rdd =>
   rdd.saveAsTextFile("/mysqlbinlog")
)
ssc.start()
ssc.awaitTermination()
}
}
```

示例代码主要修改 `loghubStream.foreachRDD(rdd => rdd.saveAsObjectFile("/mysqlbinlog") )` 为 `loghubStream.foreachRDD(rdd => rdd.saveAsTextFile("/mysqlbinlog"))` ，以便于在E-MapReduce中运行时，保存Spark Streaming中流出来的数据至EMR的HDFS。

iii. 您可以在本地完成代码调试后，通过如下命令打包。

```
mvn clean install
```

iv. 上传JAR包至OSS。

在OSS上创建存储空间和上传文件，详情请参见创建存储空间和上传文件。

> **说明** 本示例在OSS上创建的Bucket为*EMR-test*，上传*examples-1.1-shaded.jar*至*EMR-test/jar*目录。

4. 创建Spark作业。

i. 进入**作业编辑**页面。

a. 通过阿里云账号登录阿里云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**数据开发**页签。

d. 在**项目列表**页面，单击待编辑项目所在行的**作业编辑**。

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

iii. 输入**作业名称**、**作业描述**，在**作业类型**下拉列表中选择**Spark**作业类型。

iv. 单击**确定**。

v. 在**作业内容**中，填写提交该作业需要提供的命令行参数。

```
--master yarn --deploy-mode client --driver-memory 4g --executor-memory 2g --executor-cores 2 --class com.aliyun.EMR.example.LoghubSample ossref://EMR-test/jar/examples-1.1-shaded.jar canaltest canal sparkstreaming <SLS_endpoint> <SLS_access_id> <SLS_secret_key> 1
```

参数	说明
SLS_endpoint	SLS的EndPoint。
SLS_access_id	您阿里云账号的AccessKey ID。
SLS_secret_key	您阿里云账号的AccessKey Secret。

> **说明** 本示例代码中最后的1表示代码示例中的batchInterval，即Spark作业batch的大小，其余参数详情请参见Spark-Submit参数设置说明。

vi. 单击**保存**。

5. 运行作业。

i. 在作业编辑页面，单击上方的**运行**。

ii. 在弹出的**运行作业**对话框中，从**执行集群**列表中，选择已创建的Hadoop集群。

iii. 单击**确定**。

6. 查看mysqlbinlog文件。

i. 通过SSH方式连接集群，详情请参见登录集群。

ii. 执行如下命令查看mysqlbinlog目录下的文件。

```
hadoop fs -ls /mysqlbinlog
```

您还可以通过执行命令 `hadoop fs -cat /mysqlbinlog/part-00000` 查看文件内容。

2.5. 通过Spark Streaming作业处理Kafka数据

本文介绍如何使用阿里云E-MapReduce创建的Hadoop和Kafka集群，运行Spark Streaming作业以消费Kafka数据。

前提条件

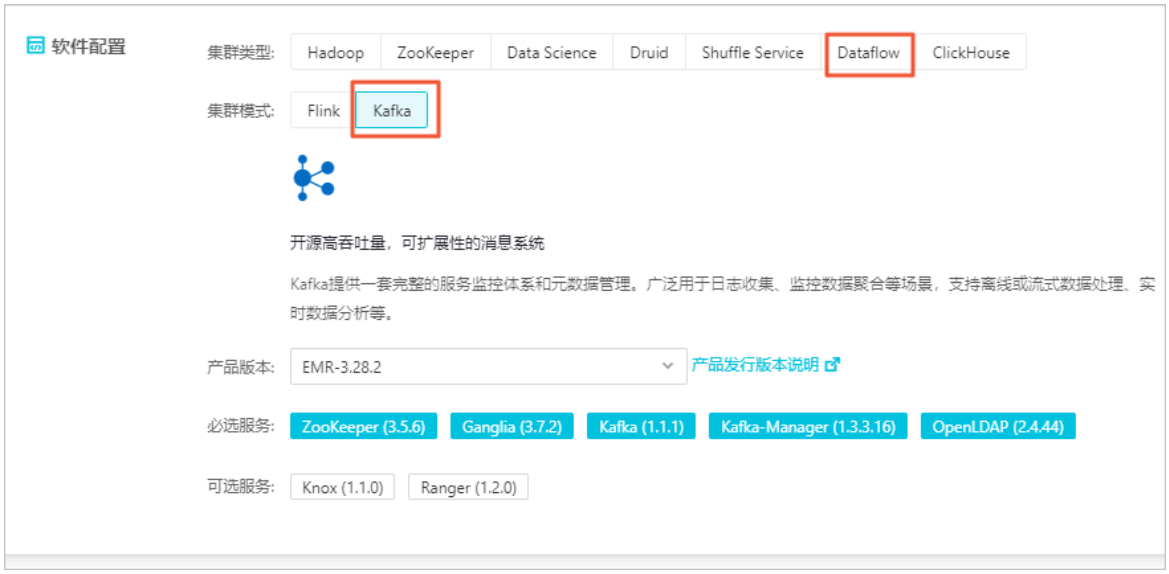
- 已开通E-MapReduce服务。
- 已完成云账号的授权，详情请参见角色授权。
- 本地安装了PuTTY和文件传输工具（SSH Secure File Transfer Client）。

步骤一：创建Hadoop集群和Kafka集群

创建同一个安全组下的Hadoop和Kafka集群。创建详情请参见创建集群。

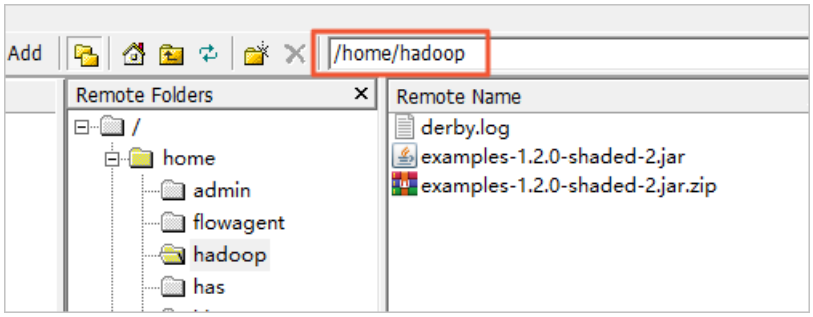
1. 登录阿里云E-MapReduce控制台。
2. 创建Hadoop集群。

3. 创建Kafka集群。



步骤二：获取JAR包并上传到Hadoop集群

1. 获取JAR包（examples-1.2.0-shaded-2.jar.zip）。
2. 使用文件传输工具，上传JAR包至Hadoop集群Master节点的*/home/hadoop*路径下。



步骤三：在Kafka集群上创建Topic

本示例将创建一个分区数为10、副本数为2、名称为test的Topic。

1. 登录Kafka集群的Master节点，详情请参见登录集群。
2. 通过如下命令创建Topic。

```
/usr/lib/kafka-current/bin/kafka-topics.sh --partitions 10 --replication-factor 2 --zoo
keeper emr-header-1:2181 /kafka-1.0.0 --topic test --create
```

> **说明** 创建Topic后，请保留该登录窗口，后续步骤仍将使用。

步骤四：运行Spark Streaming作业

本示例将运行一个流式单词统计（WordCount）的作业。

1. 登录Hadoop集群的Master节点，详情请参见登录集群。
2. 执行如下作业命令，进行流式单词统计（WordCount）。

```
spark-submit --class com.aliyun.emr.example.spark.streaming.KafkaSample  /home/hadoop/e
xamples-1.2.0-shaded-2.jar 192.168.xxx.xxx:9092 test 5
```

关键参数说明如下：

参数	描述
`192.168.xxx.xxx`	Kafka集群中任一Kafka Broker组件的内网IP地址。IP地址如Kafka集群组件所示。
`test`	Topic名称。
`5`	时间间隔。

Kafka集群组件

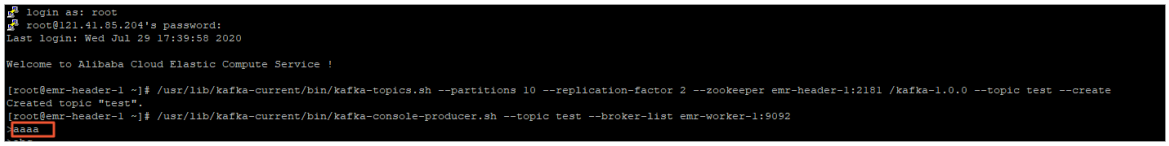
步骤五：使用Kafka发布消息

1. 在Kafka集群的命令行窗口，执行如下命令运行Kafka的生产者。

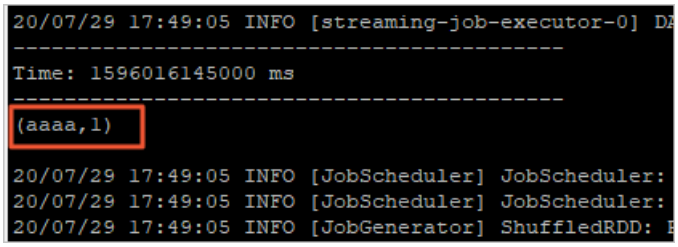
```
/usr/lib/kafka-current/bin/kafka-console-producer.sh --topic test --broker-list emr-worker-1:9092
```

2. 在Kafka集群的登录窗口中输入文本，在Hadoop集群的登录窗口中，会实时显示文本的统计信息。

   例如，在Kafka集群的登录窗口输入如下信息。



Hadoop集群的登录窗口会输出如下信息。

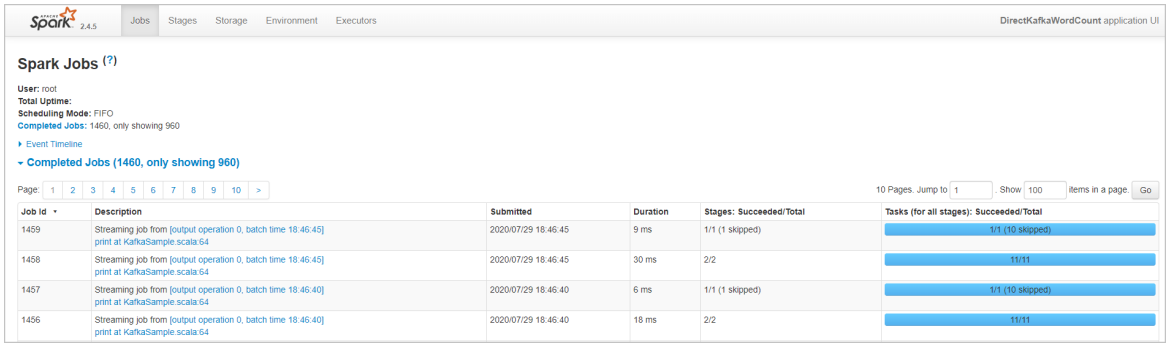


步骤六：查看Spark Streaming作业状态

1. 在E-MapReduce控制台的**集群管理**页面。
2. 单击创建的Hadoop集群所在行的**详情**。
3. 在左侧导航栏，单击**访问链接与端口**。
4. 单击**Spark History Server UI**所在行的链接。

5. 在History Server页面，单击待查看的App ID。

   您可以查看Spark Streaming作业的状态。



2.6. 通过Kafka Connect进行数据迁移

在流式数据处理过程中，E-MapReduce经常需要在Kafka与其他系统间进行数据同步或者在Kafka集群间进行数据迁移。本文向您介绍如何在E-MapReduce上通过Kafka Connect快速地实现Kafka集群间的数据迁移。

背景信息

Kafka Connect是一种可扩展的、可靠的、用于Kafka与其他系统间进行数据同步或者在Kafka集群间进行流式数据传输的工具。例如，Kafka Connect可以获取数据库的binlog数据，将数据库数据同步至Kafka集群，从而达到迁移数据库数据的目的。由于Kafka集群可对接流式处理系统，所以还可以间接实现数据库对接下游流式处理系统的目的。同时，Kafka Connect还提供了REST API接口，方便您创建和管理Kafka Connect。

Kafka Connect分为Standalone和Distributed两种运行模式。在Standalone模式下，所有的worker都在一个进程中运行。相比于Standalone模式，Distributed模式更具扩展性和容错性，是最常用的方式，也是生产环境推荐使用的模式。

本文介绍如何在E-MapReduce上使用Kafka Connect的REST API接口在Kafka集群间进行数据迁移，Kafka Connect使用distributed模式。

前提条件

- 已开通E-MapReduce服务。
- 已完成云账号的授权，详情请参见角色授权。

注意事项

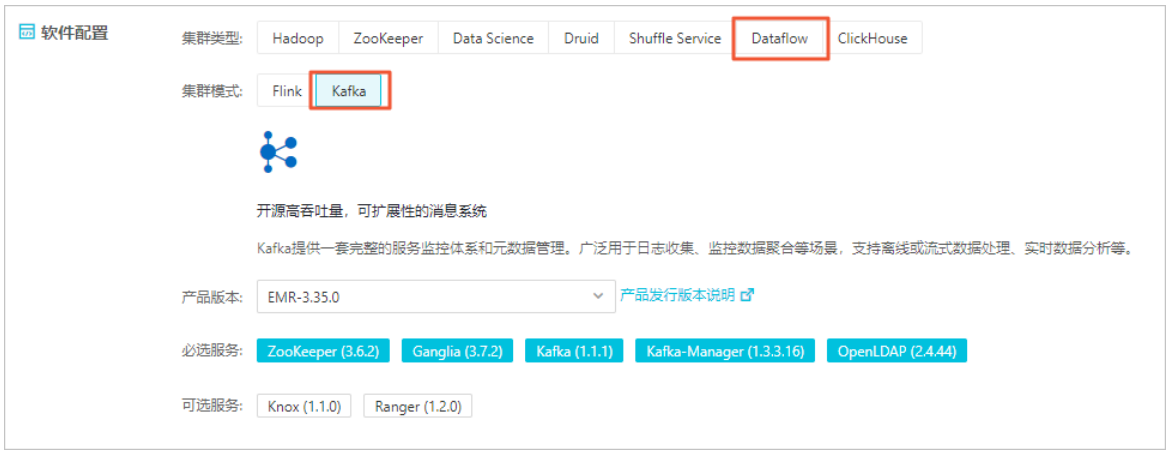
本文使用了EMRReplicatorSourceConnector，该Connector只有当Kafka组件版本是EMR Kafka 1.1.1版本时才支持使用。EMR Kafka 2.12-2.4.x之后版本，如果您需要迁移数据，可以使用MirrorMaker 2组件进行数据迁移，具体操作请参见使用MirrorMaker 2.0同步数据。

步骤一：创建Kafka集群

在EMR上创建源Kafka集群和目的Kafka集群。Kafka Connect安装在Task节点上，所以目的Kafka集群必须创建Task节点。集群创建好后，Task节点上Kafka Connect服务会默认启动，端口号为8083。

1. 登录阿里云E-MapReduce控制台。
2. 创建源Kafka集群和目的Kafka集群，详情请参见创建集群。

> **注意**
> - 源Kafka集群和目的Kafka集群创建在同一个安全组下。
>
> 如果源Kafka集群和目的kafka集群不在同一个安全组下，则两者的网络默认是不互通的，您需要对两者的安全组分别进行相关配置，以使两者的网络互通。
> - 创建目的Kafka集群后，需要扩容Task节点，详情请参见新增机器组。



步骤二：准备待迁移数据Topic

在源Kafka集群上创建一个名称为**connect**的Topic。

1. 以SSH方式登录到源Kafka集群的Master节点（本例为**emr-header-1**）。
2. 以root用户运行如下命令，创建一个名称为**connect**的Topic。

```
kafka-topics.sh --create --zookeeper emr-header-1:2181 --replication-factor 2 --partitions 10 --topic connect
```

```
[root@emr-header-1 ~]# kafka-topics.sh --create --zookeeper emr-header-1:2181 --replication-factor 2 --partitions 10 --topic connect
Created topic "connect".
[root@emr-header-1 ~]#
```

> **说明** 完成上述操作后，请保留该登录窗口，后续仍将使用。

步骤三：创建Kafka Connect的connector

在目的Kafka集群的Task节点上，使用 `curl` 命令，通过JSON数据创建一个Kafka Connect的connector。

1. 自定义Kafka Connect配置。

进入阿里云E-MapReduce控制台目的Kafka集群Kafka服务的**配置**页面，在**connect-distributed.properties**中自定义**offset.storage.topic**、**config.storage.topic**和**status.storage.topic**三个配置项，详情请参见管理组件参数。

Kafka Connect会将offsets、configs和任务状态保存在Topic中，Topic名对应**offset.storage.topic**、**config.storage.topic**和**status.storage.topic**三个配置项。Kafka Connect会自动使用默认的partition和replication factor创建这三个Topic，其中partition和repication factor配置项保存在*/etc/ecm/kafka-conf/connect-distributed.properties*文件中。

2. 以SSH方式登录到目的Kafka集群的Master节点（本例为**emr-header-1**）。
3. 切换至Task节点（本例为**emr-worker-3**）。

```
[root@emr-header-1 ~]# ssh emr-worker-3
root@emr-worker-3's password:
Last login: Fri Sep 18 11:23:39 2020 from 192.

Welcome to Alibaba Cloud Elastic Compute Service !

[root@emr-worker-3 ~]#
```

4. 以root用户运行如下命令创建一个Kafka Connect。

```
curl -X POST -H "Content-Type: application/json" --data '{"name": "connect-test", "conf
ig": { "connector.class": "EMRReplicatorSourceConnector", "key.converter": "org.apache.
kafka.connect.converters.ByteArrayConverter", "value.converter": "org.apache.kafka.conn
ect.converters.ByteArrayConverter", "src.kafka.bootstrap.servers": "${src-kafka-ip}:909
2", "src.zookeeper.connect": "${src-kafka-curator-ip}:2181", "dest.zookeeper.connect":
"${dest-kafka-curator-ip}:2181", "topic.whitelist": "${source-topic}", "topic.rename.fo
rmat": "${dest-topic}", "src.kafka.max.poll.records": "300" } }' http://emr-worker-3:80
83/connectors
```

在JSON数据中，name字段代表创建的Kafka Connect的名称，本例为*connect-test*；config字段需要根据实际情况进行配置，关键变量的说明如下。

变量	说明
${source-topic}	源Kafka集群中需要迁移的Topic，多个Topic需用英文逗号（,）隔开，例如**connect**。
${dest-topic}	目的Kafka集群中迁移后的Topic，例如**connect.replica**。
${src-kafka-curator-hostname}	源Kafka集群中安装了ZooKeeper服务的节点的内网IP地址。
${dest-kafka-curator-hostname}	目的Kafka集群中安装了ZooKeeper服务的节点的内网IP地址。

> **说明** 完成上述操作后，请保留该登录窗口，后续仍将使用。

步骤四：查看Kafka Connect和Task节点状态

查看Kafka Connect和Task节点信息，确保两者的状态正常。

1. 返回到目的Kafka集群的Task节点（本例为**emr-worker-3**）的登录窗口。

2. 以root用户运行如下命令查看所有的Kafka Connect。

```
curl emr-worker-3:8083/connectors
```

```
[root@emr-worker-3 ~]# curl emr-worker-3:8083/connectors
["connect-test"][root@emr-worker-3 ~]#
```

3. 以root用户运行如下命令查看本例创建的Kafka Connect（本例为**connect-test**）的状态。

```
curl emr-worker-3:8083/connectors/connect-test/status
```

```
[root@emr-worker-3 ~]# curl emr-worker-3:8083/connectors/connect-test/status
{"name":"connect-test","connector":{"state":"RUNNING","worker_id":"192.168.  :8083"},"tasks":[{"state":"RUNNING","id":0,"worker_id":"192.168.  :8083"}],"type":"source"}[r
oot@emr-worker-3 ~]#
```

确保Kafka Connect（本例为**connect-test**）的状态为**RUNNING**。

4. 以root用户运行如下命令查看Task节点信息。

```
curl emr-worker-3:8083/connectors/connect-test/tasks
```

```
[root@emr-worker-3 ~]# curl emr-worker-3:8083/connectors/connect-test/status
{"name":"connect-test","connector":{"state":"RUNNING","worker_id":"192.168.  :8083"},"tasks":[{"state":"RUNNING","id":0,"worker_id":"192.168.  :8083"}],"type":"source"}[r
oot@emr-worker-3 ~]# curl emr-worker-3:8083/connectors/connect-test/tasks
[{"id":{"connector":"connect-test","task":0},"config":{"connector.class":"EMRReplicatorSourceConnector","src.zookeeper.connect":"emr-header-1.cluster-127390:2181","topic.rename.f
ormat":"connect.replica","dest.zookeeper.connect":"emr-header-1.cluster-127499:2181","task.class":"org.apache.kafka.connect.replicator.EMRReplicatorSourceTask","src.kafka.max.pol
l.records":"300","name":"connect-test","task.id":"connect-test-0","value.converter":"org.apache.kafka.connect.converters.ByteArrayConverter","key.converter":"org.apache.kafka.con
nect.converters.ByteArrayConverter","src.kafka.bootstrap.servers":"192.168.  :9092","topic.whitelist":"connect","partition.assignment":"AAAAAAABAAdjb25uZWN0AAAACgAAAAAAAAABAA
AAAgAAAAMAAAAEAAAABQAAAAYAAAAHAAAACAAAAAkAAAA"}}][root@emr-worker-3 ~]#
```

确保Task节点的返回信息中无错误信息。

步骤五：生成待迁移数据

通过命令向源集群中的**connect** Topic发送待迁移的数据。

1. 返回到源Kafka集群的header节点（本例为**emr-header-1**）的登录窗口。
2. 以root用户运行如下命令向**connect**的Topic发送数据。

```
kafka-producer-perf-test.sh --topic connect --num-records 100000 --throughput 5000 --re
cord-size 1000 --producer-props bootstrap.servers=emr-header-1:9092
```

当提示如下信息，则表示已经成功生成待迁移数据。

```
[root@emr-header-1 ~]# kafka-producer-perf-test.sh --topic connect --num-records 100000 --throughput 5000 --record-size 1000   --producer-props bootstrap.servers=emr-header-1:909
2
24992 records sent, 4997.4 records/sec (4.77 MB/sec), 4.5 ms avg latency, 149.0 max latency.
25025 records sent, 5005.0 records/sec (4.77 MB/sec), 0.8 ms avg latency, 25.0 max latency.
25000 records sent, 5000.0 records/sec (4.77 MB/sec), 0.7 ms avg latency, 22.0 max latency.
24972 records sent, 4884.0 records/sec (4.66 MB/sec), 0.8 ms avg latency, 122.0 max latency.
100000 records sent, 4901.960784 records/sec (4.67 MB/sec), 1.73 ms avg latency, 393.00 ms max latency, 1 ms 50th, 4 ms 95th, 29 ms 99th, 77 ms 99.9th.
[root@emr-header-1 ~]#
```

步骤六：查看数据迁移结果

生成待迁移数据后，Kafka Connect会自动将这些数据迁移到目的集群的相应文件（本例为**connect.replica**）中。

1. 返回到目的Kafka集群的Task节点（本例为**emr-worker-3**）的登录窗口。
2. 以root用户运行如下命令查看数据迁移是否成功。

```
kafka-consumer-perf-test.sh --topic connect.replica --broker-list emr-header-1:9092 --m
essages 100000
```

```
[root@emr-worker-3 ~]# kafka-consumer-perf-test.sh  --topic connect.replica --broker-list emr-header-1:9092 --messages 1000000
start.time, end.time, data.consumed.in.MB, MB.sec, data.consumed.in.nMsg, nMsg.sec, rebalance.time.ms, fetch.time.ms, fetch.MB.sec, fetch.nMsg.sec
2019-07-22 10:13:17:855, 2019-07-22 10:13:32:055, 95.3674, 6.7160, 100000, 7042.2535, 3019, 11181, 8.5294, 8943.7439
[root@emr-worker-3 ~]#
```

从上述返回结果可以看出，在源Kafka集群发送的100000条数据已经迁移到了目的Kafka集群。

小结

本文介绍并演示了使用Kafka Connect在Kafka集群间进行数据迁移的方法。如果需要了解Kafka Connect更详细的使用方法，请参见Kafka官网资料和REST API。

2.7. 通过PyFlink作业处理Kafka数据

本文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。

前提条件

- 已开通E-MapReduce服务。
- 已完成云账号的授权，详情请参见角色授权。
- 已创建项目，详情请参见项目管理。
- 本地安装了PuTTY和文件传输工具（SSH Secure File Transfer Client）。

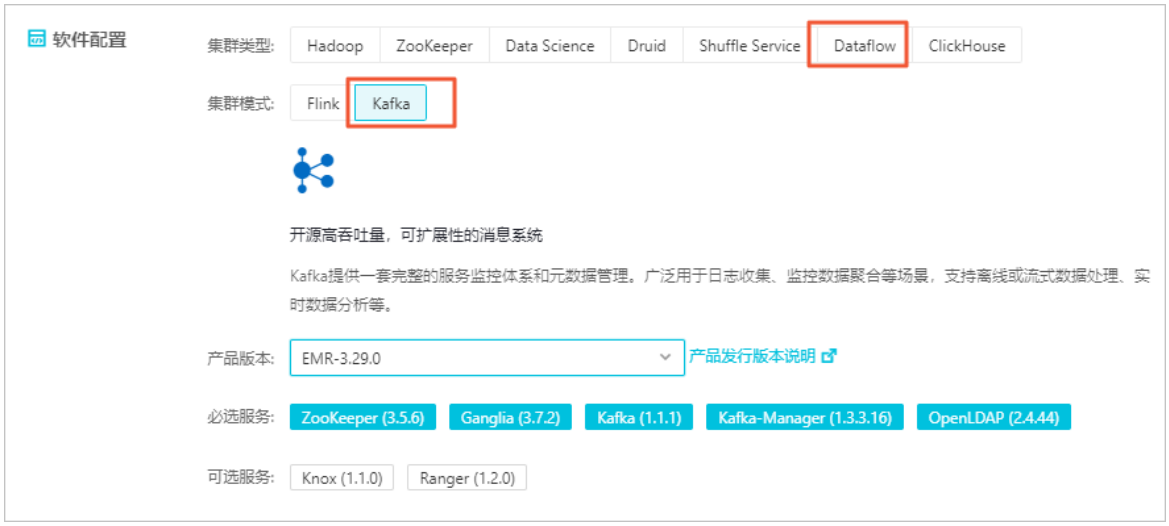
步骤一：创建Hadoop集群和Kafka集群

创建同一个安全组下的Hadoop和Kafka集群。创建详情请参见创建集群。

> **说明** 本文以EMR-3.29.0为例介绍。

1. 登录阿里云E-MapReduce控制台。
2. 创建Hadoop集群，并选择Flink服务。

3. 创建Kafka集群。



步骤二：在Kafka集群上创建Topic

本示例将创建两个分区数为10、副本数为2、名称为payment_msg和results的Topic。

1. 登录Kafka集群的Master节点，详情请参见登录集群。
2. 执行如下命令，创建名称为payment_msg的Topic。

```
/usr/lib/kafka-current/bin/kafka-topics.sh --partitions 10 --replication-factor 2 --zoo
keeper emr-header-1:2181 /kafka-1.0.0 --topic payment_msg --create
```

3. 执行如下命令，创建名称为results的Topic。

```
/usr/lib/kafka-current/bin/kafka-topics.sh --partitions 10 --replication-factor 2 --zoo
keeper emr-header-1:2181 /kafka-1.0.0 --topic results --create
```

> **说明** 创建Topic后，请保留该登录窗口，后续步骤仍将使用。

步骤三：准备测试数据

在步骤二中Kafka集群的命令行窗口，执行如下命令，不断生成测试数据。

```
python3 -m pip install kafka
rm -rf produce_data.py
cat>produce_data.py<<EOF
import random
import time, calendar
from random import randint
from kafka import KafkaProducer
from json import dumps
from time import sleep
def write_data():
    data_cnt = 20000
    order_id = calendar.timegm(time.gmtime())
    max_price = 100000
    topic = "payment_msg"
    producer = KafkaProducer(bootstrap_servers=['emr-worker-1:9092'],
                             value_serializer=lambda x: dumps(x).encode('utf-8'))
    for i in range(data_cnt):
        ts = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
        rd = random.random()
        order_id += 1
        pay_amount = max_price * rd
        pay_platform = 0 if random.random() < 0.9 else 1
        province_id = randint(0, 6)
        cur_data = {"createTime": ts, "orderId": order_id, "payAmount": pay_amount, "payPla
tform": pay_platform, "provinceId": province_id}
        producer.send(topic, value=cur_data)
        sleep(0.5)
if __name__ == '__main__':
    write_data()
EOF
python3 produce_data.py
```

步骤四：创建并运行PyFlink作业

1. 登录Hadoop集群的Master节点，详情请参见登录集群。
2. 执行如下命令，生成*lib.jar*和*job.py*。

```
rm -rf job.py
cat>job.py<<EOF
import os
from pyflink.datastream import StreamExecutionEnvironment, TimeCharacteristic
from pyflink.table import StreamTableEnvironment, DataTypes, EnvironmentSettings
from pyflink.table.udf import udf
provinces = ("beijing", "shanghai", "hangzhou", "shenzhen", "jiangxi", "chongqing", "xi
zang")
@udf(input_types=[DataTypes.INT()], result_type=DataTypes.STRING())
def province_id_to_name(id):
    return provinces[id]
#请根据创建的Kafka集群，输入以下信息。
def log_processing():
    kafka_servers = "xx.xx.xx.xx:9092,xx.xx.xx.xx:9092,xx.xx.xx.xx:9092"
    kafka_zookeeper_servers = "xx.xx.xx.xx:2181,xx.xx.xx.xx:2181,xx.xx.xx.xx:2181"
    source_topic = "payment_msg"
```

```
    sink_topic = "results"
    kafka_consumer_group_id = "test_3"
    env = StreamExecutionEnvironment.get_execution_environment()
    env.set_stream_time_characteristic(TimeCharacteristic.EventTime)
    env_settings = EnvironmentSettings.Builder().use_blink_planner().build()
    t_env = StreamTableEnvironment.create(stream_execution_environment=env, environment
_settings=env_settings)
    t_env.get_config().get_configuration().set_boolean("python.fn-execution.memory.mana
ged", True)
    source_ddl = f"""
            CREATE TABLE payment_msg(
                createTime VARCHAR,
                rt as TO_TIMESTAMP(createTime),
                orderId BIGINT,
                payAmount DOUBLE,
                payPlatform INT,
                provinceId INT,
                WATERMARK FOR rt as rt - INTERVAL '2' SECOND
            ) WITH (
              'connector.type' = 'kafka',
              'connector.version' = 'universal',
              'connector.topic' = '{source_topic}',
              'connector.properties.bootstrap.servers' = '{kafka_servers}',
              'connector.properties.zookeeper.connect' = '{kafka_zookeeper_servers}',
              'connector.properties.group.id' = '{kafka_consumer_group_id}',
              'connector.startup-mode' = 'latest-offset',
              'format.type' = 'json'
            )
            """
    es_sink_ddl = f"""
            CREATE TABLE es_sink (
            province VARCHAR,
            pay_amount DOUBLE,
            rowtime TIMESTAMP(3)
            ) with (
              'connector.type' = 'kafka',
              'connector.version' = 'universal',
              'connector.topic' = '{sink_topic}',
              'connector.properties.bootstrap.servers' = '{kafka_servers}',
              'connector.properties.zookeeper.connect' = '{kafka_zookeeper_servers}',
              'connector.properties.group.id' = '{kafka_consumer_group_id}',
              'connector.startup-mode' = 'latest-offset',
              'format.type' = 'json'
            )
    """
    t_env.sql_update(source_ddl)
    t_env.sql_update(es_sink_ddl)
    t_env.register_function('province_id_to_name', province_id_to_name)
    query = """
    select province_id_to_name(provinceId) as province, sum(payAmount) as pay_amount, t
umble_start(rt, interval '5' second) as rowtime
    from payment_msg
    group by tumble(rt, interval '5' second), provinceId
    """
```

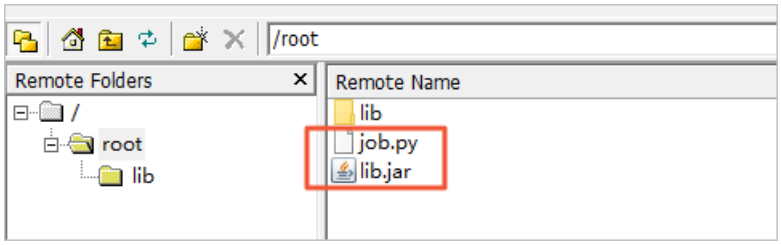
```
    t_env.sql_query(query).insert_into("es_sink")
    t_env.execute("payment_demo")
if __name__ == '__main__':
    log_processing()
EOF
rm -rf lib
mkdir lib
cd lib
wget https://maven.aliyun.com/nexus/content/groups/public/org/apache/flink/flink-sql-co
nnector-kafka_2.11/1.10.1/flink-sql-connector-kafka_2.11-1.10.1.jar
wget https://maven.aliyun.com/nexus/content/groups/public/org/apache/flink/flink-json/1
.10.1/flink-json-1.10.1-sql-jar.jar
cd ../
zip -r lib.jar lib/*
```

请您根据集群的实际情况，修改*job.py*中如下参数。

参数	描述
`kafka_servers`	Kafka集群中Kafka Broker组件的地址列表。IP地址为Kafka集群的内网IP地址，端口号默认为9092。IP地址如Kafka集群组件所示。
`kafka_zookeeper_servers`	Kafka集群中Zookeeper服务的地址列表。IP地址为Kafka集群的内网IP地址，端口号默认为2181。IP地址如Kafka集群组件所示。
`source_topic`	源表的Kafka Topic，本文示例为payment_msg。
`sink_topic`	结果表的Kafka Topic，本文示例为results。

Kafka集群组件

本地以Windows为例，生成*lib.jar*和*job.py*示例如下。



3. 使用文件传输工具链接Hadoop集群的Master节点，下载生成的*lib.jar*和*job.py*至本地。
4. 上传生成的*lib.jar*和*job.py*至OSS管理控制台。
   i. 登录 OSS管理控制台。

ii. 创建存储空间和上传文件，详情请参见创建存储空间和上传文件。

本示例的上传路径分别为*oss://emr-logs2/test/lib.jar*和*oss://emr-logs2/test/job.py*。

5. 创建PyFlink作业。

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**数据开发**页签。

iv. 在**项目列表**页面，单击待编辑项目所在行的**作业编辑**。

v. 在**作业编辑**区域，在需要操作的文件夹上，右键选择**新建作业**。

vi. 输入作业名称、作业描述，在**作业类型**下拉列表中选择**Flink**。

vii. 配置作业内容，示例如下。

```
run -m yarn-cluster -py ossref://emr-logs2/test/job.py -j ossref://emr-logs2/test/l
ib.jar
```

6. 运行PyFlink作业。

i. 单击右上角的**保存**。

ii. 单击右上角的**运行**。

iii. 在**运行作业**对话框中，从**执行集群**列表，选择新建的Hadoop集群。

iv. 单击**确定**。

步骤五：查看作业信息

1. 通过Yarn UI查看Flink作业的信息。

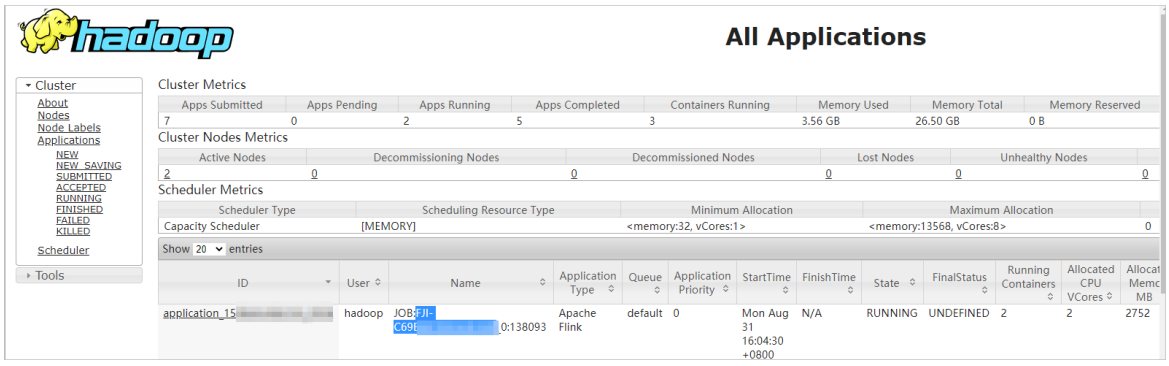
访问Yarn UI支持如下两种方式：

- SSH隧道方式：详情请参见通过SSH隧道方式访问开源组件Web UI。
- Knox方式：详情请参见访问链接与端口。

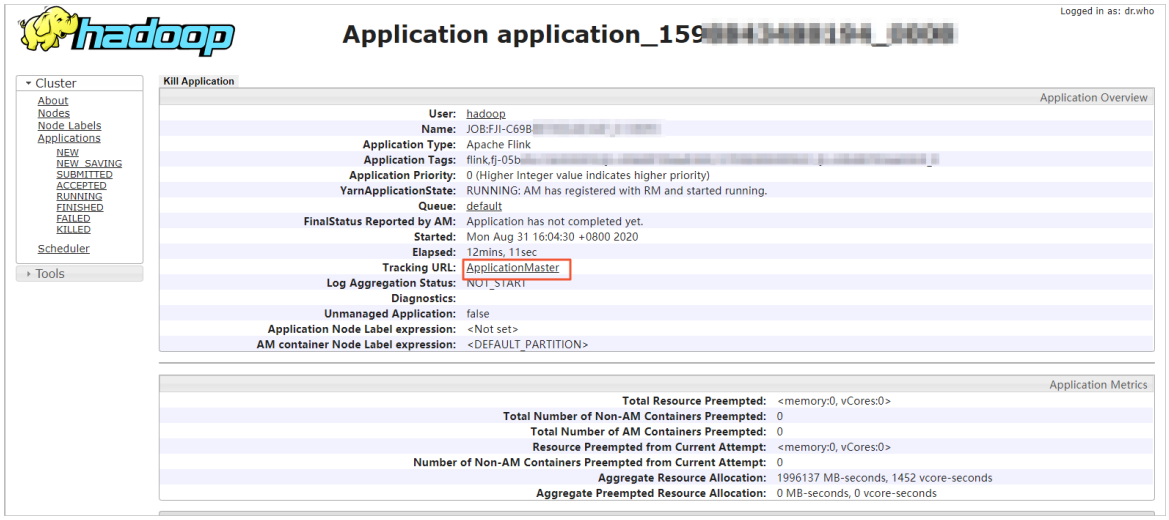
本示例通过Knox方式查看Flink作业的信息。

2. 在Hadoop控制台，单击作业的**ID**。

您可以查看作业运行详情。



详细信息如下。



3. （可选）单击Tracking URL后面的链接，进入Apache Flink Dashboard页面。

   您可以查看详情的作业信息。

步骤六：查看输出数据

1. 登录Kafka集群的Master节点。详情请参见登录集群。
2. 执行如下命令，查看results的数据。

```
kafka-console-consumer.sh --bootstrap-server emr-header-1:9092 --topic results
```

返回信息如下。

```
[root@emr-header-1 ~]# kafka-console-consumer.sh --bootstrap-server emr-header-1:9092 --topic results
{"province":"chongqing","pay_amount":185572.3127896842,"rowtime":"2020-08-31T15:07:20Z"}
{"province":"hangzhou","pay_amount":20400.98404899038,"rowtime":"2020-08-31T15:07:20Z"}
{"province":"xizang","pay_amount":35183.866669616575,"rowtime":"2020-08-31T15:07:20Z"}
{"province":"beijing","pay_amount":94473.2055565579,"rowtime":"2020-08-31T15:07:20Z"}
{"province":"jiangxi","pay_amount":108376.45927606692,"rowtime":"2020-08-31T15:07:20Z"}
{"province":"xizang","pay_amount":14806.092503785805,"rowtime":"2020-08-31T15:07:25Z"}
{"province":"hangzhou","pay_amount":298879.5889229643,"rowtime":"2020-08-31T15:07:25Z"}
{"province":"chongqing","pay_amount":65469.68964449772,"rowtime":"2020-08-31T15:07:25Z"}
{"province":"jiangxi","pay_amount":46932.41704081994,"rowtime":"2020-08-31T15:07:25Z"}
{"province":"shenzhen","pay_amount":51900.6404747701,"rowtime":"2020-08-31T15:07:25Z"}
{"province":"beijing","pay_amount":116525.48560284806,"rowtime":"2020-08-31T15:07:25Z"}
{"province":"shanghai","pay_amount":165567.69040983776,"rowtime":"2020-08-31T15:07:30Z"}
{"province":"beijing","pay_amount":60992.148080331324,"rowtime":"2020-08-31T15:07:30Z"}
{"province":"hangzhou","pay_amount":162761.93676065805,"rowtime":"2020-08-31T15:07:30Z"}
{"province":"xizang","pay_amount":115971.41127212322,"rowtime":"2020-08-31T15:07:30Z"}
{"province":"chongqing","pay_amount":38370.04311796407,"rowtime":"2020-08-31T15:07:30Z"}
```

查看完信息后，您可以在**数据开发**的Flink作业页面，单击右上角的**停止**，停掉正在运行的Flink作业。