阿里云
E-MapReduce

开源组件介绍

文档版本：20190916

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
	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。	 **禁止:** 重置操作将丢失用户配置数据。
	该类警示信息可能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，恢复业务所需时间约10分钟。
	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。	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。	单击 **确定**。
`courier` 字体	命令。	执行 `cd /d C:/windows` 命令，进入Windows系统文件夹。
*## *	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。	`swich` *`{stand | slave}`*

目录

1 开源组件介绍

2 Hue 使用说明

目前 E-MapReduce 中支持了 Hue，可以通过 Apache Knox 访问 Hue。

准备工作

在集群#unique_5中设置安全组规则，打开 8888 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。

访问 Hue

在 E-MapReduce 控制台中提供了快速访问集群中 Hue 服务的链接入口，您可通过以下方式访问 Hue 服务：

1. 在集群列表页面，单击集群 ID 右侧的管理。
2. 在页面左侧导航栏中单击访问链接与端口。
3. 单击 Hue 服务对应的访问链接。

查看初始密码

Hue 服务默认第一次运行时，如果未设置管理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 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. 在集群列表页面，单击集群 ID 右侧的管理。
2. 单击左侧导航栏中的集群服务，在服务列表中，选择 Hue。
3. 单击配置页签，找到 `admin_pwd` 参数，该参数对应的就是随机密码。

> 注意:
> 注意: admin_pwd 仅为 admin 账号的初始密码，在 EMR 控制台上改变该密码不会同步到 HUE 中。如果需要改变 admin 账号的登入密码，请使用该初始密码登入HUE，然后在 HUE 的用户管理模块中进行修改。

创建 Hue 用户账号

如果用户忘记了自己的 Hue 账号所对应的密码，可以通过以下方式重新创建一个账号：

1. 在集群列表页面，单击集群 ID 右侧的管理。
2. 在页面左侧导航栏中单击集群基础信息。
3. 在主实例组部分获取 Master 节点的公网 IP。

4. 通过 #unique_7的方式登录 Master 节点。
5. 执行以下命令，创建新账号。

```
/opt/apps/hue/build/env/bin/hue createsuperuser
```

6. 输入新用户名、电子邮件，然后输入密码，再次输入密码后按回车键。

   如果提示 Superuser created successfully，则说明新账号创建成功，稍后用新账号登录 Hue 即可。

添加/修改配置

您可以通过自定义配置添加相关配置：

1. 在 E-MapReduce 管理控制台，单击集群 ID 右侧的管理。
2. 在服务列表中，选择Hue，然后单击配置页签。
3. 单击右上角的自定义配置按钮，添加配置的 key/value 值，其中 key 需要遵循下面规范：

```
$section_path.$real_key
```

说明：

- $real_key 即为需要添加的实际的key，如 hive_server_host。
- $real_key 前面的 $section_path 可以通过 *hue.ini* 文件进行查看，例如：

  hive_server_host，通过hue.ini文件可以看出它属于[beeswax]这个section下，则 $section_path 为 beeswax。
- 综上，添加的 key 为 beeswax.hive_server_host。
- 同理，如需修改 *hue.ini* 文件中的多级 section [desktop] -> [[ldap]] -> [[[ldap_servers]]] -> [[[[users]]]] ->user_name_attr 的值，则需要配置的 key 为desktop.ldap.ldap_servers.users.user_name_attr。

3 Oozie 使用说明

本文将介绍如何在 E-MapReduce 上使用 Oozie。

> 说明:
> 阿里云 E-MapReduce 在 2.0.0 及之后的版本中提供了对 Oozie 的支持，如果需要在集群中使用 Oozie，请确认集群的版本不低于 2.0.0。

准备工作

在集群建立出来之后，需要打通 SSH 隧道，详细步骤请参见 #unique_7。

这里以 MAC 环境为例，使用 Chrome 浏览器实现端口转发（假设集群 Master 节点公网 IP 为 xx.xx.xx.xx）：

1. 登录到 Master 节点。

```
ssh root@xx.xx.xx.xx
```

2. 输入密码。
3. 查看本机的 id_rsa.pub 内容（注意在本机执行，不要在远程的 master 节点上执行）。

```
cat ~/.ssh/id_rsa.pub
```

4. 将本机的 id_rsa.pub 内容写入到远程 Master 节点的 *~/.ssh/authorized_keys* 中（在远端 Master 节点上执行）。

```
mkdir ~/.ssh/
vim ~/.ssh/authorized_keys
```

5. 然后将步骤 2 中看到的内容粘贴进来，现在应该可以直接使用 `ssh root@xx.xx.xx.xx` 免密登录 master 节点了。
6. 在本机执行以下命令进行端口转发。

```
ssh -i ~/.ssh/id_rsa -ND 8157 root@xx.xx.xx.xx
```

7. 启动 Chrome（在本机新开 terminal 执行）。

```
/Applications/Google\ Chrome.app/Contents/MacOS/Google\ Chrome --
proxy-server="socks5://localhost:8157" --host-resolver-rules="MAP *
0.0.0.0 , EXCLUDE localhost" --user-data-dir=/tmp
```

访问 Oozie UI 页面

在进行端口转发的 Chrome 浏览器中访问：xx.xx.xx.xx:11000/oozie，localhost:11000/oozie 或者内网 ip:11000/oozie。

提交 Workflow 作业

运行 Oozie 需要先安装 Oozie 的 shareLib。

在 E-MapReduce 集群中，默认给 Oozie 用户安装了 sharelib，即如果使用 Oozie 用户来提交 workflow 作业，则不需要再进行 sharelib 的安装。

由于开启 HA 的集群和没有开启 HA 的集群，访问 NameNode 和 ResourceManager 的方式不同，在提交 oozie workflow job 的时候，job.properties 文件中需要指定不同的 NameNode 和 JobTracker （ResourceManager）。具体如下：

- 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
```

下面操作示例中，已经针对是否是 HA 集群配置好了，即样例代码不需要任何修改即可以直接运行。关于 workflow 文件的具体格式，请参见 Oozie 官方文档。

- 在非 HA 集群上提交 workflow 作业

1. 登录集群的主 Master 节点。

```
ssh root@master公网Ip
```

2. 下载示例代码。

```
[root@emr-header-1 ~]# su oozie
[oozie@emr-header-1 root]$ cd /tmp
[oozie@emr-header-1 tmp]$ wget http://emr-sample-projects.oss-cn-
hangzhou.aliyuncs.com/oozie-examples/oozie-examples.zip
[oozie@emr-header-1 tmp]$ unzip oozie-examples.zip
```

3. 将 Oozie workflow 代码同步到 hdfs 上。

```
[oozie@emr-header-1 tmp]$ hadoop fs -copyFromLocal examples/ /user
/oozie/examples
```

4. 提交 Oozie workflow 样例作业。

```
[oozie@emr-header-1 tmp]$ $OOZIE_HOME/bin/oozie job -config
examples/apps/map-reduce/job.properties -run
```

执行成功之后，会返回一个 jobId，类似：

```
job: 0000000-160627195651086-oozie-oozi-W
```

5. 访问 Oozie UI 页面，可以看到刚刚提交的 Oozie workflow job。

- 在 HA 集群上提交 workflow 作业

  1. 登录集群的主 Master 节点。

     ```
     ssh root@主master公网Ip
     ```

     可以通过是否能访问 Oozie UI 来判断哪个 Master 节点是当前的主 Master 节点， Oozie server 服务默认是启动在主 Master 节点 `xx.xx.xx.xx:11000/oozie`。

  2. 下载 HA 集群的示例代码。

     ```
     [root@emr-header-1 ~]# su oozie
     [oozie@emr-header-1 root]$ cd /tmp
     [oozie@emr-header-1 tmp]$ wget http://emr-sample-projects.oss-cn-
     hangzhou.aliyuncs.com/oozie-examples/oozie-examples-ha.zip
     [oozie@emr-header-1 tmp]$ unzip oozie-examples-ha.zip
     ```

  3. 将 Oozie workflow 代码同步到 HDFS 上。

     ```
     [oozie@emr-header-1 tmp]$ hadoop fs -copyFromLocal examples/ /user
     /oozie/examples
     ```

  4. 提交 Oozie workflow 样例作业。

     ```
     [oozie@emr-header-1 tmp]$ $OOZIE_HOME/bin/oozie job -config
     examples/apps/map-reduce/job.properties -run
     ```

     执行成功之后，会返回一个 jobId，类似：

     ```
     job: 0000000-160627195651086-oozie-oozi-W
     ```

  5. 访问 Oozie UI 页面，可以看到刚刚提交的 Oozie workflow job。

4 Zeppelin 使用说明

阿里云 E-MapReduce 可以通过 Apache Knox 访问 Zeppelin。

准备工作

1. 在集群#unique_5中设置安全组规则，打开 8080 端口。
2. 在 Knox 中，添加访问用户名和密码，详细参见 #unique_10，设置 Knox 用户。用户名和密码仅用于登录 Knox 的各项服务，与阿里云 RAM 的用户名无关。

> **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。

访问 Zeppelin

访问链接查看方式如下:

1. 在 E-MapReduce 集群管理页面单击集群 ID 右侧的管理。
2. 在页面左侧导航栏单击访问链接与端口。

5 ZooKeeper 使用说明

目前 E-MapReduce 集群中默认启动了 ZooKeeper 服务。

说明:

目前无论集群内有多少台机器，ZooKeeper 只会有 3 个节点。目前还不支持更多的节点。

创建集群

E-MapReduce 创建集群的软件配置页面，可选择ZooKeeper服务 。

节点信息

集群创建成功，状态空闲后，在集群与服务管理页面选择ZooKeeper，单击部署拓扑可以查看到 ZooKeeper 的节点信息，E-MapReduce 会启动 3 个 ZooKeeper 节点。IP 一栏标有 ZooKeeper 节点对应的内网 IP (端口默认为 2181)，即可访问 ZooKeeper 服务。

6 Kafka使用说明

6.1 Kafka 快速入门

从 E-MapReduce 3.4.0 版本开始将支持 Kafka 服务。

创建Kafka集群

在 E-MapReduce 控制台创建集群时，选择集群类型为 Kafka，则会创建一个默认只包含 Kafka 组件的集群，除了基础组件外包括 Zookeeper，Kafka 和 KafkaManager 三个组件。每个节点将只部署一个 Kafka broker。我们建议您的Kafka 集群是一个专用集群，不要和 Hadoop 相关服务混部在一起。

本地盘Kafka集群

为了更好地降低单位成本以及应对更大的存储需求，E-MapReduce 将在 EMR-3.5.1 版本开始支持基于本地盘（D1类簇机型，详情请参见#unique_14介绍文档）的 Kafka 集群。相比较云盘，本地盘 Kafka 集群有如下特点：

- 实例配备大容量、高吞吐 SATA HDD 本地盘，单磁盘 190 MB/s 顺序读写性能，单实例最大 5 GB/s 存储吞吐能力。
- 本地存储价格比 SSD 云盘降低 97%。
- 更高网络性能，最大 17 Gbit/s实例间网络带宽，满足业务高峰期实例间数据交互需求。

但本地盘机型也有特殊之处：

操作	本地磁盘数据状态	说明
操作系统重启/控制台重启/强制重启	保留	本地磁盘存储卷保留，数据保留
操作系统关机/控制台停止/强制停止	保留	本地磁盘存储卷保留，数据保留
控制台上释放（实例）	擦除	本地磁盘存储卷擦除，数据不保留

注意：

- 当宿主机宕机或者磁盘损坏时，磁盘中的数据将会丢失。
- 请勿在本地磁盘上存储需要长期保存的业务数据，并及时做好数据备份和采用高可用架构。如需长期保存，建议将数据存储在云盘上。

为了能够在本地盘上部署 Kafka 服务，E-MapReduce 默认以下要求：

1. `default.replication.factor = 3`，即要求 topic 的分区副本数至少为3。如果设置更小副本数，则会增加数据丢失风险。
2. `min.insync.replicas = 2`，即要求当 producer 设定 acks 为 all(-1)时，每次至少写入两个副本才算写入成功。

当出现本地盘损坏时，E-MapReduce 会进行：

1. 将坏盘从 Broker 的配置中剔除，重启 Broker，在其他正常可用的本地盘上恢复坏盘丢失的数据。根据坏盘上已经写入的数据量不等，恢复的总时间也不等。
2. 当机器磁盘损坏数目过多（超过 20%）是，E-MapReduce 将主动进行机器迁移，恢复异常的磁盘。
3. 如果当前机器上可用剩余磁盘空间不足以恢复坏盘丢失数据时，Broker 将异常 Down 掉。这种情况，您可以选择清理一些数据，腾出磁盘空间并重启 Broker 服务；也可以联系 E-MapReduce 进行机器迁移，恢复异常的磁盘。

参数说明

您可以在 E-MapReduce 的集群配置管理中查看 Kafka 的软件配置，当前主要有：

配置项	说明
zookeeper.connect	Kafka 集群的 Zookeeper 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，默认为机器 CPU 核数目的 2 倍
num.network.threads	Kafka broker 的网络线程数，默认为机器的 CPU 核数目

6.2 Kafka 跨集群访问

通常，我们会单独部署一个Kafka集群来提供服务，所以经常需要跨集群访问Kafka服务。

Kafka 跨集群访问说明

跨集群访问Kafka场景分为两种：

- 阿里云内网环境中访问E-MapReduce Kafka集群。
- 公网环境访问E-MapReduce Kafka集群。

> 说明:
> 经典网络的E-MapReduce Kafka集群暂不支持公网访问。

针对不同E-MapReduce主版本，我们提供了不同的解决方案。

EMR-3.11.x及之后版本

- 阿里云内网中访问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前请保证网络是互通的，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，可以参考以下步骤完成：

  1. 使 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，以下操作步骤使用此方式。
     - 部署高速通道打通内网和公网网络，请参见高速通道文档。

  2. 在 E-MapReduce 集群列表页单击对应集群操作栏中的详情进入集群基础信息页面。
  3. 单击右上角网络管理，在下拉框中选择挂载公网。
  4. 配置 Kafka 集群安全组规则来限制公网可访问 Kafka 集群的 IP 等，目的是提高 Kafka 集群暴露在公网中的安全性。您可以在 E-MapReduce 控制台查看到集群所属的安全组，根据安全组 ID 去查找并配置安全组规则。文档地址
  5. 在 集群基础信息页单击页面右上角的同步主机信息，在下拉框中选择同步主机信息。
  6. 在集群与服务管理页面服务列表中依次单击 Kafka > 配置，在服务配置中找到 `kafka.public-access.enable`参数，并修改为 true。
  7. 重启 Kafka 服务。
  8. 公网环境使用 Kafka 集群节点的 EIP 访问 9093 端口。

EMR-3.11.x 以前版本

- 阿里云内网中访问 Kafka

  我们需要在主机上配置 Kafka 集群节点的 host 信息。注意，这里我们需要在 client 端的主机上配置 Kafka 集群节点的长域名，否则会出现访问不到 Kafka 服务的问题。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
```

- 公网环境访问 Kafka

  Kafka 集群的 Core 节点默认无法通过公网访问，所以如果您需要在公网环境访问 Kafka 集群，可以参考以下步骤完成：

  1. 使 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，以下步骤使用此方式。
     - 部署高速通道打通内网和公网网络，参考高速通道文档。

  2. 在VPC 控制台申请 EIP，根据您 Kafka 集群 Core 节点个数购买相应的 EIP。
  3. 配置 Kafka 集群安全组规则来限制公网可访问 Kafka 集群的 IP 等，目的是提高 Kafka 集群暴露在公网中的安全性。您可以在 EMR 控制台查看到集群所属的安全组，根据安全组 ID 去查找并配置安全组规则。文档地址
  4. 修改 Kafka 集群的软件配置 `listeners.address.principal` 为 `HOST`，并重启 Kafka 集群。
  5. 配置本地客户端主机的 *hosts* 文件。

6.3 Kafka Ranger 使用说明

前面简介中介绍了 E-MapReduce 中创建启动 Ranger 服务的集群，以及一些准备工作，本节介绍 Kafka 集成 Ranger 的一些步骤流程。

Kafka 集成 Ranger

从 E-MapReduce-3.12.0 版本开始，E-MapReduce Kafka 支持用 Ranger 进行权限配置，步骤如下：

- Enable Kakfa Plugin

  1. 在集群与服务管理页面的 Ranger 服务下，单击右侧的操作下拉菜单，选择 Enable Kafka PLUGIN。

  2. 单击右上角的查看操作历史查看任务进度，等待任务完成100%。

- 重启 Kafka broker

  上述任务完成后，需要重启 broker 才能生效。

  1. 在集群与服务管理页面的服务列表中单击 Kafka 进入 Kafka 服务配置页面。
  2. 点击右上角 操作 中的 RESTART Kafka Broker。
  3. 点击右上角查看操作历史查看任务进度，等待重启任务完成。

- Ranger UI 页面添加 Kafka Service

  参见 Ranger 简介进入 Ranger UI 页面。

  在 Ranger 的 UI 页面添加 Kafka Service：

  配置 Kafka Service：

权限配置示例

上面一节中已经将 Ranger 集成到 Kafka，现在可以进行相关的权限设置。

> 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：

按照上述步骤设置添加一个 Policy 后，就实现了对 `test` 的授权，然后用户 `test` 就可以对 `test` 的topic进行写入操作。

> 说明:
> Policy 添加后需要 1 分钟左右才会生效。

6.4 Kafka SSL使用说明

E-MapReduce Kafka 从 EMR-3.12.0 版本开始，开始支持 SSL 功能。

创建集群

具体的创建集群操作，请参见#unique_22。

开启 SSL 服务

Kafka 集群默认没有开启 SSL 功能，您可以在 Kafka 服务的配置页面开启 SSL。

如图，修改配置项 `kafka.ssl.enable` 为 `true`，部署配置并重启组件。

客户端访问 Kafka

客户端通过 SSL 访问 Kafka 时需要设置 `security.protocol`、`truststore` 和 `keystore` 的相关配置。以非安全集群为例，如果是在 Kafka 集群运行作业，可以配置如下：

```
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}
```

如果是在 Kafka 集群以外的环境运行作业，可将 Kafka 集群中的 truststore 和 keystore 文件（位于集群任意一个节点的 */etc/ecm/kafka-conf/*目录中）拷贝至运行环境作相应配置。

以 Kafka 自带的 producer 和 consumer 程序，在 Kafka 集群运行为例：

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}
```

2. 创建 topic。

```
kafka-topics.sh --zookeeper emr-header-1:2181/kafka-1.0.1 --
replication-factor 2 --
```

```
partitions 100 --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
```

6.5 Kafka Manager使用说明

从EMR-3.4.0版本开始，E-MapReduce支持通过Kafka Manager服务对Kafka集群进行管理。

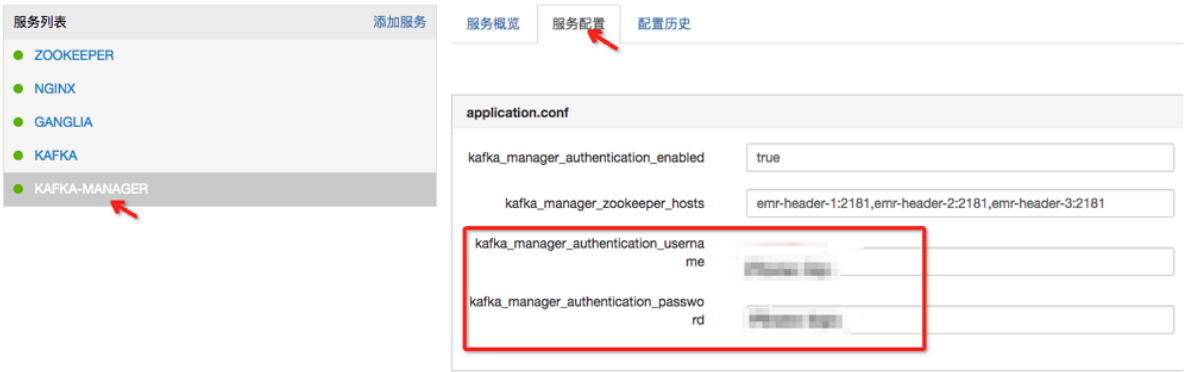
操作步骤

> **注意:**
> 创建 Kafka集群时将默认安装Kafka Manager软件服务，并开启Kafka Manager的认证功能。

1. 使用SSH隧道方式访问 Web 界面，请参见 #unique_7。

> **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界面时，请做好IP白名单保护，避免数据泄漏。

2. 输入用户名密码，请参考Kafka Manager的配置信息。



3. 添加一个创建好的Kafka集群，需要注意配置正确Kafka集群的Zookeeper地址，可以参考Kakfa的配置信息。选择对应的Kafka版本，另外建议打开JMX功能。

4. 创建好之后即可使用一些常见的Kakfa功能。

6.6 Kafka 常见问题

本文介绍 Kafka 常见问题的一些问题以及解决方法。

- `Error while executing topic command : Replication factor: 1 larger than available brokers: 0.`

  常见原因：

  - Kafka 服务异常，集群 Broker 进程都退出了，需要结合日志排查问题。
  - Kafka 服务的 Zookeeper 地址使用错误，请注意查看并使用集群配置管理中 Kafka 组件的 Zookeeper 连接地址。

- `java.net.BindException: Address already in use (Bind failed)`

  当您使用 Kafka 命令行工具时，有时会碰到 `java.net.BindException: Address already in use (Bind failed)` 异常，这一般是由 JMX 端口被占用导致，您可以在命令行前手动指定一个 JMX 端口即可。例如：

  ```
  JMX_PORT=10101 kafka-topics --zookeeper emr-header-1:2181/kafka-1.0.0 --list
  ```

7 Druid 使用说明

7.1 Druid 简介

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进行快速的、交互式的查询和分析。Apache Druid 由 Metamarkets 公司（一家为在线媒体或广告公司提供数据分析服务的公司）开发，在2019年春季被捐献给 Apache 软件基金会。

基本特点

Apache Druid 具有以下特点：

- 亚秒级 OLAP 查询，包括多维过滤、Ad-hoc 的属性分组、快速聚合数据等等。
- 实时的数据消费，真正做到数据摄入实时、查询结果实时。
- 高效的多租户能力，最高可以做到几千用户同时在线查询。
- 扩展性强，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。
- 极高的高可用保障，支持滚动升级。

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

这些场景的特点都是拥有大量的数据，且对数据查询的时延要求非常高。在实时指标监控中，系统问题需要在出现的一刻被检测到并被及时给出报警。在推荐模型中，用户行为数据需要实时采集，并及时反馈到推荐系统中。用户几次点击之后系统就能够识别其搜索意图，并在之后的搜索中推荐更合理的结果。

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、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 阶段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 节点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，比如配置信息、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、易用的产品接口等等，真正做到了即买即用和 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、缩容（缩容针对 task 节点）。
- 丰富的监控指标和告警规则。
- 坏节点迁移。
- 支持高安全。
- 支持 HA。

7.2 快速入门

E-MapReduce 从 EMR-3.11.0 版本开始支持 E-MapReduce Druid 作为 E-MapReduce 的一个集群类型。

将 E-MapReduce Druid 作为一种单独的集群类型（而不是在 Hadoop 集群中增加 Druid 组件）主要基于几个方面的考虑：

- E-MapReduce Druid 可以完全脱离 Hadoop 使用。
- 大数据量下 E-MapReduce Druid 对内存要求比较高，尤其是 Broker 节点和 Historical 节点。E-MapReduce Druid 本身不受 YARN 管控，在多服务运行时容易发生资源争抢。
- Hadoop 作为基础设施，其规模可以比较大，而 E-MapReduce Druid 集群可以比较小，两者配合起来工作灵活性更高。

创建 Druid 集群

在创建集群时选择 Druid 集群类型即可。您在创建 E-MapReduce Druid 集群时可以勾选 HDFS 和 YARN 服务，E-MapReduce Druid 集群自带的 HDFS 和 YARN 仅供测试使用，原因如本文档开头所述。对于生产环境，我们强烈建议您采用专门的 Hadoop 集群。

配置集群

- 配置使用 HDFS 作为 E-MapReduce Druid 的 deep storage

  对于独立的 E-MapReduce Druid 集群，您可能需要将您的索引数据存放在另外一个 Hadoop 集群的 HDFS 之上。为此，您需要首先设置一下两个集群的连通性（见下文与 Hadoop 集群交互一节），再在 E-MapReduce Druid 的配置页面配置如下两个选项并重启服务即可（配置项位于配置页面的 common.runtime）。

  - druid.storage.type: hdfs
  - druid.storage.storageDirectory:（请注意这里 HDFS 目录最好写完整目录，比如 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，下同。

- 配置使用 OSS 作为 E-MapReduce Druid 的 deep storage

  E-MapReduce Druid 支持以 OSS 作为 deep storage，借助于 E-MapReduce 的免 AccessKey 能力，E-MapReduce Druid 不用做 AccessKey 配置即可访问 OSS。由于 OSS

的访问能力是借助于 HDFS 的 OSS 功能实现的，因此在配置时，`druid.storage.type`**需要仍然配置为 HDFS。**

- `druid.storage.type: hdfs`
- `druid.storage.storageDirectory:`（**如** *oss://emr-druid-cn-hangzhou/segments*）

由于 OSS 访问借助了 HDFS，因此您需要选择以下两种方案之一：

- **建集群的时候选择安装 HDFS，系统自动配好（安装好 HDFS 您可以不使用它，关闭它，或者仅作为测试用途）。**
- **在 E-MapReduce Druid 的配置目录**`/etc/ecm/druid-conf/druid/_common/`**下新建** *hdfs-site.xml*，**内容如下，然后将该文件拷贝至所有节点的相同目录下。**

```
<?xml version="1.0"?>
  <configuration>
    <property>
      <name>fs.oss.impl</name>
      <value>com.aliyun.fs.oss.nat.NativeOssFileSystem</value>
    </property>
    <property>
      <name>fs.oss.buffer.dirs</name>
      <value>file:///mnt/disk1/data,...</value>
    </property>
    <property>
      <name>fs.oss.impl.disable.cache</name>
      <value>true</value>
    </property>
  </configuration>
```

其中`fs.oss.buffer.dirs`**可以设置多个路径。**

- 配置使用 RDS 作为 E-MapReduce Druid 的元数据存储

  默认情况下 E-MapReduce Druid 利用 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。

  下面以 RDS MySQL 版作为例演示配置。在具体配置之前，请先确保：

  - RDS MySQL 实例已经被创建。
  - 为 E-MapReduce Druid 访问 RDS MySQL 创建了单独的账户（不推荐使用 root），假设账户名为 druid，密码为 druidpw。
  - 为 E-MapReduce Druid 元数据创建单独的 MySQL 数据库，假设数据库名为 druiddb。
  - 确保账户 druid 有权限访问 druiddb。

  在 E-MapReduce 管理控制台，进入 E-MapReduce Druid 集群，单击 Druid 组件，选择配置选项卡，找到 *common.runtime* 配置文件。单击自定义配置，添加如下三个配置项：

  - druid.metadata.storage.connector.connectURI，值为 jdbc:mysql://rm-xxxxx.mysql.rds.aliyuncs.com:3306/druiddb。
  - druid.metadata.storage.connector.user，值为druid。
  - druid.metadata.storage.connector.password，值为druidpw。

  依次点击右上角的保存 > > 部署配置文件到主机 > 重启所有组件，配置即可生效。

  登录 RDS 控制台查看 druiddb 创建表的情况，如果正常，您将会看到一些 druid 自动创建的表。

- 组件内存配置

  E-MapReduce Druid 组件内存设置主要包括两方面：堆内存（通过 jvm.config 配置）和 direct 内存（通过 jvm.config 和 runtime.properteis 配置）。在创建集群时，E-MapReduce 会自动生成一套配置，不过在某些情况下您仍然可能需要自己调整内存配置。

  要调整组件内存配置，您可以通过 E-MapReduce 控制台进入到集群组件，在页面上进行操作。

  说明:

  对于 direct 内存，调整时请确保：

  ```
  -XX:MaxDirectMemorySize >= druid.processing.buffer.sizeBytes * (
  druid.processing.numMergeBuffers + druid.processing.numThreads + 1)
  ```

访问 Druid web 页面

E-MapReduce Druid 自带两个 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。

E-MapReduce 提供三种方式访问 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 overlord 或 Druid coordinator 链接，单击链接进入（推荐方式，EMR-3.20.0 及之后版本）。
- 通过 SSH 隧道方式建立 SSH 隧道，开启代理浏览器访问。
- 通过公网 IP+端口访问，如 *http://123.123.123.123:18090*（不推荐，请通过安全组设置合理控制公网访问集群权限）。

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 HDFS 和 YARN（自带 Hadoop 集群），那么系统将会自动为您配置好与 HDFS 和 YARN 的交互，您无需做额外操作。下面的介绍是配置独立 E-MapReduce Druid 集群与独立 Hadoop 集群之间交互，这里假设 E-MapReduce Druid 集群 cluster id 为 1234，Hadoop 集群 cluster id 为 5678。另外请仔细阅读并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。

  对于与非安全独立 Hadoop 集群交互，请按照如下操作进行：

  1. 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）。
  2. 将Hadoop集群*/etc/ecm/hadoop-conf*的 core-site.xml、hdfs-site.xml、yarn-site.xml、 mapred-site.xml 放在 E-MapReduce Druid 集群每个节点的 */etc/ecm/duird-conf/druid/_common* 目录下（如果创建集群时选了自带Hadoop，在该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）。
  3. 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：

     ```
     ...
     10.157.201.36   emr-as.cn-hangzhou.aliyuncs.com
     10.157.64.5     eas.cn-hangzhou.emr.aliyuncs.com
     192.168.142.255 emr-worker-1.cluster-1234 emr-worker-1 emr-header-
     2.cluster-1234 emr-header-2 iZbp1h9g7boqo9x23qbifiZ
     192.168.143.0   emr-worker-2.cluster-1234 emr-worker-2 emr-header-
     3.cluster-1234 emr-header-3 iZbp1eaa5819tkjx55yr9xZ
     192.168.142.254 emr-header-1.cluster-1234 emr-header-1 iZbp1e3zwu
     vnmakmsjer2uZ
     --以下为hadoop集群的hosts信息
     ```

```
192.168.143.6   emr-worker-1.cluster-5678 emr-worker-1 emr-header-
2.cluster-5678 emr-header-2 iZbp195rj7zvx8qar4f6b0Z
192.168.143.7   emr-worker-2.cluster-5678 emr-worker-2 emr-header-
3.cluster-5678 emr-header-3 iZbp15vy2rsxoegki4qhdpZ
192.168.143.5   emr-header-1.cluster-5678 emr-header-1 iZbp10tx4e
gw3wfnh5oii1Z
```

对于安全 Hadoop 集群，请按如下操作进行：

1. 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）。
2. 将 Hadoop 集群 */etc/ecm/hadoop-conf* 的 core-site.xml、hdfs-site.xml、yarn-site.xml、mapred-site.xml 放在 E-MapReduce Druid 集群每个节点的 */etc/ecm/duird-conf/druid/_common* 目录下（如果创建集群时选了自带 Hadoop，在该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），并修改 core-site.xml 中 `hadoop.security.authentication.use.has` 为 `false`（这是一条客户端配置，目的是让用户能够使用 AccessKey 认证，如果用 Kerberos 认证方式，需 disable 该配置）。
3. 将 Hadoop 集群的 hosts 写入到 E-MapReduce Druid 集群每个节点的 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。
4. 设置两个集群间的 Kerberos 跨域互信（详情参考#unique_29）。
5. 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（具体预发规则参考这里）创建 Kerberos 账户到本地账户的映射规则。推荐第一种做法，操作简便不易出错。

   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 也支持将一个 principal 映射至本地一个账户，即上文第二种做法。
6. 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，位于*${DRUID_HOME}/quickstart/tutorial*下面（*${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：

1. 将该压缩文件解压，并放置于 HDFS 的一个目录下（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集群上进行操作，需要做好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-2015-09-16-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 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。

2. 准备一个数据索引任务文件 *${DRUID_HOME}/quickstart/tutorial/wikiticker-index.json*，如下所示：

```
{
     "type" : "index_hadoop",
     "spec" : {
         "ioConfig" : {
             "type" : "hadoop",
             "inputSpec" : {
                 "type" : "static",
                 "paths" : "hdfs://emr-header-1.cluster-5678:9000/druid/wikiticker-2015-09-16-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:2.7.2"]
 }
```

说明:

- spec.ioConfig.type 设置为 hadoop。
- spec.ioConfig.inputSpec.paths 为输入文件路径。
- tuningConfig.type 为 hadoop。
- tuningConfig.jobProperties 设置了 mapreduce job 的 classloader。
- hadoopDependencyCoordinates 制定了 hadoop client 的版本。

3. 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/tutorial/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。

4. 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。

5. 根据 Druid 语法查询数据。

Druid 有自己的查询语法。您需要准备一个描述您如何查询的 json 格式的查询文件，如下所示为对 wikiticker 数据的一个 top N 查询（*${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/tutorial/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。如果一切正常，您将能看到具体地查询结果。

实时索引

对于数据从 Kafka 集群实时到 E-MapReduce Druid 集群进行索引，我们推荐使用 Kafka Indexing Service 扩展，提供了高可靠保证，支持 exactly-once 语义。详见 #unique_30 中“使用 Druid Kafka Indexing Service 实时消费Kafka数据”一节。

如果您的数据实时打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，我们提供了 SLS Indexing Service 扩展。使用 SLS Indexing Service 避免了您额外建立并维护 Kafka 集群的开销。SLS Indexing Service 的作用与 Kafka Indexing Service 相同，也提供高可靠保证和 Exactly-Once语义。在这里，您完全可以把 SLS 当成一个 Kafka 来使用。详见 SLS-Indexing-Service 一节。

对于其他方式，如使用 Flink，Storm，Spark Streaming 等等，我们推荐使用 Tranquility 客户端向 E-MapReduce Druid 集群推送数据。详见 #unique_32 一节。

Kafka Indexing Service 和 SLS Indexing Service 是类似的，都使用拉的方式从数据源拉取数据到 E-MapReduce Druid 集群，并提供高可靠保证和 exactly-once 语义；而 Tranquility 则使用推的方式，把数据推送至 E-MapReduce Druid 进行索引。Tranquility 并不提供 Exactly-once 语义，因此如果有这方面的要求，需要使用者自己解决。

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

  1. 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。
  2. 在 Overlord 页面观察作业执行情况，如果失败，查看页面上的 logs。
  3. 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。
  4. 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。
  5. 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那个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

  1. 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 是否合法。
  2. 查看失败 task 的 log。
  3. 如果不能从 task log 定位出失败原因，则需要从 Overlord log 入手，排查问题，其排查思路与批量索引一节中最后两步类似。

- 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 步骤 2～ 步骤 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（如 ioConfig 等）。

7.3 数据格式描述文件

本小节简要介绍一下索引数据的描述文件——Ingestion Spec 文件，更为详细的信息请参考 Apache Druid 官方文档。

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	描述所要消费数据的schema信息。dataSchema 是固定的，不随数据消费方式改变。	是
ioConfig	JSON 对象	描述所要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	调节数据消费时的一些参数。数据消费方式不同，可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”，“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。下面的表是 timestampSpec 和 dimensionsSpec 的描述：

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

键	格式	描述	是否必须
dimensions	JSON 数组	描述数据包含哪些维度。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，比如 “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：

```
```json
{
    "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，您需要参考 Apache Druid 官方文档学习如何使用它们。

- 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

键	格式	描述	是否必须
intervals	字符串	数据消费时间间隔	对于batch 是，对于 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。比如 Hadoop 对批量数据创建索引，可以在这里指定一些 MapReduce 参数。Tunning Config 的内容依赖于您的数据来源可能有不同的内容。详细的例子可参考本服务自带的示例文件或官方文档。

7.4 Kafka Indexing Service

本文将介绍在 E-MapReduce 中如何使用 Apache Druid Kafka Indexing Service 实时消费 Kafka 数据。

Kafka Indexing Service 是 Apache Druid 推出的利用 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 启动之后会在 Middlemanager 中启动一些 indexing task，这些 task 会连接到 Kafka 集群消费 topic 数据，并完成索引创建。您需要做的，就是准备一个数据消费格式文件，之后通过 REST API 手动启动 supervisor。

与 Kafka集群交互

请参考Druid 使用 Tranquility Kafka一节的介绍。

使用 Apache Druid Kafka Indexing Service 实时消费Kafka数据

1. 在 Kafka 集群上（或者 gateway 上）执行下述命令创建一个名为 “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,emr-header-3/kafka-1.0.0 --partitions 1 --replication-factor 1 --
topic metrics
```

   其中各个参数可根据需要进行调整。–zookeeper 参数中 */kafka-1.0.0* 部分为 path，其具体值您可以登录 EMR 控制台，在 Kafka 集群的 Kafka 服务配置页面查看 zookeeper.connect 配置项的值。如果是您自己搭建的 Kafka 集群，–zookeeper 参数可根据您的实际配置进行改变。

2. 定义数据源的数据格式描述文件，我们将其命名为 metrics-kafka.json，并置于当前目录下面（或者放置于其他您指定的目录）。

```
{
     "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. 执行下述命令添加 Kafka supervisor。

```
curl --negotiate -u:druid -b ~/cookies -c ~/cookies -XPOST -H '
Content-Type: application/json' -d @metrics-kafka.json http://emr-
header-1.cluster-1234:18090/druid/indexer/v1/supervisor
```

其中 `–negotiate`、`-u`、`-b`、`-c` 等选项是针对安全 E-MapReduce Druid 集群。

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-worker-1:9092, emr-worker-2:9092, emr-worker-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 的命令提示符下输入一些数据。

```
{"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}
```

其中时间戳可用如下 python 命令生成：

```
python -c 'import datetime; print(datetime.datetime.utcnow().
strftime("%Y-%m-%dT%H:%M:%SZ"))'
```

6. 准备一个查询文件，命名为 metrics-search.json。

```
{
     "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 master 上执行查询。

```
curl --negotiate -u:Druid -b ~/cookies -c ~/cookies -XPOST -H '
Content-Type: application/json' -d @metrics-search.json http://emr-
header-1.cluster-1234:8082/druid/v2/?pretty
```

其中`–negotiate`、`-u`、`-b`、`-c`等选项是针对安全 E-MapReduce Druid 集群。

8. 如果一切正常，您将看到类似如下的查询结果。

```
[ {
   "timestamp" : "2018-03-06T09:00:00.000Z",
   "result" : [ {
     "dimension" : "user",
     "value" : "bob",
     "count" : 2
   } ]
 } ]
```

7.5 SLS Indexing Service

SLS Indexing Service 是 E-MapReduce 推出的一个 Druid 插件，用于从 SLS 消费数据。

背景介绍

SLS Indexing Service 消费原理与 Kafka Indexing Service 类似，因此也支持 Kafka Indexing Service 一样的 Exactly-Once 语义。其综合了 SLS 与 Kafka Indexing Service 两个服务的优点：

- 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。
- 不用额外维护一个 Kafka 集群，省去了数据流的一个环节。
- 支持 Exactly-Once 语义。
- 消费作业高可靠保证，作业失败重试，集群重启/升级业务无感知等。

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（注意要用内网服务入口）
  - 可访问 SLS 服务的 AccessKeyId 和对应的 AccessKeySecret

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。参考 Kafka Indexing Service 小节的介绍，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>,
        "consumerProperties": {
            "endpoint": "cn-hangzhou-intranet.log.aliyuncs.com", (以
杭州为例，注意使用内网服务入口)
            "access-key-id": <your_access_key_id>,
            "access-key-secret": <your_access_key_secret>,
            "logtail.collection-mode": "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，那么该出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： 你要收集的日志的 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：账户的 AccessKeyID
- ioConfig.consumerProperties.access-key-secret： 账户的 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。具体请参考 SLS 文档。

4. 在 Druid 端进行相关查询

7.6 Tranquility

本文以 Kafka 为例，介绍在 E-MapReduce 中如何使用 Tranquility 从 Kafka 集群采集数据，并实时推送至 E-MapReduce Druid 集群。

Tranquility 是一个以 push 方式向 E-MapReduce Druid 实时发送数据的应用。它替用户解决了分区、多副本、服务发现、防止数据丢失等多个问题，简化了用户使用 E-MapReduce Druid 的难度。它支持多种数据来源，包括 Samza、Spark、Storm、Kafka、Flink 等等。

与 Kafka 集群交互

首先是 E-MapReduce Druid 集群与 Kafka 集群的交互。两个集群交互的配置方式大体和 Hadoop 集群类似，均需要设置连通性、hosts 等。对于非安全 Kafka 集群，请按照以下步骤操作：

1. 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）。
2. 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，如 emr-header-1.cluster-xxxxxxxx。

对于安全的 Kafka 集群，您需要执行下列操作（前两步与非安全 Kafka 集群相同）：

1. 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）。
2. 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，如 emr-header-1.cluster-xxxxxxxx。
3. 设置两个集群间的 Kerberos 跨域互信（详情参考#unique_29），且最好做双向互信。
4. 准备一个客户端安全配置文件：

```
KafkaClient {
      com.sun.security.auth.module.Krb5LoginModule required
      useKeyTab=true
      storeKey=true
      keyTab="/etc/ecm/druid-conf/druid.keytab"
      principal="druid@EMR.1234.COM";
  };
```

   之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*）。

5. 在 E-MapReduce Druid 配置页面的 overlord.jvm 里新增如下选项：

```
Djava.security.auth.login.config=/tmp/kafka/kafka_client_jaas.conf
```

   。

6. 在 E-MapReduce Druid 配置页面的 middleManager.runtime 里配置`druid.indexer.runner.javaOpts=-Djava.security.auth.login.confi=/tmp/kafka/kafka_client_jaas.conf`和其他 JVM 启动参数。
7. 重启Druid服务。

Druid使用Tranquility Kafka

由于 Tranquility 是一个服务，它对于 Kafka 来说是消费者，对于 E-MapReduce Druid 来说是客户端。您可以使用中立的机器来运行 Tranquility，只要这台机器能够同时连通 Kafka 集群和 E-MapReduce Druid 集群即可。

1. Kafka 端创建一个名为 pageViews 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"
 --
 ./bin/kafka-topics.sh --create --zookeeper emr-header-1:2181,emr-header-2:2181,emr-header-3:2181/kafka-1.0.1 --partitions 1 --replication-factor 1 --topic pageViews
```

2. 下载 Tranquility 安装包，并解压至某一路径下。

3. 配置 datasource。

   这里假设您的 topic name 为 pageViews，并且每条 topic 都是如下形式的 json 文件：

```
{"time": "2018-05-23T11:59:43Z", "url": "/foo/bar", "user": "alice",
 "latencyMs": 32}
 {"time": "2018-05-23T11:59:44Z", "url": "/", "user": "bob", "
latencyMs": 11}
 {"time": "2018-05-23T11:59:45Z", "url": "/foo/bar", "user": "bob",
 "latencyMs": 45}
```

   对应的 dataSrouce 的配置如下：

```
{
   "dataSources" : {
     "pageViews-kafka" : {
       "spec" : {
         "dataSchema" : {
           "dataSource" : "pageViews-kafka",
           "parser" : {
             "type" : "string",
             "parseSpec" : {
               "timestampSpec" : {
                 "column" : "time",
                 "format" : "auto"
               },
               "dimensionsSpec" : {
                 "dimensions" : ["url", "user"],
                 "dimensionExclusions" : [
                   "timestamp",
                   "value"
                 ]
               },
               "format" : "json"
             }
           },
           "granularitySpec" : {
             "type" : "uniform",
             "segmentGranularity" : "hour",
             "queryGranularity" : "none"
           },
           "metricsSpec" : [
             {"name": "views", "type": "count"},
             {"name": "latencyMs", "type": "doubleSum", "fieldName":
 "latencyMs"}
           ]
         },
         "ioConfig" : {
           "type" : "realtime"
         },
         "tuningConfig" : {
           "type" : "realtime",
           "maxRowsInMemory" : "100000",
           "intermediatePersistPeriod" : "PT10M",
           "windowPeriod" : "PT10M"
         }
       },
       "properties" : {
         "task.partitions" : "1",
         "task.replicants" : "1",
         "topicPattern" : "pageViews"
       }
```

```
      }
    },
    "properties" : {
      "zookeeper.connect" : "localhost",
      "druid.discovery.curator.path" : "/druid/discovery",
      "druid.selectors.indexing.serviceName" : "druid/overlord",
      "commit.periodMillis" : "15000",
      "consumer.numThreads" : "2",
      "kafka.zookeeper.connect" : "emr-header-1.cluster-500148518:
2181,emr-header-2.cluster-500148518:2181,   emr-header-3.cluster-
500148518:2181/kafka-1.0.1",
      "kafka.group.id" : "tranquility-kafka",
    }
 }
```

4. 运行如下命令启动 Tranquility。

```
./bin/tranquility kafka -configFile
```

5. 在 Kafka 端启动 producer 并发送一些数据。

```
./bin/kafka-console-producer.sh --broker-list emr-worker-1:9092,emr-
worker-2:9092,emr-worker-3:9092 --topic pageViews
```

输入：

```
{"time": "2018-05-24T09:26:12Z", "url": "/foo/bar", "user": "alice",
 "latencyMs": 32}
 {"time": "2018-05-24T09:26:13Z", "url": "/", "user": "bob", "
latencyMs": 11}
 {"time": "2018-05-24T09:26:14Z", "url": "/foo/bar", "user": "bob",
 "latencyMs": 45}
```

在 Tranquility 日志中查看相应的消息，在 E-MapReduce Druid 端则可以看到启动了相应的实时索引 task。

7.7 Superset

E-MapReduce Druid 集群集成了 Superset 工具。Superset 对 E-MapReduce Druid 做了深度集成，同时也支持多种关系型数据库。由于 E-MapReduce Druid 也支持 SQL，所以可以通过 Superset 以两种方式访问 E-MapReduce Druid，即Apache Druid 原生查询语言或者 SQL。

Superset 默认安装在 emr-header-1 节点，目前还不支持 HA。在使用该工具前，确保您的主机能够正常访问 emr-header-1。您可以通过打 SSH 隧道 的方式连接到主机。

1. 登录 Superset

   在浏览器地址栏中输入 *http://emr-header-1:18088* 后回车，打开 Superset 登录界面，默认用户名/密码为 admin/admin，请您登录后及时修改密码。

2. 添加 E-MapReduce Druid 集群

   登录后默认为英文界面，可点击右上角的国旗图标选择合适的语言。接下来在上方菜单栏中依次选择数据源 > Druid 集群来添加一个 E-MapReduce Druid 集群。

   配置好协调机（Coordinator）和代理机（Broker）的地址，注意 E-MapReduce 中默认端口均为相应的开源端口前加数字 1，例如开源 Broker 端口为 8082，E-MapReduce 中为 18082。

3. 刷新或者添加新数据源

   添加好 E-MapReduce Druid 集群之后，您可以单击数据源 > 扫描新的数据源，这时 E-MapReduce Druid 集群上的数据源（datasource）就可以自动被加载进来。

   您也可以在界面上单击数据源 > Druid 数据源自定义新的数据源（其操作等同于写一个 data source ingestion 的 json 文件），步骤如下：

   a. 自定义数据源时需要填写必要的信息，然后保存。

   b. 保存之后点击左侧三个小图标中的第二个，编辑该数据源，填写相应的维度列与指标列等信息。

4. 查询 E-MapReduce Druid

   数据源添加成功后，单击数据源名称，进入查询页面进行查询。

5. （可选）将 E-MapReduce Druid 作为E-MapReduce Druid数据库使用

   Superset 提供了 SQLAlchemy 以多种方言支持各种各样的数据库，其支持的数据库类型如下表所示。

   Superset 亦支持该方式访问 E-MapReduce Druid，E-MapReduce Druid 对应的 SQLAlchemy URI 为 `druid://emr-header-1:18082/druid/v2/sql`，如下图所示，将 E-MapReduce Druid 作为一个数据库添加：

   接下来就可以在 SQL 工具箱里用 SQL 进行查询了。

7.8 常见问题

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
  1. 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。
  2. 在 Overlord 页面观察作业执行情况，如果失败，查看页面上的 logs。
  3. 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。
  4. 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。
  5. 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那个 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 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（如 ioConfig 等）。

常见问题列表

- 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Apache Druid 官方文档。

- 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。 可参见 Apache Druid 官方文档。
- 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，如 hdfs://:9000/。hdfs_master 最好用 IP，如果用域名，要用完整域名，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）。
- 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。设置该路径并确保 E-MapReduce Druid 账户有权限访问即可。
- 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。注意：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。（另注意

Tranquility Kafka 设置中有两个 ZooKeeper 的设置，一个为 zookeeper.connect，连接的 E-MapReduce Druid 集群的 ZooKeeper，一个为 kafka.zookeeper.connect，连接的 Kafka 集群的 ZooKeeper。这两个 ZooKeeper 可能不是一个 ZooKeeper 集群）。

- 索引时 MiddleManager 报找不到类 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）

- 索引时报如下错误：

```
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:?]
```

  这个问题还是因为 java.io.tmp 路径不存在。设置该路径并确保 E-MapReduce Druid 账户有权限访问。

8 Presto

8.1 产品简介

Presto 是一款由 FaceBook 开源的一个分布式 SQL-on—Hadoop 分析引擎。Presto 目前由开源社区和 FaceBook 内部工程师共同维护，并衍生出多个商业版本。

基本特性

Presto 使用 Java 语言进行开发，具备易用、高性能、强扩展能力等特点，具体的：

- 完全支持 ANSI SQL。
- 支持丰富的数据源 Presto可接入丰富的数据来源，如下所示：
  - 与 Hive 数仓互操作
  - Cassandra
  - Kafka
  - MongoDB
  - MySQL
  - PostgreSQL
  - SQL Server
  - Redis
  - Redshift
  - 本地文件
- 支持高级数据结构：
  - 支持数组和 Map 数据。
  - 支持 JSON 数据。
  - 支持 GIS 数据。
  - 支持颜色数据。
- 功能扩展能力强 Presto 提供了多种扩展机制，包括：
  - 扩展数据连接器。
  - 自定义数据类型。
  - 自定义 SQL 函数。

  用户可以根据自身业务特点扩展相应的模块，实现高效的业务处理。

- 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 WebUI，可视化的呈现查询任务执行过程。
  - 支持 JMX 协议。

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，比较适合如下几个应用场景：

- ETL
- Ad-Hoc 查询
- 海量结构化数据/半结构化数据分析
- 海量多维数据聚合/报表

特别需要注意的是，Presto 是一个数仓类产品，其设计目标并不是为了替代 MySQL、PostgreSQL 等传统的 RDBMS 数据库，对事务对支持有限，不适合在线业务场景。

产品优势

EMR Presto 产品除了开源 Presto 本身具有的优点外，还具备如下优势：

- 即买即用 分分钟完成上百节点的 Presto 集群搭建。
- 弹性扩容 简单操作即可完成集群的扩容和缩容。
- 与 EMR 软件栈完美结合，支持处理存储在 OSS 的数据。
- 免运维 7*24 一站式服务。

8.2 快速入门

8.2.1 系统组成

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。

8.2.2 基本概念

本节介绍 Presto 中的基本概念，以便更好到理解 Presto 到工作机制。

数据模型

数据模型即数据的组织形式。Presto 使用 Catalog、Schema 和 Table 这3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 访问改数据源。一次查询可以访问一个或多个 Catalog。

- Schema

  相当于一个数据库示例，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：

Connector

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 实现。

8.2.3 命令行工具

本节介绍如何通过使用命令行工具操作 Presto 控制台。

通过 SSH 登录 EMR 集群，执行下面对命令进入 Presto 控制台：

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

高安全集群使用如下命令形式：

```
$ presto  --server https://emr-header-1:7778  \
          --enable-authentication \
          --krb5-config-path /etc/krb5.conf \
          --krb5-keytab-path  /etc/ecm/presto-conf/presto.keytab \
```

```
          --krb5-remote-service-name presto \
          --keystore-path /etc/ecm/presto-conf/keystore \
          --keystore-password 81ba14ce6084 \
          --catalog hive --schema default \
          --krb5-principal  presto/emr-header-1.cluster-XXXX@EMR.XXXX.
COM
```

- *XXXX* 为集群的 ecm id，为一串数字，可以通过 *cat /etc/hosts* 获取。
- *81ba14ce6084* 为 */etc/ecm/presto-conf/keystore* 的默认密码，建议部署后替换为自己的 keystore.

接下来就可已在该控制台下执行以下命令：

```
presto:default> show schemas;
    Schema
--------------------
default
hive
information_schema
tpch_100gb_orc
tpch_10gb_orc
tpch_10tb_orc
tpch_1tb_orc
(7 rows)
```

执行 `presto --help` 命令可以获取控制台的帮助，各个参数对解释如下所示：

```
--server <server>                          # 指定Coordinator的URI
--user <user>                              # 设置用户名
--catalog <catalog>                        # 指定默认的Catalog
--schema <schema>                          # 指定默认的Schema
--execute <execute>                        # 执行一条语句，然后退出
-f <file>, --file <file>                   # 执行一个SQL文件，然后退出
--debug                                    # 显示调试信息
--client-request-timeout <timeout>         # 指定客户端超时时间，默认为2m
--enable-authentication                    # 使能客户端认证
--keystore-password <keystore password> # KeyStore密码
--keystore-path <keystore path>            # KeyStore路径
--krb5-config-path <krb5 config path>   # Kerberos配置文件路径（默认为/
etc/krb5.conf)
--krb5-credential-cache-path <path>        # Kerberos凭据缓存路径
--krb5-keytab-path <krb5 keytab path>   # Kerberos Key table路径
--krb5-principal <krb5 principal>          # 要使用的Kerberos principal
--krb5-remote-service-name <name>          # 远程Kerberos节点名称
--log-levels-file <log levels>             # 调试日志配置文件路径
--output-format <output-format>            # 批量导出的数据格式，默认为 CSV
--session <session>                        # 指定回话属性，格式如下 key=value
--socks-proxy <socks-proxy>                # 设置代理服务器
--source <source>                          # 设置查询的Source
--version                                  # 显示版本信息
```

```
-h, --help                                  # 显示帮组信息
```

8.2.4 使用 JDBC

Java 应用可以使用 Presto 提供的 JDBC driver 连接数据库，使用方式与一般 RDBMS 数据库差别不大。

在 Maven 中引入

可以在 pom 文件中加入如下配置引入 Presto JDBC driver：

```
<dependency>
    <groupId>com.facebook.presto</groupId>
    <artifactId>presto-jdbc</artifactId>
    <version>0.187</version>
</dependency>
```

Driver 类名

Presto JDBC driver 类为`com.facebook.presto.jdbc.PrestoDriver`。

连接字串

可以使用如下格式的连接字串：

```
jdbc:presto://<COORDINATOR>:<PORT>/[CATALOG]/[SCHEMA]
```

例如：

```
jdbc:presto://emr-header-1:9090                # 连接数据库，使用默认的Catalog和Schema
jdbc:presto://emr-header-1:9090/hive           # 连接数据库，使用Catalog(hive)和默认的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

Presto JDBC driver 支持很多参数，这些参数既可以通过 `Properties` 对象传入，也可以通过 URL 参数传入，这两种方式是等价的。

通过 `Properties` 对象传入示例：

```
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 参数传入示例：

```
String url = "jdbc:presto://emr-header-1:9090/hive/default?user=hadoop
";
Connection connection = DriverManager.getConnection(url);
```

```
......
```

各参数说明如下：

参数名称	格式	参数说明
user	STRING	用户名
password	STRING	密码
socksProxy	\:\	SOCKS 代理服务器地址，如 localhost:1080
httpProxy	\:\	HTTP 代理服务器地址，如 localhost:8888
SSL	true\	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	true\	是否使用规范化主机名，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 缓存路径

Java 示例

下面给出一个 Java 使用 Presto JDBC driver 的例子。

```
.....
// 加载JDBC Driver类
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-1:9090/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
    }
  }
}
```

使用反向代理

可以使用 HAProxy 反向代理 Coodinator，实现通过代理服务访问 Presto 服务。

- 非安全集群代理配置

  非安全集群配置集群代理步骤如下：

  1. 在代理节点安装 HAProxy
  2. 修改 HAProxy 配置(*/etc/haproxy/haproxy.cfg*), 添加如下内容：

     ```
     ......

     listen prestojdbc :9090
         mode tcp
         option tcplog
         balance source
         server presto-coodinator-1 emr-header-1:9090
     ```

  3. 重启 HAProxy 服务

  至此，可以使用代理服务器来访问 Presto 了，只需要将连接的服务器 IP 改为代理服务的 IP 即可。

8.2.5 通过 Gateway 访问

本节介绍使用 HAProxy 反向代理实现通过 Gateway 节点访问 Presto 服务的方法。该方法也很容扩展到其他组件，如 Impala 等。

Gateway 是与 EMR 集群处于同一个内网中的 ECS 服务器，可以使用 Gateway 实现负载均衡和安全隔离。您可以通过控制台页面 > 概览 > 创建 Gateway，也可以通过控制台页面 > 集群管理 > 创建 Gateway来创建对应集群的 Gateway 节点。

 说明:

Gateway 节点默认已经安装了 HAProxy 服务，但没有启动。

普通集群

普通集群配置 Gateway 代理比较简单，只需要配置 HAProxy 反向代理，对 EMR 集群上Header 节点的 Presto Coodrinator 的 9090 端口实现反向代理即可。配置步骤如下：

1. 配置 HAProxy

   通过 SSH 登入Gateway节点，修改 HAProxy 的配置文件*/etc/haproxy/haproxy.cfg*。

   添加如下内容：

```
#---------------------------------------------------------------------
# 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
```

2. 保存退出，使用如下命令重启 HAProxy 服务：

```
$> service haproxy restart
```

3. 配置安全组

   需要配置的规则如下：

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 Header 节点 Coodinator 端口

至此就可以通过 ECS 控制台删除 Header 节点的公网 IP，在自己的客户机上通过 Gateway 访问 Presto 服务了。

- 命令行使用 Presto 的示例请参见：#unique_50
- JDBC 访问 Presto 的示例请参见：#unique_51

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，同时支持 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。另外，EMR Presto Coodinator 集群默认使用 keystore 配置的 CN 为 emr-header-1，只能内网使用，因此需要重新生成 CN=emr-header-1.cluster-xxx 的 keystore。

- HTTPS 认证相关

1. 创建服务端 CN=emr-header-1.cluster-xxx的keystore：

```
[root@emr-header-1 presto-conf]# keytool -genkey -dname "CN=emr-
header-1.cluster-xxx,OU=Alibaba,O=Alibaba,L=HZ, ST=zhejiang, C=CN
" -alias server -keyalg RSA -keystore keystore -keypass 81ba14ce60
84 -storepass 81ba14ce6084 -validity 36500
Warning:
JKS 密钥库使用专用格式。建议使用 "keytool -importkeystore -srckeystore
 keystore -destkeystore keystore -deststoretype pkcs12" 迁移到行业标
准格式 PKCS12。
```

2. 导出证书：

```
[root@emr-header-1 presto-conf]# keytool -export -alias server -
file server.cer -keystore keystore -storepass 81ba14ce6084
存储在文件 <server.cer> 中的证书
Warning:
JKS 密钥库使用专用格式。建议使用 "keytool -importkeystore -srckeystore
 keystore -destkeystore keystore -deststoretype pkcs12" 迁移到行业标
准格式 PKCS12。
```

3. 制作客户端 keystore：

```
[root@emr-header-1 presto-conf]# keytool -genkey -dname "CN=myhost
,OU=Alibaba,O=Alibaba,L=HZ, ST=zhejiang, C=CN" -alias client -
keyalg RSA -keystore client.keystore -keypass 123456 -storepass
123456 -validity 36500
Warning:
JKS 密钥库使用专用格式。建议使用 "keytool -importkeystore -srckeystor
e client.keystore -destkeystore client.keystore -deststoretype
pkcs12" 迁移到行业标准格式 PKCS12。
```

4. 将证书导入到客户端 keystore 中：

```
[root@emr-header-1 presto-conf]# keytool -import -alias server -
keystore client.keystore -file server.cer -storepass 123456
所有者: CN=emr-header-2.cluster-xxx, OU=Alibaba, O=Alibaba, L=HZ,
ST=zhejiang, C=CN
发布者: CN=emr-header-2.cluster-xxx, OU=Alibaba, O=Alibaba, L=HZ,
ST=zhejiang, C=CN
序列号: 4247108
有效期为 Thu Mar 01 09:11:31 CST 2018 至 Sat Feb 05 09:11:31 CST
2118
```

```
证书指纹：
     MD5:  75:2A:AA:40:01:5B:3F:86:8F:9A:DB:B1:85:BD:44:8A
     SHA1: C7:25:B9:AD:5F:FE:FC:05:8E:A0:24:4A:1C:AA:6A:8D:6C:39:
28:16
     SHA256: DB:86:69:65:73:D5:C6:E2:98:7C:4A:3B:31:EF:70:80:F0:3C
:3B:0C:14:94:37:9F:9C:22:47:EA:7E:1E:DE:8C
签名算法名称: SHA256withRSA
主体公共密钥算法: 2048 位 RSA 密钥
版本: 3
扩展:
#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 45 1D A9 C7 D5 4E BB CF   BD CE B4 5E E2 16 FB 2F  E....N
.....^.../
0010: E9 5D 4A B6                                        .]J.
]
]
是否信任此证书? [否]:  是
证书已添加到密钥库中
Warning:
JKS 密钥库使用专用格式。建议使用 "keytool -importkeystore -srckeystor
e client.keystore -destkeystore client.keystore -deststoretype
pkcs12" 迁移到行业标准格式 PKCS12。
```

5. 将生成的文件拷贝到客户端：

```
$> scp root@xxx.xxx.xxx.xxx:/etc/ecm/presto-conf/client.keystore
 ./
```

- Kerberos 认证相关

1. 添加客户端用户 principal：

```
[root@emr-header-1 presto-conf]# sh /usr/lib/has-current/bin/
hadmin-local.sh /etc/ecm/has-conf -k /etc/ecm/has-conf/admin.
keytab
[INFO] conf_dir=/etc/ecm/has-conf
Debug is  true storeKey true useTicketCache false useKeyTab true
doNotPrompt true ticketCache is null isInitiator true KeyTab is /
etc/ecm/has-conf/admin.keytab refreshKrb5Config is true principal
is kadmin/EMR.xxx.COM@EMR.xxx.COM tryFirstPass is false useFirstPa
ss is false storePass is false clearPass is false
Refreshing Kerberos configuration
principal is kadmin/EMR.xxx.COM@EMR.xxx.COM
Will use keytab
Commit Succeeded
Login successful for user: kadmin/EMR.xxx.COM@EMR.xxx.COM
enter "cmd" to see legal commands.
HadminLocalTool.local: addprinc -pw 123456 clientuser
Success to add principal :clientuser
HadminLocalTool.local: ktadd -k /root/clientuser.keytab clientuser
Principal export to keytab file : /root/clientuser.keytab
successful .
HadminLocalTool.local: exit
```

2. 将生成的文件拷贝到客户端：

```
$> scp root@xxx.xxx.xxx.xxx:/root/clientuser.keytab ./
```

```
$> scp root@xxx.xxx.xxx.xxx:/etc/krb5.conf ./
```

3. 修改拷贝到客户端的 *krb5.conf* 文件，修改如下两处：

```
[libdefaults]
    kdc_realm = EMR.xxx.COM
    default_realm = EMR.xxx.COM
    # 改成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
    }
```

4. 修改客户端主机的 hosts 文件，添加如下内容：

```
#  gateway ip
xxx.xxx.xxx.xxx emr-header-1.cluster-xxx
```

- 配置 Gateway HAProxy

1. 通过 SSH 登入到 Gateway 节点，修改*/etc/haproxy/haproxy.cfg*。添加如下内容：

```
#---------------------------------------------------------------------
# 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
```

2. 保存退出，使用如下命令重启HAProxy服务：

```
$> service haproxy restart
```

3. 配置安全组规则

需要配置的规则如下：

方向	配置规则	说明
公网入	自定义 UDP，开放 88 端口	该端口用户 HAProxy 代理 Header 节点上的 KDC
公网入	自定义 TCP，开放 88 端口	该端口用户 HAProxy 代理Header节点上的 KDC

方向	配置规则	说明
公网入	自定义 TCP，开放 7778 端口	该端口用于 HAProxy 代理 Header 节点 Coodinator 端口

至此，就可以通过 ECS 控制台删除 Header 节点的公网 IP，在自己的客户机上通过 Gateway 访问 Presto 服务了。

- 使用 JDBC 访问 Presto 示例

  代码如下：

```
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-59824:7778/hive
/default";
    Properties properties = new Properties();
    properties.setProperty("user", "hadoop");
    // https相关配置
    properties.setProperty("SSL", "true");
    properties.setProperty("SSLTrustStorePath", "resources/59824/
client.keystore");
    properties.setProperty("SSLTrustStorePassword", "123456");
    // Kerberos相关配置
    properties.setProperty("KerberosRemoteServiceName", "presto");
    properties.setProperty("KerberosPrincipal", "clientuser@EMR.
59824.COM");
    properties.setProperty("KerberosConfigPath", "resources/59824/
krb5.conf");
    properties.setProperty("KerberosKeytabPath", "resources/59824/
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
        }
     }
}
```

小结

本文介绍了使用 HAProxy 反向代理实现通过 Gateway 节点访问 Presto 服务的方法。该方法也很容扩展到其他组件，如 Impala。

8.2.6 使用 ApacheDS 进行认证

Presto 可以对接 LDAP，实现用户密码认证。只需要 Coordinator 节点对接 LDAP 即可

主要步骤

1. 配置 ApacheDS，启用 LDAPS
2. 在 ApacheDS 中创建用户信息
3. 配置 Presto Coordinator，重启生效
4. 验证配置

启用 LDAPS

1. 创建 ApacheDS 服务端使用的 keystore, 此处密码全部使用 '123456':

```
## 创建keystore
> cd /var/lib/apacheds-2.0.0-M24/default/conf/
> keytool -genkeypair -alias apacheds -keyalg RSA -validity 7 -
keystore ads.keystore

Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  apacheds
What is the name of your organizational unit?
  [Unknown]:  apacheds
What is the name of your organization?
  [Unknown]:  apacheds
What is the name of your City or Locality?
  [Unknown]:  apacheds
What is the name of your State or Province?
  [Unknown]:  apacheds
What is the two-letter country code for this unit?
  [Unknown]:  CN
Is CN=apacheds, OU=apacheds, O=apacheds, L=apacheds, ST=apacheds, C=
CN correct?
  [no]:  yes
```

```
Enter key password for <apacheds>
    (RETURN if same as keystore password):
Re-enter new password:

Warning:
The JKS keystore uses a proprietary format. It is recommended to
 migrate to PKCS12 which is an industry standard format using "
keytool -importkeystore -srckeystore ads.keystore -destkeystore ads.
keystore -deststoretype pkcs12".

## 修改文件用户，否则ApacheDS没有权限读取
> chown apacheds:apacheds ./ads.keystore

## 导出证书。
## 需要输入密码，密码为上一步设置的值，这里为：123456
> keytool -export -alias apacheds -keystore ads.keystore -rfc -file
apacheds.cer
Enter keystore password:
Certificate stored in file <apacheds.cer>

Warning:
The JKS keystore uses a proprietary format. It is recommended to
 migrate to PKCS12 which is an industry standard format using "
keytool -importkeystore -srckeystore ads.keystore -destkeystore ads.
keystore -deststoretype pkcs12".

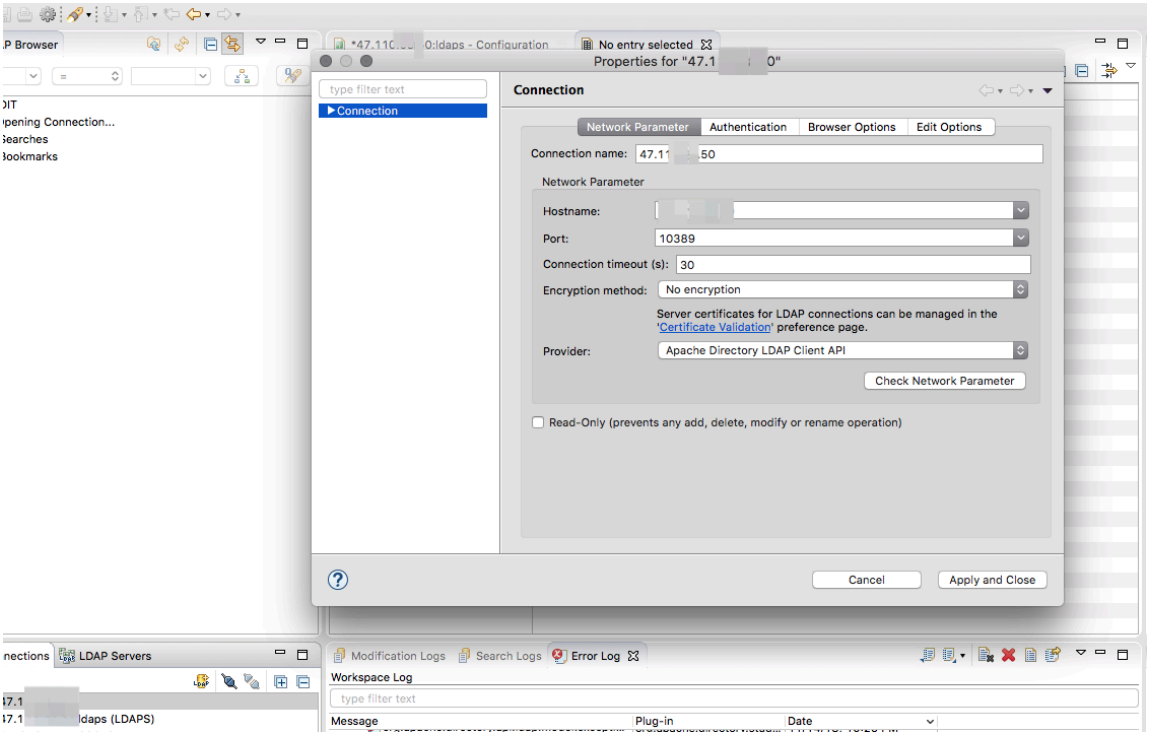
## 将证书导入系统证书库，实现自认证
```

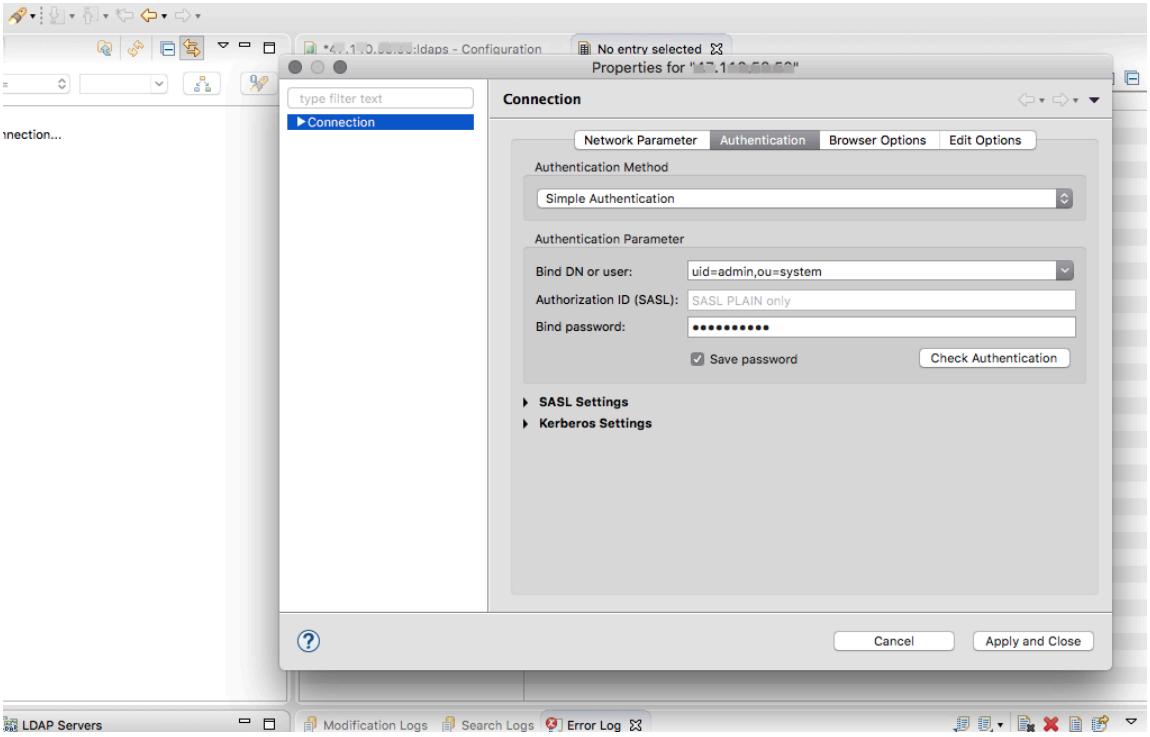
```
> keytool -import -file apacheds.cer -alias apacheds -keystore /usr/
lib/jvm/java-1.8.0/jre/lib/security/cacerts
```

2. **修改配置，启用 LDAPS**

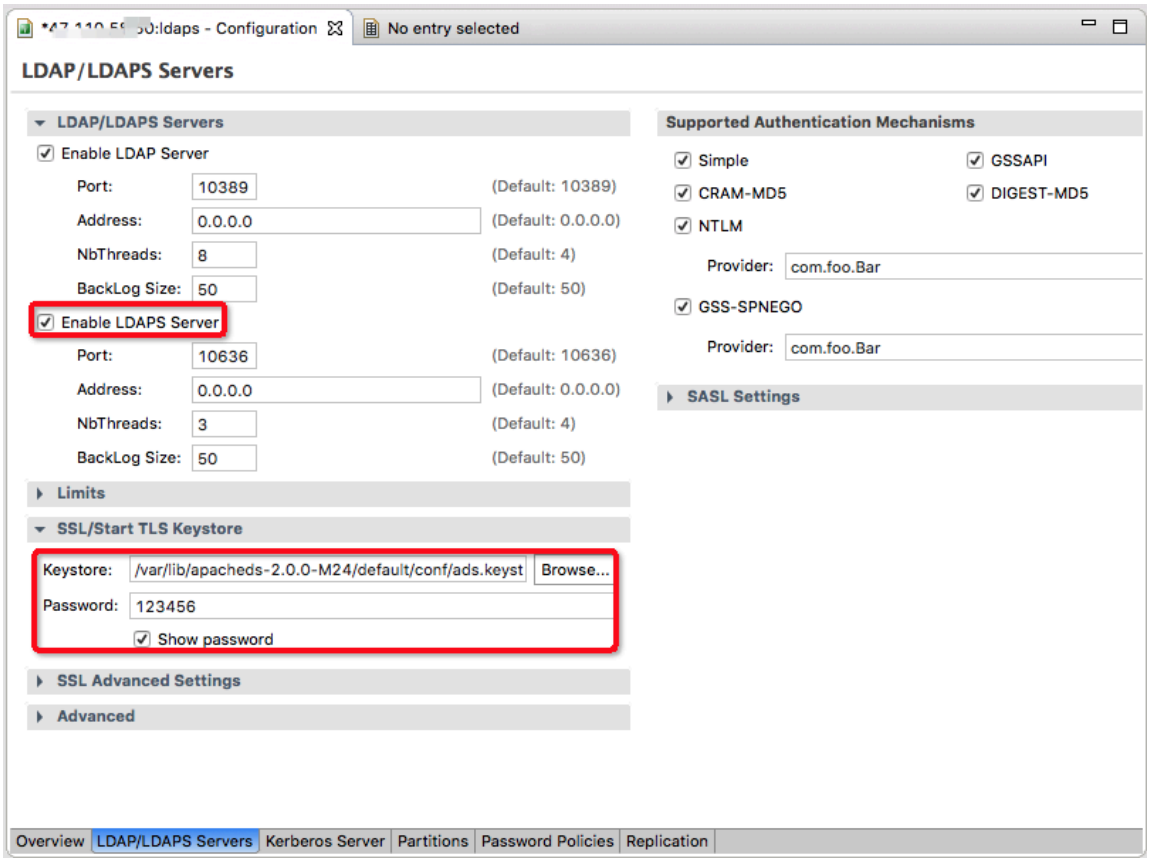
   打开 ApacheDS Studio，链接到集群上到 ApacheDS 服务：

   - **DN 设置为：uid=admin,ou=system**
   - **密码在此文件中查看：** *`/var/lib/ecm-agent/cache/ecm/service/APACHEDS/2.0.0.1.1/package/files/modifypwd.ldif`*





链接后，打开配置页，启用 LDAPs，将第一步创建的 keystore 设置到相关配置中，保存（ctrl + s）。



3. 重启 ApacheDS 服务

登入集群，执行如下命令重启 ApacheDS：

```
> service apacheds-2.0.0-M24-default restart
```

到此，LDAPS 启动， 服务端口是 10636。

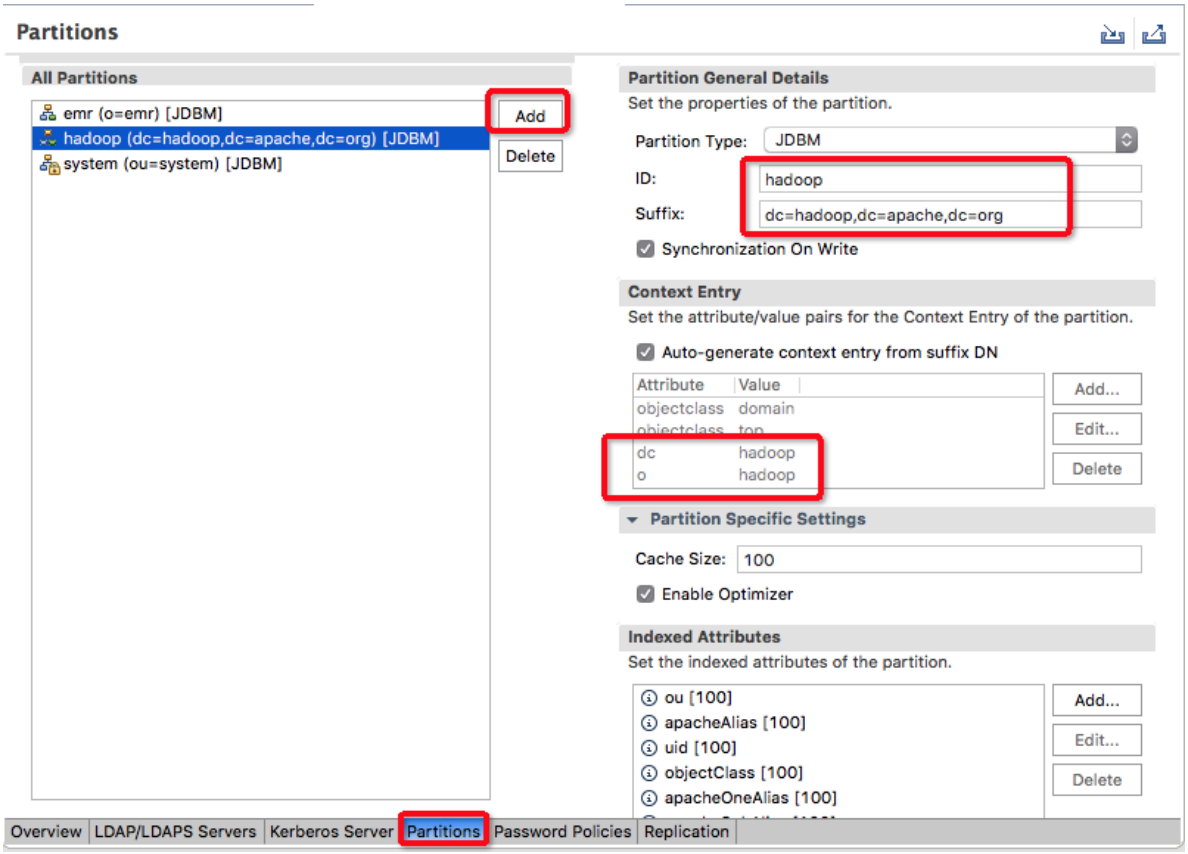
说明：

ApacheDS Studio 有 Bug，在连接属性页测试 LDAPS 服务连接时会报握手失败，主要是内部默认的超时时间太短导致的，不会影响实际使用。

创建用户信息

本用例在 DN: dc=hadoop,dc=apache,dc=org 下创建相关用户。

1. 创建 dc=hadoop,dc=apache,dc=org 分区，打开配置页，作如下配置，保存（ctrl+s）。重启 ApacheDS 服务生效。



2. 创建用户

登入集群，创建如下文件：*/tmp/users.ldif*

```
# Entry for a sample people container
# Please replace with site specific values
dn: ou=people,dc=hadoop,dc=apache,dc=org
objectclass:top
objectclass:organizationalUnit
ou: people

# Entry for a sample end user
# Please replace with site specific values
dn: uid=guest,ou=people,dc=hadoop,dc=apache,dc=org
objectclass:top
objectclass:person
objectclass:organizationalPerson
objectclass:inetOrgPerson
cn: Guest
sn: User
uid: guest
userPassword:guest-password

# entry for sample user admin
dn: uid=admin,ou=people,dc=hadoop,dc=apache,dc=org
objectclass:top
objectclass:person
objectclass:organizationalPerson
objectclass:inetOrgPerson
```

```
cn: Admin
sn: Admin
uid: admin
userPassword:admin-password

# entry for sample user sam
dn: uid=sam,ou=people,dc=hadoop,dc=apache,dc=org
objectclass:top
objectclass:person
objectclass:organizationalPerson
objectclass:inetOrgPerson
cn: sam
sn: sam
uid: sam
userPassword:sam-password

# entry for sample user tom
dn: uid=tom,ou=people,dc=hadoop,dc=apache,dc=org
objectclass:top
objectclass:person
objectclass:organizationalPerson
objectclass:inetOrgPerson
cn: tom
sn: tom
uid: tom
userPassword:tom-password

# create FIRST Level groups branch
dn: ou=groups,dc=hadoop,dc=apache,dc=org
objectclass:top
objectclass:organizationalUnit
ou: groups
description: generic groups branch

# create the analyst group under groups
dn: cn=analyst,ou=groups,dc=hadoop,dc=apache,dc=org
objectclass:top
objectclass: groupofnames
cn: analyst
description:analyst  group
member: uid=sam,ou=people,dc=hadoop,dc=apache,dc=org
member: uid=tom,ou=people,dc=hadoop,dc=apache,dc=org

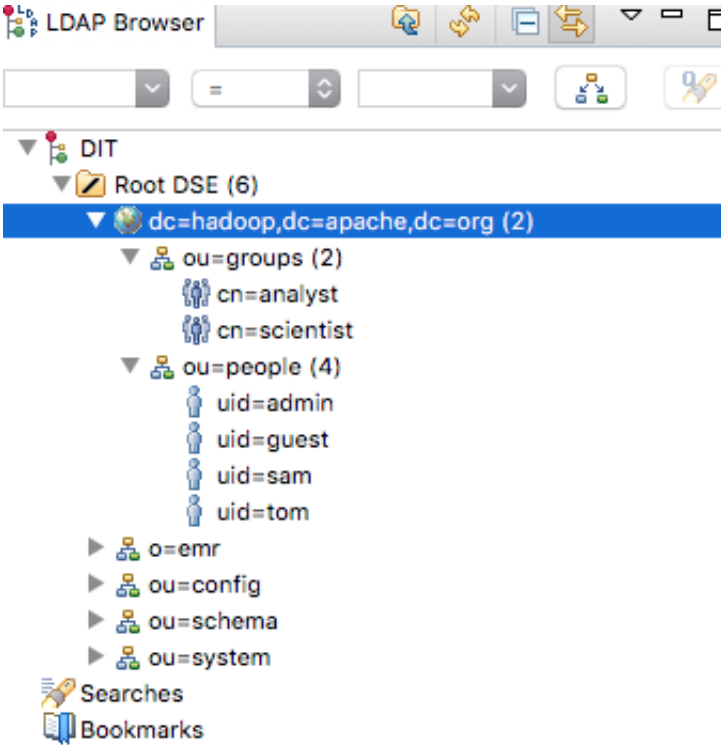

# create the scientist group under groups
dn: cn=scientist,ou=groups,dc=hadoop,dc=apache,dc=org
objectclass:top
objectclass: groupofnames
cn: scientist
description: scientist group
```

```
member: uid=sam,ou=people,dc=hadoop,dc=apache,dc=or
```

执行如下命令，导入用户：

```
> ldapmodify -x -h localhost -p 10389 -D "uid=admin,ou=system" -w
 {密码} -a -f /tmp/users.ldif
```

执行完成后，可以在 ApacheDS Studio 上看到相关到用户，如下所示：



配置 Presto

1. 开启 Coordinator Https

   a. 创建 Presto coordinator 使用的 keystore

   ```
   ## 使用EMR自带的脚本生成keystore
   ## keystore地址：/etc/ecm/presto-conf/keystore
   ## keystore密码：81ba14ce6084
   > expect /var/lib/ecm-agent/cache/ecm/service/PRESTO/0.208.0.1.2/
   package/files/tools/gen-keystore.exp
   ```

   b. 配置 Presto coordinator 配置

   编辑*/etc/ecm/presto-conf/config.properties*， 加入如下内容：

   ```
   http-server.https.enabled=true
   http-server.https.port=7778

   http-server.https.keystore.path=/etc/ecm/presto-conf/keystore
   ```

```
http-server.https.keystore.key=81ba14ce6084
```

2. 配置认证模式，接入 ApacheDS

   a. 编辑 */etc/ecm/presto-conf/config.properties*， 加入如下内容：

   ```
   http-server.authentication.type=PASSWORD
   ```

   b. 编辑 *jvm.config*， 加入如下内容：

   ```
   -Djavax.net.ssl.trustStore=/usr/lib/jvm/java-1.8.0/jre/lib/
   security/cacerts
   -Djavax.net.ssl.trustStorePassword=changeit
   ```

   c. 创建 *password-authenticator.properties*，加入如下内容：

   ```
   password-authenticator.name=ldap
   ldap.url=ldaps://emr-header-1.cluster-84423:10636
   ldap.user-bind-pattern=uid=${USER},ou=people,dc=hadoop,dc=apache,
   dc=org
   ```

   d. 创建 *jndi.properties*， 加入如下内容：

   ```
   java.naming.security.principal=uid=admin,ou=system
   java.naming.security.credentials={密码}
   java.naming.security.authentication=simple
   ```

   e. 将 *jndi.properties* 打包到 jar 包中，复制到 presto 库文件目录中：

   ```
   jar -cvf jndi-properties.jar jndi.properties
   > cp ./jndi-properties.jar /etc/ecm/presto-current/lib/
   ```

    说明：

   - 下面 3 个参数用于登入 LDAP 服务。然而，在 Presto 上没地方配置这几个参数。分析源码可以先将这几个参数加到 jvm 参数里，并不会生效。（会被过滤掉）： java.naming.security.principal=uid=admin,ou=system java.naming.security.credentials=ZVixyOY+5k java.naming.security.authentication=simple
   - 进一步分析代码，发现 JNDI 库会用 classload 加载 jndi.properties 这个资源文件，因此可以将这几个参数放到 jndi.properties 这个文件中；
   - Presto 的 launcher 只会把 jar 文件加到 classpath 里，所以还需把这个 jndi.properties 打成 jar 包，拷贝到 lib 目录中。

3. 重启 Presto，至此完成所有配置

验证配置

使用 Presto cli 验证配置是否生效。

```
## 使用用户sam，输入正确的密码
```

```
> presto  --server https://emr-header-1:7778  --keystore-path /etc/ecm
/presto-conf/keystore --keystore-password 81ba14ce6084 --catalog hive
 --schema default --user sam --password
Password: <输入了正确的密码>
presto:default> show schemas;
              Schema
--------------------------------
 tpcds_bin_partitioned_orc_5
 tpcds_oss_bin_partitioned_orc_10
 tpcds_oss_text_10
 tpcds_text_5
 tst
(5 rows)

Query 20181115_030713_00002_kp5ih, FINISHED, 3 nodes
Splits: 36 total, 36 done (100.00%)
0:00 [20 rows, 331B] [41 rows/s, 694B/s]

## 使用用户sam，输入错误的密码
> presto  --server https://emr-header-1:7778  --keystore-path /etc/ecm
/presto-conf/keystore --keystore-password 81ba14ce6084 --catalog hive
 --schema default --user sam --password
Password: <输入了错误的密码>
presto:default> show schemas;
Error running command: Authentication failed: Access Denied: Invalid
credentials
```

8.3 概述

本章主要面向应用开发人员，将介绍如下内容：

- Presto 数据库的 SQL 语法和特性
- Presto 数据库的性能调优方法
- 开发 Presto数据库插件，扩展 Presto 数据库的功能

8.4 数据类型

Presto 默认支持多种常见的数据类型，包括布尔类型、整型、浮点型、字符串、日期等。同时，用户可以通过插件等方式增加自定义的数据类型。并且，自定义的 Presto 连接器不需要支持所有数据类型。

数值类型

Presto 内置支持如下几种数值类型：

- `BOOLEAN`

  表示一个二值选项，值为`TRUE`或`FALSE`

- `TINYINT`

  表示一个 8 位有符号整型,二进制补码形式存储

- `SMALLINT`

  表示一个 16 位有符号整型，二进制补码形式存储

- `INTEGER`

  表示一个 32 位有符号整型，二进制补码形式存储

- `BIGINT`

  表示一个 64 位有符号整型，二进制补码形式存储

- `REAL`

  一个 32 位多精度的[IEEE-754]二进制浮点数值实现

- `DOUBLE`

  一个 64 位多精度的[IEEE-754]二进制浮点数值实现

- `DECIMAL`

  一个固定精度的数值类型，最大可支持 38 位有效数字，有效数字在 17 位以下性能最好。定义`DECIMAL`类型字段时需要确定两个字面参数：

  1. 精度（precision）数值总的位数，不包括符号位
  2. 范围（scale）小数位数，可选参数，默认为 0

  示例：`DECIMAL '-10.7'`该值可用`DECIMAL(3,1)`类型表示。

下表说明整型数值类型的位宽和取值范围：

数值类型	位宽	最小值	最大值
TINYINT	8 bit	-2^7	2^7 - 1
SMALLINT	16 bit	2^15	2^15 - 1
INTEGER	32 bit	-2^31	-2^31 - 1
BIGINT	64 bit	-2^63	-2^63 - 1

字符类型

Presto 内置支持如下几种字符类型：

- `VARCHAR`

  表示一个可变长度的字符串类型，可以设置最大长度。

  示例：`VARCHAR`, `VARCHAR(10)`

- CHAR

  表示一个固定长度的字符串类型，使用时可以指定字符串的长度，不指定则默认为1。

  示例：CHAR, CHAR(10)

  > **注意:**
  > 指定了长度的字符串总是包含与该长度相同的字符，如果字符串字面长度小于指定长度，则不足部分会用不可见字符填充，填充部分也会参与字符比较，因此，两个字面量一样的字段，如果它们的长度定义不一样，就永远不可能相等。

- VARBINARY 表示一块可变长度的二进制数据。

日期和时间

Presto 内置支持如下几种时间和日期类型：

- DATE

  表示一个日期类型的字段，日期包括年、月、日，但是不包括时间。

  示例： DATE '1988-01-30'

- TIME

  表示一个时间类型，包括时、分、秒、毫秒。时间类型可以加时区进行修饰。

  示例：

  - TIME '18:01:02.345'， 无时区定义，使用系统时区进行解析。
  - TIME '18:01:02.345 Asia/Shanghai'， 有时区定义，使用定义的时区进行解析。

- TIMESTAMP

  表示一个时间戳类型的字段，时间戳包含了日期和时间两个部分的信息，取值范围为'1970-01-01 00:00:01' UTC到'2038-01-19 03:14:07' UTC，支持使用时区进行修饰。

  示例：TIMESTAMP '1988-01-30 01:02:03.321', TIMESTAMP '1988-01-30 01:02:03.321 Asia/Shanghai'

- INTERVAL

  主要用于时间计算表达式中，表示一个间隔，单位可以是如下几个：

  - YEAR - 年
  - QUARTER - 季度
  - MONTH - 月
  - DAY - 天
  - HOUR - 小时
  - MINUTE - 分钟
  - SECOND - 秒
  - MILLISECOND - 毫秒

  示例：DATE '2012-08-08' + INTERVAL '2' DAY

复杂类型

Presto内置支持多种复杂的数据类型，以便支持更加复杂的业务场景，这些类型包括：

- JSON

  表示字段为一个 JSON 字符串，包括 JSON 对象、JSON 数组、JSON 单值(数值或字符串)，还包括布尔类型的true、false以及表示空的null。

  示例：

  - JSON '[1, null, 1988]'
  - JSON '{"k1":1, "k2": "abc"}'

- ARRAY

  表示一个数组，数组中各个元素的类型必须一致。

  示例：ARRAY[1, 2, 3]

- MAP

  表示一个映射关系，由键数组和值数组组成。

  示例：MAP(ARRAY['foo', 'bar'], ARRAY[1, 2])

- ROW

  表示一行数据，行中每个列都有列名对应，可以使用.运算符+列名的方式来访问数据列。

  示例：CAST(ROW(1988, 1.0, 30) AS ROW(y BIGINT, m DOUBLE, d TINYINT))

- `IPADDRESS`

  表示一个 IPv4 或 IPv6 地址。内部实现上，将 IPv4 处理成 IPv6 地址使用（IPv4 到 IPv6 映射表）。

  示例：`IPADDRESS '0.0.0.0'`, `IPADDRESS '2001:db8::1'`

8.5 常用连接器

8.5.1 Kafka 连接器

本连接器将 Kafka 上的 topic 映射为 Presto 中的表。Kafka 中的每条记录都映射为 Presto 表中的消息。

> **注意:**
>
> 由于 Kafka 中数据是动态变化的，因此，在使用 Presto 进行多次查询时，可能会出现数据不一致的情形。目前，Presto 还没有处理这些情况的能力。

配置

创建`etc/catalog/kafka.properties`文件，添加如下内容，一边启用 Kafka 连接器。

```
connector.name=kafka
kafka.table-names=table1,table2
kafka.nodes=host1:port,host2:port
```

> **说明:**
>
> Presto 可以同时连接多个 Kafka 集群，只需要在配置目录中添加新的 properties 文件即可，文件名将会被映射为 Presto 的 catalog。如新增一个"orders.properties"配置文件，Presto 会创建一个新的名为'orders'的 catalog。

```
## orders.properties
connector.name=kafka   # 这个表示连接器类型，不能变
kafka.table-names=tableA,tableB
kafka.nodes=host1:port,host2:port
```

Kafka 连接器提供了如下几个属性：

参数	是否必选	描述	说明
kafka.table-names	是	定义本连接器支持的表格列表	这里的文件名可以使用 Schema 名称进行修饰，形式如{schema_name}.{table_name}。也可以不使用 Schema 名称修饰，此时，表格将被映射到kafka.default-schema中定义的schema中。
kafka.default-schema		默认的Schema名称，默认值为default	
kafka.nodes	是	Kafka集群中的节点列表	配置格式形如hostname:port[,hostname:port...]。此处可以只配置部分 Kafka 节点，但是，Presto 必须能够连接到 Kafka 集群中的所有节点。否则，可能拿不到部分数据。
kafka.connect-timeout	否	连接器与 Kafka 集群的超时时间，默认为 10 秒	如果 Kafka 集群压力比较大，创建连接可能需要相当长的时间，从而导致 Presto 在执行查询时出现超时的情况。此时，增加当前的配置值是一个不错的选择。
kafka.buffer-size	否	读缓冲区大小，默认为 64 kb	用于设置从 Kafka 读取数据的内部数据缓冲区的大小。 数据缓冲区必须至少大于一条消息的大小。 每个 Worker 和数据节点都会分配一个数据缓冲区。

参数	是否必选	描述	说明
kafka.table-description-dir	否	Topic(表)描述文件目录，默认为`etc/kafka`	该目录下保存着 JSON 格式的数据表定义文件（必须使用`.json`作为后缀）。
kafka.hide-internal-columns	否	需要隐藏的预置列清单，默认为`true`	除了在表格描述文件中定义的数据列之外，连接器还为每个表格维护了许多额外的列。本属性用于控制这些列在是否会在`DESCRIBE <table-name>`和`SELECT *`语句的执行结果中显示。无论本配置设置是什么，这些列都可以参与查询过程中。

· Kafka连接器提供的内部列如下表所示：

列名	类型	说明
_partition_id	BIGINT	本行记录所在的 Kafka 分区 ID。
_partition_offset	BIGINT	本行记录在 Kafka 分区中的偏移量。
_segment_start	BIGINT	包含此行的数据段的最低偏移量。此偏移量是针对每个分区的。
_segment_end	BIGINT	包含此行的数据段的最大偏移量(为下一个段的起始偏移量)。此偏移量是针对每个分区的。
_segment_count	BIGINT	当前行在数据段中的序号. 对于没有压缩的topic, `_segment_start`+`_segment_count` = `_partition_offset`.
_message_corrupt	BOOLEAN	如果解码器无法解码本行记录，本字段将会被设为`TRUE`。

列名	类型	说明
_message	VARCHAR	将消息字节作为 UTF-8 编码的字符串。当 topic 的消息为文本类型是，这个字段比较有用。
_message_length	BIGINT	本行消息的字节长度。
_key_corrupt	BOOLEAN	如果键解码器无法解码本行记录，该字段将会被设为`TRUE`。
_key	VARCHAR	将键字节作为 UTF-8 编码的字符串。当 topic 的消息为文本类型是，这个字段比较有用。
_key_length	BIGINT	消息中键的字节长度。

说明:

对于那些没有定义文件的表，`_key_corrupt`和`_message_corrupt`默认为`FALSE`。

表格定义文件

Kafka 本身是 Schema-Less 的消息系统，消息的格式需要生产者和消费者自己来定义。而 Presto 要求数据必须可以被映射成表的形式。因此，通常需要用户根据消息的实际合适，提供对应的表格定义文件。对于 JSON 格式的消息，如果没有提供定义文件，也可以在查询时使用Presto的JSON函数进行解析。这种处理方式虽然比较灵活，但是对会增加 SQL 语句的编写难度。

一个表格描述文件使用 JSON 来定义一个表，文件名可以任意，但是必须以`.json`作为扩展名。

```
{
    "tableName": ...,
    "schemaName": ...,
    "topicName": ...,
    "key": {
        "dataFormat": ...,
        "fields": [
            ...
        ]
    },
    "message": {
        "dataFormat": ...,
        "fields": [
            ...
       ]
    }
}
```

字段	可选性	类型	说明
tableName	必选	string	Presto 表名
schemaName	可选	string	表所在的 Schema 名称
topicName	必选	string	Kafka topic 名称
key	可选	JSON object	消息键到列的映射规则
message	可选	JSON object	消息到列的映射规则

其中，键和消息的映射规则使用如下几个字段来描述。

字段	可选性	类型	说明
dataFormat	必选	string	设置一组列的解码器
fields	必选	JSON 数组	列定义列表

这里的`fields`是一个 JSON 数组，每一个元素为如下的 JOSN 对象：

```
{
    "name": ...,
    "type": ...,
    "dataFormat": ...,
    "mapping": ...,
    "formatHint": ...,
    "hidden": ...,
    "comment": ...
}
```

字段	可选性	类型	说明
name	必选	string	列名
type	必选	string	列的数据类型
dataFormat	可选	string	列数据解码器
mapping	可选	string	解码器参数
formatHint	可选	string	设置在该列上的提示，可以被解码器使用
hiddent	可选	boolean	是否隐藏
comment	可选	string	列的描述

解码器

解码器的功能是将 Kafka 的消息（key+message）映射到数据表到列中。如果没有表的定义文件，Presto 将使用`dummy`解码器。

Kafka 连接器提供了如下 3 中解码器：

- `raw` - 不做转换，直接使用原始的 bytes
- `csv` - 将消息作为 CSV 格式的字符串进行处理
- `json` - 将消息作为 JSON 进行处理

8.5.2 JMX 连接器

可以通过 JMX 连接器查询 Presto 集群中所有节点的 JMX 信息。本连接器通常用于系统监控和调试。通过修改本连接器的配置，可以实现 JMX 信息定期转储的功能。

配置

创建`etc/catalog/jmx.properties`文件，添加如下内容，一边启用 JMX 连接器。

```
connector.name=jmx
```

如果希望定期转储 JMX 数据，可以在配置文件中添加如下内容：

```
connector.name=jmx
jmx.dump-tables=java.lang:type=Runtime,com.facebook.presto.execution.
scheduler:name=NodeScheduler
jmx.dump-period=10s
jmx.max-entries=86400
```

其中：

- `dump-tables`是用逗号隔开的 MBean（Managed Beans）列表。该配置项指定了每个采样周期哪些 MBean 指标会被采样并存储到内存中；
- `dump-period`用于设置采样周期，默认为 10 s；
- `max-entries`用于设置历史记录的最大长度，默认为 86400 条。

如果指标项的名称中包含逗号，则需要使用`\\,`进行转义，如下所示：

```
connector.name=jmx
jmx.dump-tables=com.facebook.presto.memory:type=memorypool\\,name=
general,\
   com.facebook.presto.memory:type=memorypool\\,name=system,\
   com.facebook.presto.memory:type=memorypool\\,name=reserved
```

```
```

数据表

JMX 连接器提供了两个 schemas，分别为`current`和`history`。其中：

`current`中包含了 Presto 集群中每个节点的当前的 MBean，BMean 的名称即为`current`中的表名(如果 bean 的名称中包含非标准字符，则需在查询时用双引号将表名扩起来)，可以通过如下语句获取：

```
SHOW TABLES FROM jmx.current;
```

示例：

```
--- 获取每个节点的jvm信息
SELECT node, vmname, vmversion
FROM jmx.current."java.lang:type=runtime";

      node    |              vmname                | vmversion
--------------+------------------------------------+-----------
 ddc4df17-xxx | Java HotSpot(TM) 64-Bit Server VM | 24.60-b09
(1 row)
```

```
--- 获取每个节点最大和最小的文件描述符个数指标
SELECT openfiledescriptorcount, maxfiledescriptorcount
FROM jmx.current."java.lang:type=operatingsystem";

 openfiledescriptorcount | maxfiledescriptorcount
-------------------------+------------------------
                     329 |                  10240
(1 row)
```

`history`中包含了配置文件中配置的需要转储的指标对应的数据表。可以通过如下语句进行查询：

```
SELECT "timestamp", "uptime" FROM jmx.history."java.lang:type=runtime
";

        timestamp        | uptime
-------------------------+--------
 2016-01-28 10:18:50.000 |  11420
 2016-01-28 10:19:00.000 |  21422
 2016-01-28 10:19:10.000 |  31412
(3 rows)
```

8.5.3 系统连接器

通过本连接器可以使用 SQL 查询 Presto 集群的基本信息和度量。

配置

无需配置，所有信息都可以通过名为`system`的 catalog 获取。

示例：

```
--- 列出所有支持的数据项目
SHOW SCHEMAS FROM system;

--- 列出运行时项目中的所有数据项
SHOW TABLES FROM system.runtime;

--- 获取节点状态
SELECT * FROM system.runtime.nodes;

     node_id |         http_uri         | node_version | coordinator
 | state
-------------+--------------------------+--------------+-------------
+--------
 3d7e8095-...| http://192.168.1.100:9090| 0.188        | false
 | active
 7868d742-...| http://192.168.1.101:9090| 0.188        | false
 | active
 7c51b0c1-...| http://192.168.1.102:9090| 0.188        | true
 | active

--- 强制取消一个查询
CALL system.runtime.kill_query('20151207_215727_00146_tx3nr');
```

数据表

本连接器提供了如下几个数据表：

TABLE	SCHEMA	说明
catalogs	metadata	该表包含了本连接器支持的所有 catalog 列表
schema_properties	metadata	本表包含创建新 Schema 时可以设置的可用属性列表。
table_properties	metadata	本表包含创建新表时可以设置的可用属性列表。
nodes	runtime	本表包含了 Presto 集群中所有可见节点及其状态列表
queries	runtime	查询表包含了 Presto 群集上当前和最近运行的查询的信息，包括原始查询文本（SQL），运行查询的用户的身份以及有关查询的性能信息，如查询排队和分析的时间等。
tasks	runtime	任务表包含了关于 Presto 查询中涉及的任务的信息，包括它们的执行位置以及每个任务处理的行数和字节数。

TABLE	SCHEMA	说明
transactions	runtime	本表包含当前打开的事务和相关元数据的列表。这包括创建时间，空闲时间，初始化参数和访问目录等信息。

存储过程

本连接器支持如下存储过程：

- `runtime.kill_query(id)`

  取消给定 id 的查询

8.6 SQL 语句

8.6.1 SQL 语句一览

DDL

- ALTER SCHEMA
- ALTER TABLE
- CREATE SCHEMA
- CREATE TABLE
- CREATE TABLE AS
- CREATE VIEW
- DROP SCHEMA
- DROP TABLE
- DROP VIEW
- VALUES
- DESCRIBE
- SHOW CATALOGS
- SHOW COLUMNS
- SHOW CREATE TABLE
- SHOW CREATE VIEW
- SHOW FUNCTIONS
- SHOW PARTITIONS
- SHOW SCHEMAS

- SHOW TABLES

DML

- DELETE
- INSERT
- EXPLAIN
- EXPLAIN ANALYZE

DQL

- 查询
  - SELECT
- 预编译
  - PREPARE
  - EXECUTE
  - DEALLOCATE PREPARE
  - DESCRIBE INPUT
  - DESCRIBE OUTPUT

8.6.2 ALTER SCHEMA

概要

```
ALTER SCHEMA name RENAME TO new_name
```

描述

重命名 SCHEMA。

示例

```
ALTER SCHEMA web RENAME TO traffic -- 将Schema 'web'重命名为'traffic'
```

8.6.3 ALTER TABLE

概要

```
ALTER TABLE name RENAME TO new_name
ALTER TABLE name ADD COLUMN column_name data_type
ALTER TABLE name DROP COLUMN column_name
```

```
ALTER TABLE name RENAME COLUMN column_name TO new_column_name
```

描述

更新表格定义。

示例

```
ALTER TABLE users RENAME TO people; --- 重命名
ALTER TABLE users ADD COLUMN zip varchar; --- 添加列
ALTER TABLE users DROP COLUMN zip; --- 删除列
ALTER TABLE users RENAME COLUMN id TO user_id; --- 重命名列
```

8.6.4 CALL

概要

```
CALL procedure_name ( [ name => ] expression [, ...] )
```

描述

调用存储过程。存储过程可以由连接器提供，用于数据操作或管理任务。如果底层存储系统具有自己的存储过程框架，如PostgreSQL，则需要通过连接器提供的存储过程来访问这些存储系统自己的存储过程，而不能使用`CALL`直接访问。

示例

```
CALL test(123, 'apple'); --- 调用存储过程并传参（参数位置）
CALL test(name => 'apple', id => 123); --- 调用存储过程并传参（命名参数）
CALL catalog.schema.test(); --- 调用权限定名称的存储过程
```

8.6.5 COMMIT

概要

```
COMMIT [WORK]
```

描述

提交当前事务。

示例

```
COMMIT;
COMMIT WORK;
```

8.6.6 CREATE SCHEMA

概要

```
CREATE SCHEMA [ IF NOT EXISTS ] schema_name
```

```
[ WITH ( property_name = expression [, ...] ) ]
```

描述

创建新的SCHEMA。Schema 是管理数据库表、视图和其他数据库对象的容器。

- 如果 SCHEMA 已经存在，使用IF NOT EXISTS子句可以避免抛错；
- 使用WITH子句可以在创建时为 SCHEMA 设置属性。可以通过下列查询语句获取所有支持的属性列表：

```
SELECT * FROM system.metadata.schema_properties;
```

示例

```
CREATE SCHEMA web;
CREATE SCHEMA hive.sales;
CREATE SCHEMA IF NOT EXISTS traffic;
```

8.6.7 CREATE TABLE

概要

```
CREATE TABLE [ IF NOT EXISTS ]
table_name (
  { column_name data_type [ COMMENT comment ]
  | LIKE existing_table_name [ { INCLUDING | EXCLUDING } PROPERTIES
 ] }
  [, ...]
)
[ COMMENT table_comment ]
[ WITH ( property_name = expression [, ...] ) ]
```

描述

创建空表。可以使用CREATE TABLE AS从已有数据集中创建表。

- 使用IF NOT EXISTS子句可以避免在表存在时抛出异常；
- 使用WITH子句可以在创建表格时，为表格设置属性。支持的属性列表可以通过下列语句获取：

  ```
  SELECT * FROM system.metadata.table_properties;
  ```

- 使用LIKE子句可以引用已经存在的表的列定义。支持使用多个LIKE子句；
- 使用INCLUDING PROPERTIES可以在创建表时，引用已有表的属性。如果同时还使用了WITH子句，WITH子句中指定的属性会覆盖INCLUDING PROPERTIES引入的同名属性。默认为EXCLUDING PROPERTIES。

示例

```
--- 创建表orders
CREATE TABLE orders (
```

```
  orderkey bigint,
  orderstatus varchar,
  totalprice double,
  orderdate date
)
WITH (format = 'ORC')

--- 创建表orders，带备注信息
CREATE TABLE IF NOT EXISTS orders (
  orderkey bigint,
  orderstatus varchar,
  totalprice double COMMENT 'Price in cents.',
  orderdate date
)
COMMENT 'A table to keep track of orders.'

--- 创建表bigger_orders， 其中部分列的定义引用自表orders
CREATE TABLE bigger_orders (
  another_orderkey bigint,
  LIKE orders,
  another_orderdate date
)
```

8.6.8 CREATE TABLE AS

概要

```
CREATE TABLE [ IF NOT EXISTS ] table_name [ ( column_alias, ... ) ]
[ COMMENT table_comment ]
[ WITH ( property_name = expression [, ...] ) ]
AS query
[ WITH [ NO ] DATA ]
```

描述

创建新表，表中的数据来自`SELECT`子句。

- 使用`IF NOT EXISTS`子句可以避免在表存在时抛出异常；
- 使用`WITH`子句可以在创建表格时，为表格设置属性。支持的属性列表可以通过下列语句获取：

  `SELECT * FROM system.metadata.table_properties;`

示例

```
--- 从表orders中选择两个列来创建新的表
CREATE TABLE orders_column_aliased (order_date, total_price)
AS
SELECT orderdate, totalprice
FROM orders

--- 创建新表，使用聚合函数
CREATE TABLE orders_by_date
COMMENT 'Summary of orders by date'
WITH (format = 'ORC')
AS
SELECT orderdate, sum(totalprice) AS price
FROM orders
GROUP BY orderdate
```

```
--- 创建新表，使用`IF NOT EXISTS`子句
CREATE TABLE IF NOT EXISTS orders_by_date AS
SELECT orderdate, sum(totalprice) AS price
FROM orders
GROUP BY orderdate

--- 创建新表，schema和表nation一样，但是没有数据
CREATE TABLE empty_nation AS
SELECT *
FROM nation
WITH NO DATA
```

8.6.9 CREATE VIEW

概要

```
CREATE [ OR REPLACE ] VIEW view_name AS query
```

描述

创建视图。视图是一个逻辑表，不包含实际数据，可以在查询中被引用。每次查询引用视图时，定义视图的查询语句都会被执行一次。

创建视图时使用`OR REPLACE`可以避免在视图存在时抛出异常。

示例

```
--- 创建一个简单的视图
CREATE VIEW test AS
SELECT orderkey, orderstatus, totalprice / 2 AS half
FROM orders
--- 创建视图，使用聚合函数
CREATE VIEW orders_by_date AS
SELECT orderdate, sum(totalprice) AS price
FROM orders
GROUP BY orderdate
--- 创建视图，如果视图已经存在，就替换它
CREATE OR REPLACE VIEW test AS
SELECT orderkey, orderstatus, totalprice / 4 AS quarter
FROM orders
```

8.6.10 DEALLOCATE PREPARE

概要

```
DEALLOCATE PREPARE statement_name
```

描述

从会话中删除一个命名 Statement。

示例

```
--- 释放名为my_query的查询
```

```
DEALLOCATE PREPARE my_query;
```

8.6.11 DELETE

概要

```
DELETE FROM table_name [ WHERE condition ]
```

描述

删除表中与 WHERE 子句匹配的行，如果没有指定 WHERE 子句，将删除所有行。

示例

```
--- 删除匹配行
DELETE FROM lineitem WHERE shipmode = 'AIR';

--- 删除匹配行
DELETE FROM lineitem
WHERE orderkey IN (SELECT orderkey FROM orders WHERE priority = 'LOW
');

--- 清空表
DELETE FROM orders;
```

局限

部分连接器可能不支持`DELETE`。

8.6.12 DESCRIBE

概要

```
DESCRIBE table_name
```

描述

获取表格定义信息,等同与 SHOW COLUMNS。

示例

```
DESCRIBE orders;
```

8.6.13 DESCRIBE INPUT

概要

```
DESCRIBE INPUT statement_name
```

描述

列出一个预编译查询中各个参数的位置及其类型。

示例

```
--- 创建一个预编译查询'my_select1'
PREPARE my_select1 FROM
SELECT ? FROM nation WHERE regionkey = ? AND name < ?;

--- 获取该预编译查询的描述信息
DESCRIBE INPUT my_select1;
```

查询结果：

```
Position | Type
--------------------
        0 | unknown
        1 | bigint
        2 | varchar
(3 rows)
```

8.6.14 DESCRIBE OUTPUT

概要

```
DESCRIBE OUTPUT statement_name
```

描述

列出输出结果的所有列信息，包括列名（或别名）、Catalog、Schema、表名、类型、类型大小（单位字节）和一个标识位，说明该列是否为别名。

示例

示例1

准备一个预编译查询：

```
PREPARE my_select1 FROM
SELECT * FROM nation;
```

执行`DESCRIBE OUTPUT`，输出：

```
DESCRIBE OUTPUT my_select1;

 Column Name | Catalog | Schema | Table  |  Type   | Type Size |
Aliased
-------------+---------+--------+--------+---------+-----------
+---------
 nationkey   | tpch    | sf1    | nation | bigint  |         8 | false
 name        | tpch    | sf1    | nation | varchar |         0 | false
 regionkey   | tpch    | sf1    | nation | bigint  |         8 | false
 comment     | tpch    | sf1    | nation | varchar |         0 | false
(4 rows)
```

示例2

```
PREPARE my_select2 FROM
```

```
SELECT count(*) as my_count, 1+2 FROM nation
```

执行DESCRIBE OUTPUT，输出：

```
DESCRIBE OUTPUT my_select2;

 Column Name | Catalog | Schema | Table |  Type   | Type Size | Aliased
-------------+---------+--------+-------+---------+-----------
+---------
 my_count    |         |        |       | bigint |         8 | true
 _col1       |         |        |       | bigint |         8 | false
(2 rows)
```

示例3

```
PREPARE my_create FROM
CREATE TABLE foo AS SELECT * FROM nation;
```

执行DESCRIBE OUTPUT，输出：

```
DESCRIBE OUTPUT my_create;

 Column Name | Catalog | Schema | Table |  Type   | Type Size | Aliased
-------------+---------+--------+-------+---------+-----------
+---------
 rows        |         |        |       | bigint |         8 | false
(1 row)
```

8.6.15 DROP SCHEMA

概要

```
DROP SCHEMA [ IF EXISTS ] schema_name
```

描述

删除 Schema。

- Schema 必须为空；
- 使用IF EXISTS来避免删除不存在的 Schema 时报错。

示例

```
DROP SCHEMA web;
```

```
DROP TABLE IF EXISTS sales;
```

8.6.16 DROP TABLE

概要

```
DROP TABLE  [ IF EXISTS ] table_name
```

描述

删除数据表。使用 IF EXISTS 来避免删除不存在的表时报错。

示例

```
DROP TABLE orders_by_date;
DROP TABLE IF EXISTS orders_by_date;
```

8.6.17 DROP VIEW

概要

```
DROP VIEW  [ IF EXISTS ] view_name
```

描述

删除数据表。使用 IF EXISTS 来避免删除不存在的表时报错。

示例

```
DROP VIEW orders_by_date;
DROP VIEW IF EXISTS orders_by_date;
```

8.6.18 EXECUTE

概要

```
EXECUTE statement_name [ USING parameter1 [ , parameter2, ... ] ]
```

描述

执行预编译查询， 使用`USING`**子句按位置传参。**

示例

- **示例 1：**

```
PREPARE my_select1 FROM
SELECT name FROM nation;
--- 执行预编译查询
EXECUTE my_select1;
```

```
示例2
```

- 示例 2：

```
PREPARE my_select2 FROM
SELECT name FROM nation WHERE regionkey = ? and nationkey < ?;
--- 执行预编译查询
EXECUTE my_select2 USING 1, 3;
--- 上述语句等价与执行如下查询：
SELECT name FROM nation WHERE regionkey = 1 AND nationkey < 3;
```

8.6.19 EXPLAIN

概要

```
EXPLAIN [ ( option [, ...] ) ] statement
```

这里Option可以时如下几个：

```
    FORMAT { TEXT | GRAPHVIZ }
    TYPE { LOGICAL | DISTRIBUTED | VALIDATE }
```

描述

根据选项不同，可以实现如下几个功能：

- 显示查询语句的逻辑计划
- 显示查询语句的分布式执行计划
- 验证一个查询语句的合法性

使用`TYPE DISTRIBUTED`选项来显示计划分片。每个计划分片都在单个或多个 Presto 节点上执行。分片之间的间隔表示 Presto 节点间的数据交换。分片类型说明分片是如何被 Presto 节点执行的，以及数据是如何在分片间分布的。 分片类型如下：

- `SINGLE`， 分片在单个节点上执行；
- `HASH`，分片在固定个数的节点上执行，输入数据在这些节点上通过 Hash 函数分布；
- `ROUND_ROBIN`，分片在固定个数的节点上执行，输入数据在这些节点上通过`ROUND_ROBIN`的模式分布；
- `BROADCAST`，分片在固定个数的节点上执行，处理数据通过广播的方式分发到所有节点；
- `SOURCE`，分片在存储数据的节点上执行。

示例

- 示例 1：

```
PREPARE my_select1 FROM
SELECT name FROM nation;
--- 执行预编译查询
EXECUTE my_select1;
```

```
示例2
```

- 示例 2：

```
PREPARE my_select2 FROM
SELECT name FROM nation WHERE regionkey = ? and nationkey < ?;
--- 执行预编译查询
EXECUTE my_select2 USING 1, 3;
--- 上述语句等价与执行如下查询：
SELECT name FROM nation WHERE regionkey = 1 AND nationkey < 3;
```

8.6.20 EXPLAIN ANALYZE

概要

```
EXPLAIN ANALYZE [VERBOSE] statement
```

描述

执行 Statement，并且显示该次查询实际的分布式执行计划以及每个执行过程的成本。启用 VERBOSE选项可以获取更多详细信息和底层的统计量。

示例

在下面的示例中，你可以看到每个 Stage 消耗的 CPU 时间和该 Stage 中每个计划节点的相对成本。需要注意的是，相对成本使用实际时间来计量，因此，无法显示其与 CPU 时间的相关性。对于每个计划节点，还会显示一些附加的统计信息，这些统计信息有助于发现一次查询中的数据异常情况（如倾斜，Hash 冲突等）。

```
presto:sf1> EXPLAIN ANALYZE SELECT count(*), clerk FROM orders WHERE
orderdate > date '1995-01-01' GROUP BY clerk;

                                          Query Plan
------------------------------------------------------------------------------
Fragment 1 [HASH]
    Cost: CPU 88.57ms, Input: 4000 rows (148.44kB), Output: 1000 rows
 (28.32kB)
    Output layout: [count, clerk]
    Output partitioning: SINGLE []
    - Project[] => [count:bigint, clerk:varchar(15)]
            Cost: 26.24%, Input: 1000 rows (37.11kB), Output: 1000
rows (28.32kB), Filtered: 0.00%
            Input avg.: 62.50 lines, Input std.dev.: 14.77%
          - Aggregate(FINAL)[clerk][$hashvalue] => [clerk:varchar(15), $
hashvalue:bigint, count:bigint]
                Cost: 16.83%, Output: 1000 rows (37.11kB)
                Input avg.: 250.00 lines, Input std.dev.: 14.77%
                count := "count"("count_8")
            - LocalExchange[HASH][$hashvalue] ("clerk") => clerk:
varchar(15), count_8:bigint, $hashvalue:bigint
                    Cost: 47.28%, Output: 4000 rows (148.44kB)
                    Input avg.: 4000.00 lines, Input std.dev.: 0.00%
                - RemoteSource[2] => [clerk:varchar(15), count_8:
bigint, $hashvalue_9:bigint]
                        Cost: 9.65%, Output: 4000 rows (148.44kB)
```

```
                          Input avg.: 4000.00 lines, Input std.dev.: 0.
00%

Fragment 2 [tpch:orders:1500000]
    Cost: CPU 14.00s, Input: 818058 rows (22.62MB), Output: 4000 rows
 (148.44kB)
    Output layout: [clerk, count_8, $hashvalue_10]
    Output partitioning: HASH [clerk][$hashvalue_10]
    - Aggregate(PARTIAL)[clerk][$hashvalue_10] => [clerk:varchar(15),
 $hashvalue_10:bigint, count_8:bigint]
            Cost: 4.47%, Output: 4000 rows (148.44kB)
            Input avg.: 204514.50 lines, Input std.dev.: 0.05%
            Collisions avg.: 5701.28 (17569.93% est.), Collisions std.
dev.: 1.12%
            count_8 := "count"(*)
        - ScanFilterProject[table = tpch:tpch:orders:sf1.0, originalCo
nstraint = ("orderdate" > "$literal$date"(BIGINT '9131')), filterPred
icate = ("orderdate" > "$literal$date"(BIGINT '9131'))] => [cler
                Cost: 95.53%, Input: 1500000 rows (0B), Output: 818058
 rows (22.62MB), Filtered: 45.46%
                Input avg.: 375000.00 lines, Input std.dev.: 0.00%
                $hashvalue_10 := "combine_hash"(BIGINT '0', COALESCE
("$operator$hash_code"("clerk"), 0))
                orderdate := tpch:orderdate
                clerk := tpch:clerk
```

如果开启了`VERBOSE`**选项，运行操作符会返回更多信息：**

```
EXPLAIN ANALYZE VERBOSE SELECT count(clerk) OVER() FROM orders WHERE
orderdate > date '1995-01-01';

                                          Query Plan
-------------------------------------------------------------------------------
  ...
        - Window[] => [clerk:varchar(15), count:bigint]
                Cost: {rows: ?, bytes: ?}
                CPU fraction: 75.93%, Output: 8130 rows (230.24kB)
                Input avg.: 8130.00 lines, Input std.dev.: 0.00%
                Active Drivers: [ 1 / 1 ]
                Index size: std.dev.: 0.00 bytes , 0.00 rows
                Index count per driver: std.dev.: 0.00
                Rows per driver: std.dev.: 0.00
                Size of partition: std.dev.: 0.00
                count := count("clerk")
 ...
```

8.6.21 GRANT

概要

```
GRANT ( privilege [, ...] | ( ALL PRIVILEGES ) )
ON [ TABLE ] table_name TO ( grantee | PUBLIC )
[ WITH GRANT OPTION ]
```

描述

授予指定的权限指定的受让对象。

- `ALL PRIVILEGES` **包括 DELETE， INSERT 和 SELECT 权限。**

- PUBLIC表示所有用户。
- 使用WITH GRANT OPTION允许权限受让对象给其他用户赋予相同的权限。

示例

```
GRANT INSERT, SELECT ON orders TO alice; --- 给用户alice赋权
GRANT SELECT ON nation TO alice WITH GRANT OPTION; --- 给用户alice赋
权，同时，alice还具有赋予其他用户`SELECT`权限的权限，
GRANT SELECT ON orders TO PUBLIC; --- 开放表order的`SELECT`权限给所有人
```

局限

某些连接器可能不支持GRANT。

8.6.22 INSERT

概要

```
INSERT INTO table_name [ ( column [, ... ] ) ] query
```

描述

插入数据。如果指定了列名列表,该列表必须精确匹配query的结果集。没有出现在列名列表中的列将使用null进行填充。

示例

```
INSERT INTO orders SELECT * FROM new_orders; --- 将select结果插入表
orders中
INSERT INTO cities VALUES (1, 'San Francisco'); --- 插入一行数据
INSERT INTO cities VALUES (2, 'San Jose'), (3, 'Oakland'); --- 插入多行
数据
INSERT INTO nation (nationkey, name, regionkey, comment) VALUES (26, '
POLAND', 3, 'no comment'); --- 插入一行数据
INSERT INTO nation (nationkey, name, regionkey) VALUES (26, 'POLAND',
3); --- 插入一行数据（只包括部分列）
```

8.6.23 PREPARE

概要

```
PREPARE statement_name FROM statement
```

描述

创建一个预处理语句，以备后续使用。预处理语句就是一组保存在会话中的命名查询语句集合，可以在这些语句中定义参数，在实际执行时填充实际的值。定义的参数使用?占位。

示例

```
--- 不带参数的预处理查询语句
```

```
PREPARE my_select1 FROM
SELECT * FROM nation;

--- 带参数的预处理查询语句
PREPARE my_select2 FROM
SELECT name FROM nation WHERE regionkey = ? AND nationkey < ?;

--- 不带参数的预处理插入语句
PREPARE my_insert FROM
INSERT INTO cities VALUES (1, 'San Francisco');
```

8.6.24 RESET SESSION

概要

```
RESET SESSION name
RESET SESSION catalog.name
```

描述

重置会话，使用默认属性。

示例

```
RESET SESSION optimize_hash_generation;
RESET SESSION hive.optimized_reader_enabled;
```

8.6.25 REVOKE

概要

```
REVOKE [ GRANT OPTION FOR ]
( privilege [, ...] | ALL PRIVILEGES )
ON [ TABLE ] table_name FROM ( grantee | PUBLIC )
```

描述

撤回指定用户的指定权限。

- 使用ALL PRIVILEGE可以撤销SELECT,INSERT和DELETE权限；
- 使用PUBLIC表示撤销对象为PUBLIC角色，用户通过其他角色获得的权限不会被收回；
- 使用GRANT OPTION FOR将进一步收回用户使用GRANT进行赋权的权限；
- 语句中grantee既可以是单个用户也可以是角色。

示例

```
--- 撤回用户alice对表orders进行INSET和SELECT的权限
REVOKE INSERT, SELECT ON orders FROM alice;

--- 撤回所有人对表nation进行SELECT的权限，
--- 同时，撤回所有人赋予其他人对该表进行SELECT操作权限的权限
REVOKE GRANT OPTION FOR SELECT ON nation FROM PUBLIC;
```

```
--- 撤回用户alice在表test上的所有权限
REVOKE ALL PRIVILEGES ON test FROM alice;
```

局限

有些连接器不支持REVOKE语句。

8.6.26 ROLLBACK

概要

```
ROLLBACK [ WORK ]
```

描述

回滚当前事物。

示例

```
ROLLBACK;
ROLLBACK WORK;
```

8.6.27 SELECT 语句

8.6.28 SET SESSION

概要

```
SET SESSION name = expression
SET SESSION catalog.name = expression
```

描述

设置会话属性。

示例

```
SET SESSION optimize_hash_generation = true;
SET SESSION hive.optimized_reader_enabled = true;
```

8.6.29 SHOW CATALOGS

概要

```
SHOW CATALOGS [ LIKE pattern ]
```

描述

获取可用的 Catalog 清单。可以使用 LIKE 子句过滤 Catalog 名称。

示例

```
SHOW CATALOGS;
```

8.6.30 SHOW COLUMNS

概要

```
SHOW COLUMNS FROM table
```

描述

获取给定表格所有的列及其属性。

示例

```
SHOW COLUMNS FROM orders;
```

8.6.31 SHOW CREATE TABLE

概要

```
SHOW CREATE TABLE table_name
```

描述

显示定义给定表格的 SQL 语句。

示例

```
SHOW CREATE TABLE sf1.orders;

-----------------------------------------
 CREATE TABLE tpch.sf1.orders (
    orderkey bigint,
    orderstatus varchar,
    totalprice double,
    orderdate varchar
 )
 WITH (
    format = 'ORC',
    partitioned_by = ARRAY['orderdate']
 )
```

```
(1 row)
```

8.6.32 SHOW CREATE VIEW

概要

```
SHOW CREATE VIEW view_name
```

描述

显示定义给定视图的 SQL 语句。

示例

```
SHOW CREATE VIEW view1;
```

8.6.33 SHOW FUNCTIONS

概要

```
SHOW FUNCTIONS
```

描述

列出所有可用于查询的函数。

示例

```
SHOW FUNCTIONS
```

8.6.34 SHOW GRANTS

概要

```
SHOW GRANTS [ ON [ TABLE ] table_name ]
```

描述

显示权限列表。

示例

```
--- 获取当前用户在表orders上的权限
SHOW GRANTS ON TABLE orders;

--- 获取当前用户在当前catalog中的权限
SHOW GRANTS;
```

局限

有些连接器不支持`SHOW GRANTS`操作。

8.6.35 SHOW PARTITIONS

概要

```
SHOW PARTITIONS FROM table [ WHERE ... ] [ ORDER BY ... ] [ LIMIT
 ... ]
```

描述

获取一张表的分区列表。可以使用`WHERE`子句进行条件过滤，使用`ORDER BY`进行排序，使用LIMIT来限制结果集大小。这些子句的使用方式和`SELECT`无异。

示例

```
--- 获取表orders的所有分区列表
SHOW PARTITIONS FROM orders;

--- 获取表orders从2013年至今的所有分区列表，并按照年份排序
SHOW PARTITIONS FROM orders WHERE ds >= '2013-01-01' ORDER BY ds DESC;

--- 获取表order最近的几个分区
SHOW PARTITIONS FROM orders ORDER BY ds DESC LIMIT 10;
```

8.6.36 SHOW SCHEMAS

概要

```
SHOW SCHEMAS [ FROM catalog ] [ LIKE pattern ]
```

描述

列出给定 Catalog 下的所有 Schema，Catalog 不指定则表示当前 Catalog。使用`LIKE`子句可以过滤 Schema 名称。

示例

```
SHOW SCHEMAS;
```

8.6.37 SHOW SESSION

概要

```
SHOW SESSION
```

描述

显示当前回话的属性列表。

示例

```
SHOW SESSION
```

8.6.38 SHOW TABLES

概要

```
SHOW TABLES [ FROM schema ] [ LIKE pattern ]
```

描述

列出给定Schema下的所有表格，Schema 不指定则表示当前 Schema。使用`LIKE`子句可以过滤表名。

示例

```
SHOW TABLES;
```

8.6.39 START TRANSACTION

概要

```
START TRANSACTION [ mode [, ...] ]
```

其中`mode`可以从如下几个选项中选择：

```
ISOLATION LEVEL { READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ
 | SERIALIZABLE }
READ { ONLY | WRITE }
```

描述

在当前会话中启动一个新的事务。

示例

```
START TRANSACTION;
START TRANSACTION ISOLATION LEVEL REPEATABLE READ;
START TRANSACTION READ WRITE;
START TRANSACTION ISOLATION LEVEL READ COMMITTED, READ ONLY;
START TRANSACTION READ WRITE, ISOLATION LEVEL SERIALIZABLE;
```

8.6.40 USE

概要

```
USE catalog.schema
```

```
USE schema
```

描述

更新当前回话，使用指定的 Catalog 和 Schema。如果 Catalog 没有指定，则使用当前 Catalog 下的 Schema。

示例

```
USE hive.finance;
USE information_schema;
```

8.6.41 VALUES

概要

```
VALUES row [, ...]
```

其中， `row`是一个表达式，或者如下形式的表达式列表：

```
( column_expression [, ...] )
```

描述

定义一张字面内联表。

- 任何可以使用查询语句的地方都可以使用VALUE， 如放在SELECT语句的FROM后面，放在INSERT里，甚至放在最顶层；
- VALUE默认创建一张匿名表，并且没有列名。表名和列名可以通过AS进行命名。

示例

```
--- 返回一个表对象，包含1列，3行数据
VALUES 1, 2, 3

--- 返回一个表对象，包含2列，3行数据
VALUES
    (1, 'a'),
    (2, 'b'),
    (3, 'c')

--- 在查询语句中使用
SELECT * FROM (
    VALUES
        (1, 'a'),
        (2, 'b'),
        (3, 'c')
) AS t (id, name)

--- 创建一个表
CREATE TABLE example AS
SELECT * FROM (
    VALUES
        (1, 'a'),
        (2, 'b'),
```

```
        (3, 'c')
) AS t (id, name)
```

8.6.42 SELECT

概要

```
[ WITH with_query [, ...] ]
SELECT [ ALL | DISTINCT ] select_expr [, ...]
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ GROUP BY [ ALL | DISTINCT ] grouping_element [, ...] ]
[ HAVING condition]
[ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
[ ORDER BY expression [ ASC | DESC ] [, ...] ]
[ LIMIT [ count | ALL ] ]
```

其中，`from_item`有如下两种形式：

```
table_name [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
```

```
from_item join_type from_item [ ON join_condition | USING ( join_colum
n [, ...] ) ]
```

这里的`join_type`可以是如下之一：

- **[INNER] JOIN**
- **LEFT [OUTER] JOIN**
- **RIGHT [OUTER] JOIN**
- **FULL [OUTER] JOIN**
- **CROSS JOIN**

而`grouping_element`可以是如下之一：

- **()**
- **expression**
- **GROUPING SETS ((column [, ...]) [, ...])**
- **CUBE (column [, ...])**
- **ROLLUP (column [, ...])**

描述

检索 0 到多张数据表，获取结果集。详细内容请参考如下小节：

- **WITH 子句**
- **GROUP BY 子句**
- **HIVING 子句**

- UNION|INTERSECT|EXCEPT 子句
- ORDER BY子句
- LIMIT 子句
- TABLESAMPLE
- UNNEST
- Joins
- 子查询

8.6.43 WITH 子句

基本功能

`WITH`子句可以定义一组命名关系，这些关系可以在查询中被使用，从而扁平化嵌套查询或简化子查询。如下两个查询语句是等价的：

```
--- 不使用WITH子句
SELECT a, b
FROM (
  SELECT a, MAX(b) AS b FROM t GROUP BY a
) AS x;

--- 使用WITH子句，查询语句看起来更加明了
WITH x AS (SELECT a, MAX(b) AS b FROM t GROUP BY a)
SELECT a, b FROM x;
```

定义多个子查询

`WITH`子句中可以定义多个子查询：

```
WITH
  t1 AS (SELECT a, MAX(b) AS b FROM x GROUP BY a),
  t2 AS (SELECT a, AVG(d) AS d FROM y GROUP BY a)
SELECT t1.*, t2.*
FROM t1
JOIN t2 ON t1.a = t2.a;
```

组成链式结构

`WITH`子句中的关系还可以组成链式结构：

```
WITH
  x AS (SELECT a FROM t),
  y AS (SELECT a AS b FROM x),
  z AS (SELECT b AS c FROM y)
```

```
SELECT c FROM z;
```

8.6.44 GROUP BY 子句

基本功能

使用GROUP BY子句可以对SELECT结果进行分组。GROUP BY子句支持任意的表达式，既可以使用列名，也可以使用序号（从 1 开始）。

下列示例中的查询语句是等价的（列nationkey的位置为 2）。

```
--- 使用列序号
SELECT count(*), nationkey FROM customer GROUP BY 2;

--- 使用列名
SELECT count(*), nationkey FROM customer GROUP BY nationkey;
```

没有在输出列表中指定的列也可以用于GROUP BY子句，如下所示：

```
--- 列mktsegment没有在SELECT列表中指定，
--- 结果集中不包括mktsegment列的内容。
SELECT count(*) FROM customer GROUP BY mktsegment;

 _col0
-------
 29968
 30142
 30189
 29949
 29752
(5 rows)
```

需要注意的是，在SELECT语句中使用GROUP BY子句时，其输出表达式只能是聚合函数或者是GROUP BY子句中使用的列。

复杂分组操作

Presto 支持如下 3 中复杂的聚合语法，可以在一个查询中实现多个列集合的聚合分析：

- GROUPING SETS(分组集)
- CUBE(多维立方)
- ROLLUP(汇总)

Presto 的复杂分组操作只支持使用列名和序号，不支持表达式。

GROUPING SETS

GROUPING SETS可以在一条查询语句中完成多个列的分组聚合。没有在分组列表中的列使用NULL进行填充。

表 shipping 是一个包含 5 个列的数据表，如下所示：

```
SELECT * FROM shipping;

 origin_state | origin_zip | destination_state | destination_zip |
package_weight
--------------+------------+-------------------+-----------------
+----------------
 California   |      94131 | New Jersey        |            8648 |
         13
 California   |      94131 | New Jersey        |            8540 |
         42
 New Jersey   |       7081 | Connecticut       |            6708 |
        225
 California   |      90210 | Connecticut       |            6927 |
       1337
 California   |      94131 | Colorado          |           80302 |
          5
 New York     |      10002 | New Jersey        |            8540 |
          3
(6 rows)
```

现在希望在一个查询中获取如下几个分组结果：

- 按 origin_state 进行分组，获取 package_weight 的总和；
- 按 origin_state 和 origin_zip 分组，获取 package_weight 的总和；
- 按 destination_state 分组，获取 package_weight 的总和。

使用`GROUPING SETS`可以在一条语句中获取上述 3 个分组的结果集，如下所示：

```
SELECT origin_state, origin_zip, destination_state, sum(package_weight
)
FROM shipping
GROUP BY GROUPING SETS (
    (origin_state),
    (origin_state, origin_zip),
    (destination_state));

 origin_state | origin_zip | destination_state | _col0
--------------+------------+-------------------+-------
 New Jersey   | NULL       | NULL              |   225
 California   | NULL       | NULL              |  1397
 New York     | NULL       | NULL              |     3
 California   |      90210 | NULL              |  1337
 California   |      94131 | NULL              |    60
 New Jersey   |       7081 | NULL              |   225
 New York     |      10002 | NULL              |     3
 NULL         | NULL       | Colorado          |     5
 NULL         | NULL       | New Jersey        |    58
 NULL         | NULL       | Connecticut       |  1562
(10 rows)
```

上述查询逻辑也可以通过`UNION ALL`多个`GROUP BY`查询实现：

```
SELECT origin_state, NULL, NULL, sum(package_weight)
FROM shipping GROUP BY origin_state

UNION ALL
```

```
SELECT origin_state, origin_zip, NULL, sum(package_weight)
FROM shipping GROUP BY origin_state, origin_zip

UNION ALL

SELECT NULL, NULL, destination_state, sum(package_weight)
FROM shipping GROUP BY destination_state;
```

但是，使用GROUPING SETS语法往往能获得更好的性能，因为，该语法在执行时，只会读取一次基表数据，而使用上述UNION ALL方式，会读取3次，因此，如果在查询期间基表数据有变化，使用UNION ALL的方式容易出现不一致的结果。

CUBE

使用CUBE可以获得给定列列表所有可能的分组结果。如下所示：

```
SELECT origin_state, destination_state, sum(package_weight)
FROM shipping
GROUP BY CUBE (origin_state, destination_state);

 origin_state | destination_state | _col0
--------------+-------------------+-------
 California   | New Jersey        |    55
 California   | Colorado          |     5
 New York     | New Jersey        |     3
 New Jersey   | Connecticut       |   225
 California   | Connecticut       |  1337
 California   | NULL              |  1397
 New York     | NULL              |     3
 New Jersey   | NULL              |   225
 NULL         | New Jersey        |    58
 NULL         | Connecticut       |  1562
 NULL         | Colorado          |     5
 NULL         | NULL              |  1625
(12 rows)
```

该查询等价于如下语句：

```
SELECT origin_state, destination_state, sum(package_weight)
FROM shipping
GROUP BY GROUPING SETS (
    (origin_state, destination_state),
    (origin_state),
    (destination_state),
    ());
```

ROLLUP

使用ROLLUP可以获得给定列集和的小记结果。如下所示：

```
SELECT origin_state, origin_zip, sum(package_weight)
FROM shipping
GROUP BY ROLLUP (origin_state, origin_zip);

 origin_state | origin_zip | _col2
--------------+------------+-------
```

```
 California   |      94131 |    60
 California   |      90210 |  1337
 New Jersey   |       7081 |   225
 New York     |      10002 |     3
 California   | NULL       |  1397
 New York     | NULL       |     3
 New Jersey   | NULL       |   225
 NULL         | NULL       |  1625
(8 rows)
```

上述示例等价与如下语句：

```
SELECT origin_state, origin_zip, sum(package_weight)
FROM shipping
GROUP BY GROUPING SETS ((origin_state, origin_zip), (origin_state),
 ());
```

综合使用

下列 3 个语句是等价的：

```
SELECT origin_state, destination_state, origin_zip, sum(package_weight
)
FROM shipping
GROUP BY
    GROUPING SETS ((origin_state, destination_state)),
    ROLLUP (origin_zip);
```

```
SELECT origin_state, destination_state, origin_zip, sum(package_weight
)
FROM shipping
GROUP BY
    GROUPING SETS ((origin_state, destination_state)),
    GROUPING SETS ((origin_zip), ());
```

```
SELECT origin_state, destination_state, origin_zip, sum(package_weight
)
FROM shipping
GROUP BY GROUPING SETS (
    (origin_state, destination_state, origin_zip),
    (origin_state, destination_state));
```

输出结果如下：

```
 origin_state | destination_state | origin_zip | _col3
--------------+-------------------+------------+-------
 New York     | New Jersey        |      10002 |     3
 California   | New Jersey        |      94131 |    55
 New Jersey   | Connecticut       |       7081 |   225
 California   | Connecticut       |      90210 |  1337
 California   | Colorado          |      94131 |     5
 New York     | New Jersey        | NULL       |     3
 New Jersey   | Connecticut       | NULL       |   225
 California   | Colorado          | NULL       |     5
 California   | Connecticut       | NULL       |  1337
 California   | New Jersey        | NULL       |    55
```

```
(10 rows)
```

在GROUP BY子句中，可以使用ALL和DISTINCT修饰符，用来说明是否可以生成重复的统计维度。

如下所示：

```
SELECT origin_state, destination_state, origin_zip, sum(package_weight
)
FROM shipping
GROUP BY ALL
    CUBE (origin_state, destination_state),
    ROLLUP (origin_state, origin_zip);
```

等价于：

```
SELECT origin_state, destination_state, origin_zip, sum(package_weight
)
FROM shipping
GROUP BY GROUPING SETS (
    (origin_state, destination_state, origin_zip),
    (origin_state, origin_zip),
    (origin_state, destination_state, origin_zip),
    (origin_state, origin_zip),
    (origin_state, destination_state),
    (origin_state),
    (origin_state, destination_state),
    (origin_state),
    (origin_state, destination_state),
    (origin_state),
    (destination_state),
    ());
```

其中有多个维度是重复的。如果使用DISTINCT，则只会输出不重复的维度。

```
SELECT origin_state, destination_state, origin_zip, sum(package_weight
)
FROM shipping
GROUP BY DISTINCT
    CUBE (origin_state, destination_state),
    ROLLUP (origin_state, origin_zip);
```

等价于：

```
SELECT origin_state, destination_state, origin_zip, sum(package_weight
)
FROM shipping
GROUP BY GROUPING SETS (
    (origin_state, destination_state, origin_zip),
    (origin_state, origin_zip),
    (origin_state, destination_state),
    (origin_state),
    (destination_state),
    ());
```

说明：

GROUP BY 默认使用的修饰符为 ALL。

GROUPING 函数

Presto 提供了一个grouping函数，用于生成一个标记数，该数中的每一个比特位表示对应的列是否出现在该分组条件中。语法如下：

```
grouping(col1, ..., colN) -> bigint
```

grouping通常与GROUPING SETS,ROLLUP,CUBE或GROUP BY一起使用。grouping中的列必须与GROUPING SETS,ROLLUP,CUBE或GROUP BY中指定的列一一对应。

```
SELECT origin_state, origin_zip, destination_state, sum(package_weight
),
       grouping(origin_state, origin_zip, destination_state)
FROM shipping
GROUP BY GROUPING SETS (
        (origin_state),
        (origin_state, origin_zip),
        (destination_state));

origin_state | origin_zip | destination_state | _col3 | _col4
--------------+------------+-------------------+-------+-------
California   | NULL       | NULL              |  1397 |     3   ---
011
New Jersey   | NULL       | NULL              |   225 |     3   ---
011
New York     | NULL       | NULL              |     3 |     3   ---
011
California   |      94131 | NULL              |    60 |     1   ---
001
New Jersey   |       7081 | NULL              |   225 |     1   ---
001
California   |      90210 | NULL              |  1337 |     1   ---
001
New York     |      10002 | NULL              |     3 |     1   ---
001
NULL         | NULL       | New Jersey        |    58 |     6   ---
100
NULL         | NULL       | Connecticut       |  1562 |     6   ---
100
NULL         | NULL       | Colorado          |     5 |     6   ---
100
(10 rows)
```

如上所示，grouping函数返回的是右对齐的比特标记数，0表示列存在，1表示列不存在。

8.6.45 HAVING 子句

HAVING子句用来控制查询中分组的选择，与聚合函数和GROUP BY子句一起使用。HAVING子句的功能会在分组和聚合计算完成后进行，过滤掉不满足条件的分组。

下面的示例将账户结余大于 5700000 的用户选出来：

```
SELECT count(*), mktsegment, nationkey,
       CAST(sum(acctbal) AS bigint) AS totalbal
FROM customer
GROUP BY mktsegment, nationkey
```

```
HAVING sum(acctbal) > 5700000
ORDER BY totalbal DESC;
```

输出如下：

```
_col0 | mktsegment | nationkey | totalbal
-------+------------+-----------+----------
  1272 | AUTOMOBILE |        19 |  5856939
  1253 | FURNITURE  |        14 |  5794887
  1248 | FURNITURE  |         9 |  5784628
  1243 | FURNITURE  |        12 |  5757371
  1231 | HOUSEHOLD  |         3 |  5753216
  1251 | MACHINERY  |         2 |  5719140
  1247 | FURNITURE  |         8 |  5701952
(7 rows)
```

8.6.46 集合运算

概述

Presto 支持UNION、INTERSECT和EXCEPT 3 种集合运算。这些子句用来将多个查询语句的结果合并成一个总的结果集。使用方法如下所示：

```
query UNION [ALL | DISTINCT] query
query INTERSECT [DISTINCT] query
query EXCEPT [DISTINCT] query
```

参数ALL和DISTINCT来控制最终哪些行会出现在结果集中，默认设为DISTINCT。

- ALL，有可能返回重复的行；
- DISTINCT，对结果集进行去重。

INTERSECT和EXCEPT不支持ALL选项。

上述3个集合运算可以组合使用，运算从左到右进行，INTERSECT的优先级最高，因此，组合运算A UNION B INTERSECT C EXCEPT D实际的顺序是A UNION (B INTERSECT C) EXCEPT D.

UNION

UNION对两个查询的结果集做并集运算，通过ALL和DISTINCT来控制是否剔除重复项。

示例 1：

```
SELECT 13
UNION
SELECT 42;

 _col0
-------
    13
    42
```

```
(2 rows)
```

示例 2：

```
SELECT 13
UNION
SELECT * FROM (VALUES 42, 13);

 _col0
-------
    13
    42
(2 rows)
```

示例 3：

```
SELECT 13
UNION ALL
SELECT * FROM (VALUES 42, 13);

 _col0
-------
    13
    42
    13
(3 rows)
```

INTERSECT

`INTERSECT`对两个查询的结果集做交集运算。

示例：

```
SELECT * FROM (VALUES 13, 42)
INTERSECT
SELECT 13;

 _col0
-------
   13
(1 rows)
```

EXCEPT

`EXCEPT`求两个查询结果集的补集。

```
SELECT * FROM (VALUES 13, 42)
EXCEPT
SELECT 13;

 _col0
-------
   42
```

```
(1 rows)
```

8.6.47 ORDER BY 子句

在查询中可以使用ORDER BY子句对查询结果进行排序。语法如下：

```
ORDER BY expression [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...]
```

其中：

- expression由列名或列位置序号（从 1 开始）组成；
- ORDER BY在GROUP BY和HAVING之后执行；
- NULLS { FIRST | LAST }可以控制NULL值的排序方式（与ASC/DESC无关），默认为LAST。

8.6.48 LIMIT 子句

LIMIT子句用来控制结果集的规模（即行数）。使用LIMIT ALL等价于不使用LIMIT子句。

示例：

```
--- 本例中，因为没有使用ORDER BY子句，返回结果是随机的
SELECT orderdate FROM orders LIMIT 5;

 orderdate
-------------
 1996-04-14
 1992-01-15
 1995-02-01
 1995-11-12
 1992-04-26
(5 rows)
```

8.6.49 TABLESAMPLE

Presto 提供了两种采样方式，BERNOULLI和SYSTEM，但是无论哪种采样方式， 都无法确定的给出采样结果集的行数。

BERNOULLI

本方法基于样本百分比概率进行采样。当使用 Bernoulli 方法对表格进行采样时，执行器会扫描表格的所有物理块，并且基于样本百分比与运行时计算的随机值之间的比较结果跳过某些行。

每一行被选中对概率是独立对，与其他行无关，但是，这不会减少从磁盘读取采样表所需的时间。如果还需对采样结果做进一步处理，可能会对总查询时间产生影响。

SYSTEM

本方法将表格分成数据的逻辑段，并以此粒度对表格进行采样。这种抽样方法要么从一个特定的数据段中选择所有的行，要么跳过它（根据样本百分比和在运行时计算的一个随机值之间的比较）。

采样结果与使用哪种连接器有关。例如，与 Hive 一起使用时，取决于数据在 HDFS 上的分布。这种方法不保证独立的抽样概率。

示例

```
--- 使用BERNOULLI采样
SELECT *
FROM users TABLESAMPLE BERNOULLI (50);

--- 使用SYSTEM采样
SELECT *
FROM users TABLESAMPLE SYSTEM (75);
Using sampling with joins:

--- 采样过程中使用JOIN
SELECT o.*, i.*
FROM orders o TABLESAMPLE SYSTEM (10)
JOIN lineitem i TABLESAMPLE BERNOULLI (40)
  ON o.orderkey = i.orderkey;
```

8.6.50 UNNEST

UNNEST可以将ARRAY和MAP类型的变量展开成表。其中ARRAY展开为单列的表，MAP展开为双列的表（键，值）。UNNEST可以一次展开多个ARRAY和MAP类型的变量，在这种情况下，它们被扩展为多个列，行数等于输入参数列表中最大展开行数（其他列用空值填充）。UNNEST可以有一个WITH ORDINALITY子句，此时，查询结果中会附加一个序数列。UNNEST通常与JOIN一起使用，并可以引用join左侧的关系的列。

示例 1：

```
--- 使用一个列
SELECT student, score
FROM tests
CROSS JOIN UNNEST(scores) AS t (score);
```

示例 2：

```
--- 使用多个列
SELECT numbers, animals, n, a
FROM (
  VALUES
    (ARRAY[2, 5], ARRAY['dog', 'cat', 'bird']),
    (ARRAY[7, 8, 9], ARRAY['cow', 'pig'])
) AS x (numbers, animals)
CROSS JOIN UNNEST(numbers, animals) AS t (n, a);

  numbers  |     animals      |  n   |  a
-----------+------------------+------+------
```

```
 [2, 5]     | [dog, cat, bird] |     2 | dog
 [2, 5]     | [dog, cat, bird] |     5 | cat
 [2, 5]     | [dog, cat, bird] |  NULL | bird
 [7, 8, 9]  | [cow, pig]       |     7 | cow
 [7, 8, 9]  | [cow, pig]       |     8 | pig
 [7, 8, 9]  | [cow, pig]       |     9 | NULL
(6 rows)
```

示例 3：

```
--- 使用 WITH ORDINALITY 子句
SELECT numbers, n, a
FROM (
  VALUES
    (ARRAY[2, 5]),
    (ARRAY[7, 8, 9])
) AS x (numbers)
CROSS JOIN UNNEST(numbers) WITH ORDINALITY AS t (n, a);

  numbers  | n | a
-----------+---+---
 [2, 5]    | 2 | 1
 [2, 5]    | 5 | 2
 [7, 8, 9] | 7 | 1
 [7, 8, 9] | 8 | 2
 [7, 8, 9] | 9 | 3
(5 rows)
```

8.6.51 Joins

Joins 可以将多个关系的数据合并到一起。`CROSS JOIN`返回两个关系到笛卡尔积（所有排列组合的集合）。`CROSS JOIN`可以通过如下两种方式来指定：

- **显式的使用`CROSS JOIN`语法；**
- **在`FROM`子句中指定多个关系；**

下列两个 SQL 语句是等价的：

```
--- 显式的使用`CROSS JOIN`语法
SELECT *
FROM nation
CROSS JOIN region;

--- 在`FROM`子句中指定多个关系
SELECT *
FROM nation, region;
```

示例：

表 nation 包括 25 行记录，表 region 包括 5 行，对这两个表执行 cross join 操作，其结果集为 125 行记录。

```
SELECT n.name AS nation, r.name AS region
FROM nation AS n
CROSS JOIN region AS r
ORDER BY 1, 2;
```

```
     nation     |   region
----------------+-------------
 ALGERIA        | AFRICA
 ALGERIA        | AMERICA
 ALGERIA        | ASIA
 ALGERIA        | EUROPE
 ALGERIA        | MIDDLE EAST
 ARGENTINA      | AFRICA
 ARGENTINA      | AMERICA
...
(125 rows)
```

如果参与 join 的表中包含相同名称的列，则需要用表名（或别名）加以修饰。

示例：

```
--- 正确
SELECT nation.name, region.name
FROM nation
CROSS JOIN region;

--- 正确
SELECT n.name, r.name
FROM nation AS n
CROSS JOIN region AS r;

--- 正确
SELECT n.name, r.name
FROM nation n
CROSS JOIN region r;

--- 错误，会抛出"Column 'name' is ambiguous"
SELECT name
FROM nation
CROSS JOIN region;
```

8.6.52 子查询

子查询是由一个查询组成的表达式。当子查询引用了子查询之外的列时，子查询与外部查询是相关的。Presto 对相关子查询的支持比较有限。

EXISTS

谓语`EXISTS`用于确定子查询是否返回所有行，如果子查询有返回值， WHERE 表达式为 TRUE，否则为 FALSE。

示例：

```
SELECT name
FROM nation
```

```
WHERE EXISTS (SELECT * FROM region WHERE region.regionkey = nation.
regionkey);
```

IN

如果WHERE指定的列在子查询结果集中存在，则返回结果，否则不返回。子查询只能返回一个列。

示例：

```
SELECT name
FROM nation
WHERE regionkey IN (SELECT regionkey FROM region);
```

标量子查询

标量子查询是一个非相关子查询，返回 0～1 行结果。该类子查询最多只能返回一行数据，如果子查询结果集为空，则返回null。

示例：

```
SELECT name
FROM nation
WHERE regionkey = (SELECT max(regionkey) FROM region)
```

8.7 常用函数和操作符

8.7.1 逻辑运算符

Presto 支持与、或、非3中逻辑操作符，并支持 NULL参与逻辑运算，如下所示：

```
SELECT CAST(null as boolean) AND true; --- null
SELECT CAST(null AS boolean) AND false; -- false
SELECT CAST(null AS boolean) AND CAST(null AS boolean); -- null
SELECT NOT CAST(null AS boolean); -- null
```

完整的运算真值表如下所示：

a	b	a AND b	a OR b
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	NULL	NULL	TRUE
FALSE	TRUE	FALSE	TRUE
FALSE	FALSE	FALSE	FALSE
FALSE	NULL	FALSE	NULL

a	b	a AND b	a OR b
NULL	TRUE	NULL	TRUE
NULL	FALSE	FALSE	NULL
NULL	FALSE	NULL	NULL

另外， NOT NULL的结果为NULL。

8.7.2 比较函数和运算符

比较操作符

Presto 支持的比较操作如下所示：

操作符	说明
<	小于
>	大于
<=	小于等于
>=	大于等于
=	等于
<>/!=	不等于
[NOT] BETWEEN	值 X [不]介于 min 和 max 之间
IS [NOT] NULL	判断是否为 NULL
IS [NOT] DISTINCT FROM	用于比较两个值是否一致。一般情况下，NULL作为未定义数据存在，任何有NULL 参与的比较都会返回NULL。但是IS [NOT] DISTINCT FROM将NULL作为值处理，返回TRUE或FALSE。

比较函数

Presto 提供了如下几个比较操作相关的函数：

- GREATEST

  返回最大值。

  示例：GREATEST(1, 2)

- LEAST

  返回最小值。

  示例：LEAST(1, 2)

比较相关的修饰谓语

Presto 还提供了几个比较相关的修饰谓语，可以增强比较语义的表达能力。使用方式如下：

```
<EXPRESSION><OPERATOR><QUANTIFIER> (<SUBQUERY>)
```

例如：

```
SELECT 'hello' = ANY (VALUES 'hello', 'world'); -- true
SELECT 21 < ALL (VALUES 19, 20, 21); -- false
SELECT 42 >= SOME (SELECT 41 UNION ALL SELECT 42 UNION ALL SELECT 43);
 -- true
```

其中，ANY,ALL,SOME就是比较修饰谓语。

- A = ALL (...) A 和所有值相等，则返回 TRUE
- A <> ALL (...) A 和所有值不相等，则返回 TRUE
- A < ALL (...) A 小于所有值，则返回 TRUE
- A = ANY (...) A 只要等于其中一个值，则返回 TRUE，等价与A IN (...)
- A <> ANY (...) A 只要和其中一个值不相等，则返回 TRUE，等价与A IN (...)
- A < ANY (...) A 只要小于其中一个值，则返回 TRUE

ANY和SOME含义相同，使用时可以互换。

8.7.3 条件表达式

条件表达是主要用于表达分支逻辑。Presto 支持如下几种条件表达式形式：

- CASE 表达式

  在标准SQL中，CASE表达式有两种不同的形式，如下所示：

  ```
  CASE expression
   WHEN <value|condition> THEN result
   [ WHEN ... ]
   [ ELSE result]
  ```

```
END
```

CASE语句将比较expression和value|condition中的值／条件，如果符合（值相等或条件匹配）则返回结果。

示例：

```
--- 比较值
SELECT a,
       CASE a
           WHEN 1 THEN 'one'
           WHEN 2 THEN 'two'
           ELSE 'many'
       END
```

```
--- 比较条件表达式
SELECT a, b,
       CASE
           WHEN a = 1 THEN 'aaa'
           WHEN b = 2 THEN 'bbb'
           ELSE 'ccc'
       END
```

- IF 函数

  IF函数是一个简单的比较函数，用于简化二值比较逻辑的写法。其表达形式如下：

  ```
  IF(condition, true_value, [false_value])
  ```

  如果condition返回TRUE，则函数返回true_value，否则返回 false_value。其中，false_value可选，不设置则返回NULL。

- COALESCE

  COALESCE函数返回参数列表中第一个不是NULL的参数。其表达形式如下：

  ```
  COALESCE(value1, value2[, ...])
  ```

- NULLIF

  NULLIF函数在value1与value2相等时，返回NULL,否则返回value1。函数使用方法如下：

  ```
  NULLIF(value1, value2)
  ```

- TRY

  TRY函数会捕获expression计算过程中抛出的异常，并返回NULL。TRY捕获的异常包括如下几类：

  - 除零异常，如x/0
  - 类型转换错误
  - 数值越界

  通常和COALESCE一起使用，以便在出错时，返回默认值。使用方法如下：

  ```
  TRY(expression)
  ```

  示例：

  ```
  --- COALESCE和TRY搭配使用，在packages=0，抛出除零异常时，返回默认值（0）。
  SELECT COALESCE(TRY(total_cost / packages), 0) AS per_package FROM
  shipping;

  per_package
  -------------
       4
         14
       0
         19
  (4 rows)
  ```

8.7.4 转换函数

Presto提供给了如下几个显式类型转换函数：

- CAST

  显式的进行类型转换，只是在出现转换错误的时候抛出异常。使用方法如下：

  ```
  CAST(value AS type) -> value1:type
  ```

- TRY_CAST

  与CAST功能相同，只是在出现转换错误的时候，返回NULL。使用方法如下：

  ```
  TRY_CAST(value AS TYPE) -> value1:TYPE | NULL
  ```

- TYPEOF

  获取参数或表达式值的类型字符串，使用方法如下：

  ```
  TYPEOF(expression) -> type:VARCHAR
  ```

  示例：

  ```
  SELECT TYPEOF(123); -- integer
  SELECT TYPEOF('cat'); -- varchar(3)
  SELECT TYPEOF(cos(2) + 1.5); -- double
  ```

8.7.5 数学函数与运算符

数学操作符

操作符	说明
+	加
-	减
*	乘
/	除（整数相除会截断）
%	取余

数学函数

Presto 提供了十分丰富的数学函数，如下表所示：

函数	语法	说明
abs	abs(x) →	x
cbrt	cbrt(x) → double	返回立方根
ceil	ceil(x)	返回比 x 大的最小整数，是ceiling的别名
ceiling	ceiling(x)	返回比 x 大的最小整数

函数	语法	说明
cosine_similarity	cosine_similarity(x, y) → double	返回两个稀疏向量的余弦相似度
degrees	degress(x) -> double	弧度换算成角度
e	e()->double	获取欧拉常数
exp	exp(x)->double	指数函数
floor	floor(x)	获取小于 x 的最大整数
from_base	from_base(string, radix) → bigint	获取字面量的值，该值的基数为 radix
inverse_normal_cdf	inverse_normal_cdf(mean, sd,p)->double	计算给定正态分布（均值、标准差和累计概率）的累计分布函数的倒数。
ln	ln(x)->double	返回自然对数值
log2	log2(x)->double	返回以 2 为底的对数
log10	log10(x)->double	返回以 10 为底的对数
log	log(x,b) -> double	返回以 b 为底数的对数
mod	mod(n,m)	返回 n/m 的余数
pi	pi()->double	返回常数 Pi
pow	pow(x,p)->double	计算 x^p 的值，`power`的别名
power	power(x,p)->double	计算 x^p 的值
radians	radians(x)->double	将角度换算成弧度
rand	rand()->double	获取一个为随机数，返回值范围为[0.0,1.0)。`random`的别名
random	random()->double	获取一个为随机数，返回值范围为[0.0,1.0)。
random	random(n)	获取一个为随机数，返回值范围为[0.0,n)。
round	round(x)	返回与 x 最接近的整数。
round	round(x, d)	返回与 x 最接近的数，精确到小数后 d 位。

函数	语法	说明
sign	sign(x)	符号函数，x 如果是整数，则当 x=0，返回0; x>0，返回1; x<0，返回-1。x 如果是浮点数，则当 x 为 NaN 时，返回 Nan；当 x 为 +∞ 时，返回1；当x为-∞时，返回-1。
sqrt	sqrt(x)->double	平方根函数
to_base	to_base(x, radix)->varchar	返回 x 以 radix 为基数的字面量
truncate	truncate(x) → double	截取整数部分
width_bucket	width_bucket(x, bound1, bound2, n) → bigint	获取 x 在[bound1, bound2]范围内 n 等的分直方图中的 bin 数
width_bucket	width_bucket(x, bins)	获取 x 在给定分布的直方图中的 bin 数
acos	acos(x)->double	获取 x 的反余弦值，x 为弧度
asin	asin(x)->double	获取 x 的反正弦值，x 为弧度
atan	atan(x)->double	获取 x 的反正切值，x 为弧度
atan2	atan2(y,x)->double	获取 y/x 的反正切值，x 为弧度
cos	cos(x)->double	获取 x 的余弦值，x为弧度
cosh	cosh(x)->double	获取 x 的双曲余弦值，x 为弧度
sin	sin(x)->double	获取 x 的正弦值，x为弧度
tan	tan(x)->double	获取 x 的正切值，x为弧度
tanh	tanh(x)->double	获取 x 的双曲正切值，x 为弧度
infinity	infinity() → double	获取正无穷常数
is_finite	is_finite(x) → boolean	判断 x 是否为有限数值
is_infinite	is_infinite(x) → boolean	判断 x 是否为无穷数值
is_nan	is_nan(x) → boolean	判断 x 是否不是一个数值类型的变量
nan	nan()	获取一个表示NAN（not-a-number）的常数

8.7.6 位运算函数

Presto 提供了如下几种位运算函数：

函数	语法	说明
bit_count	bit_count(x, bits) → bigint	返回 x 的补码中置 1 的位数
bitwise_and	bitwise_and(x, y) → bigint	位与函数
bitwise_not	bitwise_not(x) → bigint	取非操作
bitwise_or	bitwise_or(x, y) → bigint	位或函数
bitwise_xor	bitwise_xor(x, y) → bigint	抑或函数
bitwise_and_agg	bitwise_and_agg(x) → bigint	返回 x 中所有值的与操作结果，x 为数组
bitwise_or_agg	bitwise_or_agg(x) → bigint	返回 x 中所有值的或操作结果，x 位数组

示例：

```
SELECT bit_count(9, 64); -- 2
SELECT bit_count(9, 8); -- 2
SELECT bit_count(-7, 64); -- 62
SELECT bit_count(-7, 8); -- 6
```

8.7.7 Decimal 函数

字面量

使用如下形式来表示一个类型为`DECIMAL`的值的字面量：

```
DECIMAL 'xxxx.yyyyy'
```

DECIMAL 字面量的`precision`和其字面数字个数相同（包括前导的0），而`scale`和其小数位相同（包括后置的 0），示例如下：

字面量	数据类型
DECIMAL ‘0’	DECIMAL(1)
DECIMAL ‘12345’	DECIMAL(5)
DECIMAL ‘0000012345.1234500000’	DECIMAL(20, 10)

运算符

- 算术运算符

  假设有如下两个DECIMAL类型的变量 x， y:

  - x：DECIMAL(xp,xs)
  - y：DECIMAL(yp,ps)

  两个变量在参与算术运算时，遵循如下规则：

  - x + y或x - y
    - precision = *min*(38, 1 + *min*(xs, ys) + *min*(xp-xs, yp-ys))
    - scale = *max*(xs, ys)
  - x * y
    - precision = *min*(38, xp + yp)
    - scale = xs + ys
  - x / y
    - precision = *min*(38, xp + ys + *max*(0, ys-xs))
    - scale = *max*(xs, ys)
  - x % y
    - precision = *min*(xp - xs, yp - ys) + *max*(xs, bs)
    - scale = *max*(xs, ys)
- 比较运算符

  DECIMAL可以使用标准比较运算符和BETWEEN进行比较运算。
- 一元运算符

  DECIMAL可以使用一元运算符-取负数。

8.7.8 字符函数

拼接运算符

使用||运算符实现字符串的拼接。

字符函数

下表列出了 Presto 支持的字符函数：

函数名	语法	说明
chr	chr(n) → varchar	返回 UCP(*Unicode code point*) n作为一个字符返回.
codepoint	codepoint(string) → integer	返回字符string的 UCP 值
concat	concat(string1, …, stringN) → varchar	连接字符串，功能同||运算符
hamming_distance	hamming_distance(string1 , string2) → bigint	返回两个字符串之间的汉明距离。注意，两个字符串的长度应该相等。
length	length(string) → bigint	返回字符串的长度
levenshtein_distance	levenshtein_distance(string1, string2) → bigint	返回两个字符串之间的 Levenshtein 编辑距离
lower	lower(string) → varchar	转成小写
upper	upper(string) → varchar	转成大写
replace	replace(string, search) → varchar	用空字符来替换字符串string中和search相同的子串
replace	replace(string, search, replace) → varchar	用replace来替换字符串string中和search相同的子串
reverse	reverse(string) → varchar	反转字符串
lpad	lpad(string, size, padstring) → varchar	用padstring从左边开始填充字符串string,填充成长度为size的字符串。padstring不能为空，size不能为0.
rpad	rpad(string, size, padstring) → varchar	用padstring从右边开始填充字符串string,填充成长度为size的字符串。padstring不能为空，size不能为0.
ltrim	ltrim(string) → varchar	删除前置空白符
rtrim	rtrim(string) → varchar	删除后置空白符
split	split(string, delimiter) → array	拆分字符串
split	split(string, delimiter, limit) → array	拆分字符串，对数组大小有限制

函数名	语法	说明
split_part	split_part(string, delimiter, index) → varchar	从index处开始拆分字符串，index 从 1 开始。
split_to_map	split_to_map(string, entryDelimiter, keyValueDelimiter) → map<varchar, varchar>	将字符串拆成一个map
strpos	strpos(string, substring) → bigint	返回第一个符合子串的位置序号，序号从1开始.没找到则返回 0。
position	position(substring IN string) → bigint	返回子串在给定字符串中的起始位置
substr	substr(string, start, [length]) → varchar	截取从位置start位置开始的子串.位置序号从1开始.长度参数可选。

Unicode 相关的函数

- `normalize(string) → varchar`

  转换成NFC 标准形式的字符串.

- `normalize(string, form) → varchar`

  按给定的格式归一化字符串，`form`可选如下：

  - `NFD` Canonical Decomposition
  - `NFC` Canonical Decomposition, followed by Canonical Composition
  - `NFKD` Compatibility Decomposition
  - `NFKC` Compatibility Decomposition, followed by Canonical Composition

- `to_utf8(string) → varbinary`

  转换成 UTF-8 格式的字符串

- `from_utf8(binary, [replace]) → varchar`

  将二进制数据解析成 UTF-8 字符串。非法序列会用`replace`替代，该项参数可选，默认为 Unicode 字符`U+FFFD`。需要注意的是，`replace`必须是单个字符，可以为空字符。

8.7.9 正则表达式

Presto使用Java Pattern的正则表达式语法。但也有几个例外，如下所示：

- 多行模式
  - 使用?m开启多行模式
  - 行终止符为\n
  - 不支持?d选项
- 大小写敏感模式
  - 使用?i开启
  - 不支持?u选项
  - 不支持上下文敏感匹配
  - 不支持局部敏感匹配
- 不支持 Surrogate pairs
  - 如 Unicode U+10000必须使用\x{10000}来表示，而不能用\uD800\uDC00来表示
- 如果模式字符串中不包含基本字符，并且没有间隔，那么，使用边界字符\b会出错
- 不支持在字符类（如[A-Z123]）中使用\Q和\E
- 在Unicode字符类(\p{prop})的支持上存在如下不同：
  - 不支持下划线，如使用OldItalic代替Old_Italic
  - 不支持使用Is,script=,sc=来指定脚本，取而代之的是直接使用脚本名。如\p{Hiragana},而不是\p{script=Hiragana}
  - 不支持使用block=,blk=来表示区块，只能使用In。如\p{InMongolia}
  - 不支持使用Is, general_category=, gc=来指定分类，直接使用类别名称。如\p{L}。
  - 直接使用二进制属性，如使用\p{NoncharacterCodePoint}而不是\p{IsNoncharacterCodePoint}

下面介绍 Presto 提供的正则表达式函数：

- regexp_extract_all(string, pattern, [group]) → array<varchar>

  提取字符串string中所有与模式pattern匹配的子串。pattern中如果使用了分组的功能，则可以通过设置group参数，用于说明匹配哪个捕获组。

  示例

```
SELECT regexp_extract_all('1a 2b 14m', '\d+'); -- [1, 2, 14]
```

```
SELECT regexp_extract_all('1a 2b 14m', '(\d+)([a-z]+)', 2); -- ['a',
 'b', 'm']
```

- `regexp_extract(string, pattern, [group]) → varchar`

  功能和用法与`regexp_extract_all`类似，只是本函数只提取第一个匹配的结果。

  示例

```
SELECT regexp_extract('1a 2b 14m', '\d+'); -- 1
SELECT regexp_extract('1a 2b 14m', '(\d+)([a-z]+)', 2); -- 'a'
```

- `regexp_like(string, pattern) → boolean`

  判断字符串`string`中是否包含符合`pattern`模式的子串，包含返回`TRUE`，否则返回`FALSE`。本函数的功能与 SQL 中的`LIKE`语句功能相似，不同的是`LIKE`需要匹配整个模式字串，而本函数只需要字串中包含与模式字串相匹配子串即返回`TRUE`。

  示例

```
SELECT regexp_like('1a 2b 14m', '\d+b'); -- true
```

- `regexp_replace(string, pattern, [replacement]) → varchar`

  用`replacement`替换字符串`string`中所有符合模式`pattern`的子串。`replacement`可选，不设置将使用空字符`''`替换（即删除匹配的子串）。

  可以通过在`replacement`字串中使用`$g`（g为捕获组的序号，从1开始）或`${组名称}`来设置捕获组。美元符号`$`在`replacement`字串中需要使用`\$`进行转义。

  示例

```
SELECT regexp_replace('1a 2b 14m', '\d+[ab] '); -- '14m'
SELECT regexp_replace('1a 2b 14m', '(\d+)([ab]) ', '3c$2 '); -- '3ca
 3cb 14m'
```

- `regexp_split(string, pattern) → array<varchar>`

  使用模式字串`pattern`拆分字符串，保留尾部的空字符串。

  示例

```
SELECT regexp_split('1a 2b 14m', '\s*[a-z]+\s*'); -- ['1', '2', '14
', ''] 4个元素
                                                  -- 最后一个为空字符
```

8.7.10 二进制函数

拼接运算符

使用`||`运算符实现二进制串的拼接。

二进制函数

函数	语法	说明
length	length(binary) → bigint	返回二进制块的字节长度
concat	concat(binary1, ..., binaryN) → varbinary	将多个二进制块拼接在一起
to_base64	to_base64(binary) → varchar	获取二进制块的base64 编码
from_base64	from_base64(string) → varbinary	base64 解码
to_base64url	to_base64url(binary) → varchar	使用 URL 安全字符进行 base64 编码
from_base64url	from_base64url(string) → varbinary	使用 URL 安全字符进行 base64 解码
to_hex	to_hex(binary) → varchar	将二进制块编码为 16 进制字符串
from_hex	from_hex(string) → varbinary	将 16 进制编码的字符串解码成二进制块
to_big_endian_64	to_big_endian_64(bigint) → varbinary	将 bigint 编码为64位大端补码格式
from_big_endian_64	from_big_endian_64(binary) → bigint	64 位大端补码格式的二进制解码位 bigint 类型的数字
to_ieee754_32	to_ieee754_32(real) → varbinary	根据IEEE 754算法，将单精度浮点数编码为一个 32 位大端字节序的二进制块
to_ieee754_64	to_ieee754_64(double) → varbinary	根据IEEE 754算法，将双精度浮点数编码为一个 64 位大端字节序的二进制块
crc32	crc32(binary) → bigint	计算二进制块的CRC 32 值
md5	md5(binary) → varbinary	计算二进制块的MD 5 哈希值
sha1	sha1(binary) → varbinary	计算二进制块的SHA 1 哈希值
sha256	sha256(binary) → varbinary	计算二进制块的SHA 256 哈希值
sha512	sha512(binary) → varbinary	计算二进制块的SHA 512 哈希值
xxhash64	xxhash64(binary) → varbinary	计算二进制块的XXHASH 64 值

8.7.11 日期时间处理函数

日期和时间操作符

Presto 支持两种日期时间操作符+和-。

示例：

```
--- +
date '2012-08-08' + interval '2' day             --- 2012-08-10
time '01:00' + interval '3' hour                 --- 04:00:00.000
timestamp '2012-08-08 01:00' + interval '29' hour --- 2012-08-09 06:00
:00.000
timestamp '2012-10-31 01:00' + interval '1' month --- 2012-11-30 01:00
:00.000
interval '2' day + interval '3' hour             --- 2 03:00:00.000
interval '3' year + interval '5' month             --- 3-5
--- -
date '2012-08-08' - interval '2' day             --- 2012-08-06
time '01:00' - interval '3' hour                 --- 22:00:00.000
timestamp '2012-08-08 01:00' - interval '29' hour --- 2012-08-06 20:00
:00.000
timestamp '2012-10-31 01:00' - interval '1' month --- 2012-09-30 01:00
:00.000
interval '2' day - interval '3' hour             --- 1 21:00:00.000
interval '3' year - interval '5'                   --- month 2-7
```

时区转换

使用AT TIME ZONE操作符，可以实现时区转换。

示例：

```
SELECT timestamp '2012-10-31 01:00 UTC';
--- 2012-10-31 01:00:00.000 UTC
SELECT timestamp '2012-10-31 01:00 UTC' AT TIME ZONE 'America/
Los_Angeles';
--- 2012-10-30 18:00:00.000 America/Los_Angeles
```

时间和日期函数

- 基本函数

函数	语法	说明
current_date	current_date -> date	返回当前(查询启动时)日期
current_time	current_time -> time with time zone	返回当前时间
current_timestamp	current_timestamp -> timestamp with time zone	返回当前时戳
current_timezone	current_timezone() → varchar	返回当前时区

函数	语法	说明
date	date(x) → date	将日期字面量转换成日期类型的变量
from_iso8601_timestamp	from_iso8601_timestamp(string) → timestamp with time zone	将 ISO 8601 格式的时戳字面量转换成带时区的时戳变量
from_iso8601_date	from_iso8601_date(string) → date	将 ISO 8601 格式的日期字面量转换成日期类型的变量
from_unixtime	from_unixtime(unixtime, [timezone_str]) → timestamp	将UNIX时戳转换成时戳变量.可以带时区选项。
from_unixtime	from_unixtime(unixtime, hours, minutes) → timestamp with time zone	将 UNIX 时戳转换成带时区的时戳变量。`hours`和`minutes`表示时区偏移量。
localtime	localtime -> time	获取当前时间
localtimestamp	localtimestamp -> timestamp	获取当前时戳
now	now() → timestamp with time zone	获取当前时间，`current_time`的别名
to_iso8601	to_iso8601(x) → varchar	将`x`转换成ISO8601格式的字符串。这里`x`可以是`DATE`、`TIMESTAMP [with time zone]`这几个类型.
to_milliseconds	to_milliseconds(interval) → bigint	获取当前距当天零时已经过去的毫秒数
to_unixtime	to_unixtime(timestamp) → double	将时间戳转换成 UNIX 时间

 注意:

使用下列 SQL 标准函数时，不用使用圆括号：

- `current_data`
- `current_time`
- `current_timestamp`
- `localtime`
- `localtimestamp`

- 截断函数

  截断函数将时间日期变量按给定的单位进行截取，返回该单位的时间日期值。使用方法如下：

  `date_trunc(unit, x) -> [与 x 相同类型的变量]`

  其中，`unit`可以选如下几个值：

  - `second`，截取到秒
  - `minute`，截取到分钟
  - `hour`，截取到小时
  - `day`，截取到天
  - `week`，截取到星期
  - `month`， 截取到月份
  - `quarter`， 截取到季度
  - `year`，截取到年

- 时间间隔函数

  Presto 提供两个函数用于时间间隔计算，分别是：

  - `date_add(unit, value, timestamp) → [same as input]`

  计算给定增量的时间戳，增量可以为负，带单位。

  - `date_diff(unit, timestamp1, timestamp2) → bigint`

  计算两个时间戳的时间间隔，带单位。

  上面两个函数中，`unit`可以选如下几个值：

  - `ns`, 纳秒
  - `us`, 微秒
  - `ms`, 毫秒
  - `s`, 秒
  - `m`, 分
  - `h`, 小时
  - `d`, 天

- 日期时间域提取函数

  Presto 提供了从一个日期变量中提取指定域的函数`extract`,具体如下：

  ```
  extract(field FROM x) → bigint
  ```

  其中，`x`为要提取的日期时间变量，`field`为要提取的域，可取如下列表中的值：

  - `YEAR`， 年
  - `QUARTER`，季度
  - `MONTH`， 季度
  - `WEEK`， 星期
  - `DAY`， 天
  - `DAY_OF_MONTH`， 一个月中的第几天
  - `DAY_OF_WEEK`，一星期中的第几天
  - `DOW`, 同`DAY_OF_WEEK`
  - `DAY_OF_YEAR`， 一年中的第几天
  - `DOY`, 同`DAY_OF_YEAR`
  - `YEAR_OF_WEEK`，ISO Week中的年份
  - `YOW`， 同`YEAR_OF_WEEK`
  - `HOUR`， 小时
  - `MINUTE`，分钟
  - `SECOND`， 秒
  - `TIMEZONE_HOUR`， 小时，带时区
  - `TIMEZONE_MINUTE`，分钟，带时区

  为了方便使用，Presto 提供了如下辅助函数：

函数	语法	说明
day	day(x) → bigint	返回当前日期是一个月中的第几天
day_of_month	day_of_month(x) → bigint	同`day`
day_of_week	day_of_week(x) → bigint	返回当前日期是一天中的第几天
day_of_year	day_of_year(x) → bigint	返回当前时间是一年中的第几天

函数	语法	说明
dow	dow(x) → bigint	同`day_of_week`
doy	doy(x) → bigint	同`day_of_year`
hour	hour(x) → bigint	返回给定时间的小时数，取值范围为[0, 23]
minute	minute(x) → bigint	返回给定时间的分钟数，取之范围为[0, 59]
month	month(x) → bigint	返回给定时间的月份，取值范围为[1, 12]
quarter	quarter(x) → bigint	返回给定时间所属的季度
second	second(x) → bigint	返回给定时间的秒数，取之范围为[0, 59]
timezone_hour	timezone_hour(timestamp) → bigint	返回时区偏移量，单位小时
timezone_minute	timezone_minute(timestamp) → bigint	返回时区偏移量，单位分钟
week	week(x) → bigint	返回一年中的第几个信息，取值范围[1, 53]
week_of_year	week_of_year(x) → bigint	同`week`
year	year(x) → bigint	返回给定时间的年份值
year_of_week	year_of_week(x) → bigint	返回`x`(ISO Week格式)中的年份
yow	yow(x) → bigint	同`year_of_week`

- MySQL 日期函数

  Presto 提供了两个日期解析相关的函数，用于兼容 MySQL 的日期函数`date_parse`和`str_to_date`,它们分别是：

  - `date_format(timestamp, format) → varchar`

    使用`format`格式化`timestamp`。

  - `date_parse(string, format) → timestamp`

    按`format`格式解析日期字面量。

  Presto 支持的 MySQL 格式符号如下表所示：

符号	说明
%a	星期简写 (Sun .. Sat)
%b	月份简写 (Jan .. Dec)
%c	月份，数字(1 .. 12)，不可以为0
%d	月中天数，数字(01 .. 31)，不可以为0
%e	月中天数，数字 (1 .. 31)，不可以为0
%f	秒数 (6 digits for printing: 000000 .. 999000; 1 - 9 digits for parsing: 0 .. 999999999)
%H	小时 (00 .. 23)
%h	小时 (01 .. 12)
%I	小时 (01 .. 12)
%i	分钟 (00 .. 59)
%j	一年中的第几天 (001 .. 366)
%k	小时 (0 .. 23)
%l	小时 (1 .. 12)
%M	月份名称 (January .. December)
%m	月份，数字 (01 .. 12) [4]
%p	AM ／ PM
%r	时间, 12小时 (hh:mm:ss AM/PM)
%S	秒 (00 .. 59)
%s	秒 (00 .. 59)

符号	说明
%T	时间, 24hour (hh:mm:ss)
%v	星期 (01 .. 53), 第一条为星期一，与%X配合使用
%W	星期名称 (Sunday .. Saturday)
%x	年份，数字，4位，第一天为星期一
%Y	年份，数字，4位
%y	年份，数字，2位, 表示年份范围为[1970, 2069]
%%	表示字符'%'

说明:

Presto 目前不支持的符号有：`%D %U %u %V %w %X`

- Java 日期函数

  下列函数用于兼容 Java 的日期格式（JodaTime Pattern）.

  - `format_datetime(timestamp, format) → varchar`，格式化时间戳
  - `parse_datetime(string, format) → timestamp with time zone`，解析时间戳字符串

8.7.12 聚合函数

聚合函数具有如下特点：

- 输入一个数据集
- 输出一个单一的计算结果

绝大部分聚合函数都会在计算时忽略`null`值，并且在输入为空或均为`null`时，返回`null`。但也有例外，如下几个聚合函数：

- count
- count_if
- max_by
- min_by
- approx_distinct

基本聚合函数

函数	语法	说明
arbitrary	arbitrary(x) → [same as input]	随机返回 x 中的一个非 null 值
array_agg	array_agg(x) → array<[same as input]>	从输入的元素中创建数组
avg	avg(x) → double	求算术平均值
avg	avg(time interval type) → time interval type	计算输入时间序列的平均时间间隔
bool_and	bool_and(boolean) → boolean	如果所有输入的值都为 TRUE，则返回 TRUE，否则返回 FALSE
bool_or	bool_or(boolean) → boolean	如果输入的序列中有一个为 True，则返回 True，否则返回 False
checksum	checksum(x) → varbinary	返回 x 的校验和（顺序不敏感）
count	count(*) → bigint	返回行数
count	count(x) → bigint	返回非 null 元素的个数
count_if	count_if(x) → bigint	返回 x 中元素为True 的个数，等同于`count(CASE WHEN x THEN 1 END)`.
every	every(boolean) → boolean	同`bool_and`
geometric_mean	geometric_mean(x) → double	返回 x 的几何平均值
max_by	max_by(x, y) → [same as x]	返回与 y 的最大值相关的 x 值
max_by	max_by(x, y, n) → array<[same as x]>	返回与 y 的前 n 个最大值相关的 x 值的数组
min_by	min_by(x, y) → [same as x]	返回与 y 的最小值相关的 x 值
min_by	min_by(x, y, n) → array<[same as x]>	返回与 y 的前 n 个最小值相关的 x 值的数组
max	max(x) → [same as input]	返回最大值
max	max(x, n) → array<[same as x]>	返回前 n 个最大值列表
min	min(x) → [same as input]	返回最小值

函数	语法	说明
min	min(x, n) → array<[same as x]>	返回前 n 个最小值列表
sum	sum(x) → [same as input]	求和

位聚合函数

位聚合函数参见位运算函数中介绍的`bitwise_and_agg`和`bitwise_or_agg`函数。

Map 聚合函数

函数	语法	说明
histogram	histogram(x) → map<K, bigint>	统计直方图
map_agg	map_agg(key, value) → map<K,V>	创建一个`MAP`类型的变量
map_union	map_union(x<K, V>) → map<K,V>	返回输入map列表的 Union 结果，如果有多个 map 对象包含相同的key，最终的结果中，对于 key 的 value 随机的从输入的 map 中选取。
multimap_agg	multimap_agg(key, value) → map<K,array>	创建一个多重映射的`MAP`变量

近似聚合函数

函数	语法	说明
approx_distinct	approx_distinct(x, [e]) → bigint	返回输入列表中不重复的值的个数。本函数返回的是`count (DISTINCT x)`的近似值，如果所有值都是 null，则返回 0。`e`为期望标准差的上界，可选，默认为 2.3%，当前的实现方式对`e`的取值范围有约束，要求在[0.01150, 0.26000]之间。对于特定对输入，不保证误差上界。
approx_percentile	approx_percentile(x, percentage) → [same as x]	估计序列 x 中位于第百分之 percentage 位的数值

函数	语法	说明
approx_percentile	approx_percentile(x, percentages) → array<[same as x]>	类似上面，percentages 为数组，返回值与之一一对应。
approx_percentile	approx_percentile(x, w, percentage) → [same as x]	类似上面，w为x的权值。
approx_percentile	approx_percentile(x, w, percentage, accuracy) → [same as x]	类似上面，`accuracy`为预估精度的上线，取值范围为[0, 1].
approx_percentile	approx_percentile(x, w , percentages) → array<[same as x]>	类似上面，percentages为数组，返回值与之一一对应.
numeric_histogram	numeric_histogram(buckets, value, [weight]) → map<double, double>	按给定的桶数计算数值直方图。`buckets`必须是`BIGINT`类型， `value`和`weight` 必须是数值类型。权重列表weight可选，默认为1。

统计聚合函数

函数	语法	说明
corr	corr(y, x) → double	计算相关系数
covar_pop	covar_pop(y, x) → double	计算总体协方差
covar_samp	covar_samp(y, x) → double	计算样本协方差
kurtosis	kurtosis(x) → double	计算超值峰度. 使用下列表达式进行无偏估计： `kurtosis(x) = n(n+1)/((n-1)(n-2)(n-3))sum[(x_i-mean)^4]/sttdev(x)^4-3(n-1)^2/((n-2)(n-3))`
regr_intercept	regr_intercept(y, x) → double	计算线性回归截距. `y`为相关变量. `x`为独立变量.
regr_slope	regr_slope(y, x) → double	计算线性回归斜率。`y`为相关变量. `x`为独立变量。
skewness	skewness(x) → double	计算偏度。
sttdev_pop	sttdev_pop(x) → double	计算总体标准差。

函数	语法	说明
sttdev_samp	sttdev_samp(x) → double	计算样本标准差。
sttdev	sttdev(x) → double	计算标准差，同`sttdev_samp`。
var_pop	var_pop(x) → double	计算总体方差。
var_samp	var_samp(x) → double	计算样本方差。
variance	variance(x) → double	同`var_samp`。

9 TensorFlow 使用说明

E-MapReduce 3.13.0 以后版本支持TensorFlow 。您可以在软件配置可选服务中添加 TensorFlow 组件。当您使用 E-MapReduce TensorFlow 运行高性能计算时，可通过 YARN 对集群中的 CPU、GPU 进行调度。

准备工作

- 软件层面，E-MapReduce 集群安装 TensorFlow 和 TensorFlow on YARN 组件。
- 硬件层面，E-MapReduce 支持使用 CPU 和 GPU 两种资源进行计算。如您需使用 GPU 进行计算，可在 core 或 task 节点中选择 ECS 异构计算 GPU 计算型，如 ecs.gn5，ecs.gn6 机型。选定 GPU 机型后，选择您所需的 CUDA 和 CuDNN 版本。

提交 TensorFlow 作业

用户可以登录 EMR 主节点已命令行的方式提交 Tensorflow 作业，例如：

```
el_submit [-h] [-t APP_TYPE] [-a APP_NAME] [-m MODE] [-m_arg MODE_ARG]
[-interact INTERACT] [-x EXIT]
[-enable_tensorboard ENABLE_TENSORBOARD]
[-log_tensorboard LOG_TENSORBOARD] [-conf CONF] [-f FILES]
[-pn PS_NUM] [-pc PS_CPU] [-pm PS_MEMORY] [-wn WORKER_NUM]
[-wc WORKER_CPU] [-wg WORKER_GPU] [-wm WORKER_MEMORY]
[-wnpg WNPG] [-ppn PPN] [-c COMMAND [COMMAND ...]]
```

基础参数说明：

- -t APP_TYPE 提交的任务类型，支持三种类型的任务类型 tensorflow-ps， tensorflow-mpi，standalone，三种类型要配合后面运行模式使用。
  - tensorflow-ps 采用 Parameter Server 方式通信，该方式为原生 TensorFlow PS 模式。
  - tensorflow-mpi 采用的是 UBER 开源的 MPI 架构 horovod 进行通信。
  - standalone 模式是用户将任务调度到 YARN 集群中启动单机任务，类似于单机运行。
- –a APP_NAME 是指提交 TensorFlow 提交的作业名称，用户可以根据需要自行命名。

- –m MODE，是指 TensorFlow 作业提交的运行时环境，E-MapReduce 支持三种类型运行环境 local, virtual-env, docker。
  - local 模式，使用的是 emr-worker 上面的 python 运行环境，所以如果要使用一些第三方 Python 包需要手动在所有机器上进行安装。
  - docker 模式， 使用的是 emr-worker 上面的 docker 运行时，tensorflow 运行在 docker container 内。
  - virtual-env 模式，用户上传的 Python 环境，可以在 Python 环境中安装一些不同于 worker 节点的 Python 库。
- -m_arg MODE_ARG，提交运行模式的补充参数，和 MODE 配合使用，如果运行时是 docker，则设置为 docker镜像名称，如果是 virtual-env，则指定 Python 环境文件名称，tar.gz 打包。
- –x Exit分布式 TensorFlow 有些 API 需要用户手动退出 PS，在这种情况下用户可以指定 -x 选项，当所有 worker 完成训练并成 功后，PS 节点自动退出。
- -enable_tensorboard 是否在启动训练任务的同时启动 TensorBoard。
- -log_tensorboard 如果训练同时启动 TensorBoard，需要指定 TensorBoard 日志位置，需要时 HDFS 上的位置。
- -conf CONF Hadoop conf 位置，默认可以不设，使用 EMR 默认配置即可。
- –f FILES 运行 TensorFlow 所有依赖的文件和文件夹，包含执行脚本，如果设置 virtual-env 执行的 virtual-env 文件，用户可 以将所有依赖放置到一个文件夹中，脚本会自动将文件夹按照文件夹层次关系上传到 HDFS 中。
- -pn TensorFlow 启动的参数服务器个数。
- -pc 每个参数服务器申请的CPU核数。
- -pm 每个参数服务器申请的内存大小。
- -wn TensorFlow 启动的Worker节点个数。
- -wc 每个 Worker 申请的 CPU 核数。
- -wg 每个 Worker 申请的 GPU 核数。
- -wm 每个 Worker 申请的内存个数。
- -c COMMAND执行的命令，如 pythoncensus.py。

进阶选项，用户需要谨慎使用进阶选项，可能造成任务失败。

- -wnpg 每个 GPU 核上同时跑的 Worker 数量（针对 tensorflow-ps）。
- -ppn 每个 GPU 核上同时跑的 Worker 数量（针对 horovod） 以上两个选项指的是单卡多进程操作，由于共用一张显卡，需要在程序上进行一定限制，否则会造成显卡 OOM。

10 Knox 使用说明

目前 E-MapReduce 中支持了 Apache Knox，选择支持 Knox 的镜像创建集群，完成以下准备工作后，即可在公网直接访问 Yarn，HDFS，SparkHistory 等服务的 Web UI。

准备工作

- 开启 Knox 公网 IP 访问

  1. 在 E-MapReduce 上 Knox 的服务端口是 8443，在集群详情中找到集群所在的 ECS 安全组。
  2. 在 ECS 控制台修改对应的安全组，在入方向添加一条规则，打开 8443 端口。

  > 注意:
  > - 为了安全原因，这里设置的授权对象必须是您的一个有限的 IP 段范围，禁止使用 0.0.0.0/0。
  > - 打开安全组的 8443 端口之后，该安全组内的所有机器均会打开公网入方向的 8443 端口，包括非 E-MapReduce 的 ECS 机器。

- 设置 Knox 用户

  访问 Knox 时需要对身份进行验证，会要求您输入用户名和密码。Knox 的用户身份验证基于 LDAP，您可以使用自有 LDAP 服务，也可以使用集群中的 Apache Directory Server 的 LDAP 服务。

  - 使用集群中的 LDAP 服务

    方式一（推荐）

    在用户管理中直接添加 Knox 访问账号。

    方式二

    1. SSH 登录到集群上，详细步骤请参见SSH登录集群。
    2. 准备您的用户数据，如：Tom。将文件中所有的 `emr-guest` 替换为 `Tom`，将 `cn:EMR GUEST` 替换为 `cn:Tom`，设置 `userPassword` 的值为您自己的密码。

    ```
    su knox
    cd /usr/lib/knox-current/templates
    vi users.ldif
    ```

    > 注意:

出于安全原因，导入前务必修改 users.ldif 的用户密码，即：设置 `userPassword` 的值为您自己的用户密码。

3. 导入到 LDAP。

```
su knox
cd /usr/lib/knox-current/templates
sh ldap-sample-users.sh
```

- 使用自有 LDAP 服务的情况

1. 在集群配置管理中找到 KNOX 的配置管理，在 cluster-topo 配置中设置两个属性：`main.ldapRealm.userDnTemplate` 与 `main.ldapRealm.contextFactory.url`。main.ldapRealm.userDnTemplate 设置为自己的用户 DN 模板，`main.ldapRealm.contextFactory.url` 设置为自己的 LDAP 服务器域名和端口。设置完成后保存并重启 Knox。

2. 一般自己的 LDAP 服务不在集群上运行，所以需要开启 Knox 访问公网 LDAP 服务的端口，如：10389。参考 8443 端口的开启步骤，选择出方向。

注意:
为了安全原因，这里设置的授权对象必须是您的Knox所在集群的公网 IP，禁止使用 0.0.0.0/0。

开始访问 Knox

- 使用 E-MapReduce 快捷链接访问

1. 登录阿里云 E-MapReduce 控制台。
2. 单击对应的集群 ID
3. 在左侧导航栏中单击集群与服务管理。
4. 单击相应的服务，如：HDFS，Yarn等。
5. 单击右上角的快捷链接。

- 使用集群公网 IP 地址访问

  1. 通过集群详情查看公网 IP。
  2. 在浏览器中访问相应服务的 URL。
     - HDFS UI：https://{集群公网ip}:8443/gateway/cluster-topo/hdfs/
     - Yarn UI：https://{集群公网ip}:8443/gateway/cluster-topo/yarn/
     - SparkHistory UI：https://{集群公网ip}:8443/gateway/cluster-topo/sparkhistory/
     - Ganglia UI：https://{集群公网ip}:8443/gateway/cluster-topo/ganglia/
     - Storm UI：https://{集群公网ip}:8443/gateway/cluster-topo/storm/
     - Oozie UI：https://{集群公网ip}:8443/gateway/cluster-topo/oozie/
  3. 浏览器显示您的链接不是私密链接，是因为 Knox 服务使用了自签名证书，请再次确认访问的是自己集群的 IP，且端口为 8443。单击高级 > 继续前往。
  4. 在登录框中输入您在 LDAP 中设置的用户名和密码

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，特定的用户组和特定的 IP 地址访问特定的服务，可以参见Apache Knox 授权。

- 示例：
  - 场景：Yarn UI 只允许用户 Tom 访问。
  - 步骤：在集群配置管理中找到 KNOX 的配置管理，找到 cluster-topo 配置，在 cluter-topo 配置的`<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>
```

警告：

Knox 会开放相应服务的 REST API，用户可以通过各服务的 REST API 操作服务，如：HDFS 的文件添加，删除等。出于安全原因，请务必确保在 ECS 控制台开启安全组 Knox 端口 8443 时，授权对象必须是您的一个有效 IP 地址段，禁止使用 0.0.0.0/0；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。

11 Flume

11.1 Flume 使用说明

本文以 E-MapReduce-Flume 实时同步 HDFS audit 日志至 HDFS 为例，介绍 Flume 的使用。

背景信息

E-MapReduce 从 3.19.0 版本开始对 EMR-Flume 提供集群管理的功能。通过集群管理功能，可以在 Web 页面方便的配置和管理 Flume Agent。示例中，在 master 实例启动 Flume agent，收集本地磁盘中的 audit 日志通过 Avro 协议发送数据至 core 实例，在 core 实例配置并启动 failover sink processor，接收 master 实例发送的数据并 sink 到 HDFS 中。 Flume 其他使用场景的配置见 #unique_159。

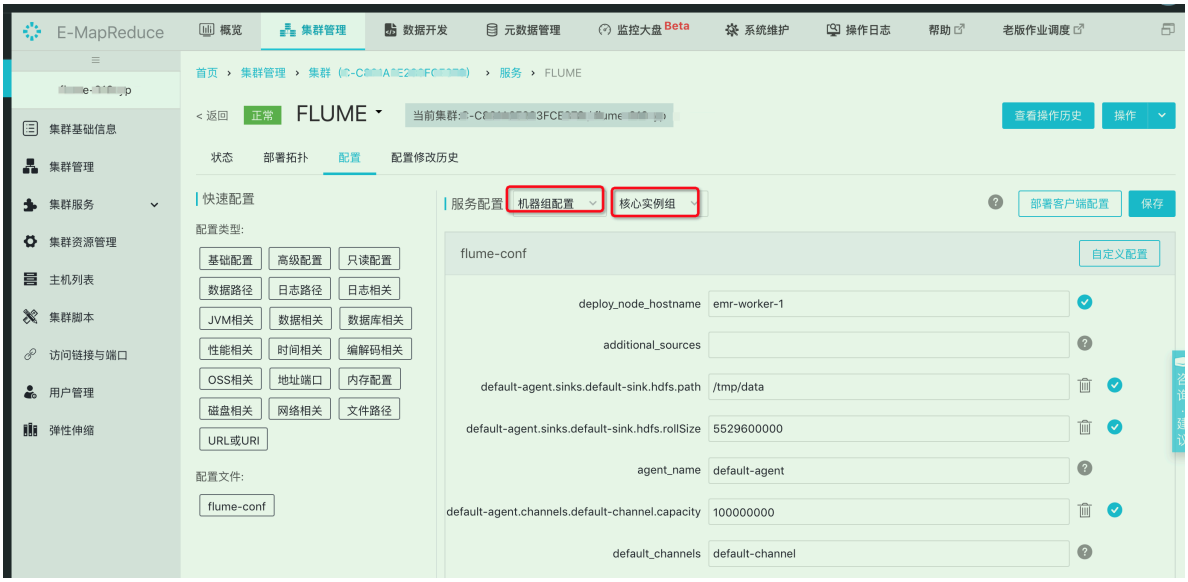
准备工作

创建 E-MapReduce Hadoop 集群，在可选服务中选择 Flume。具体操作请参见#unique_160。

操作步骤

- Core 实例配置并启动 Flume Agent

  例如在 emr-worker-1 节点进行操作，选择核心实例组进行配置，如下图所示：



1. 在配置页面设置如下：

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

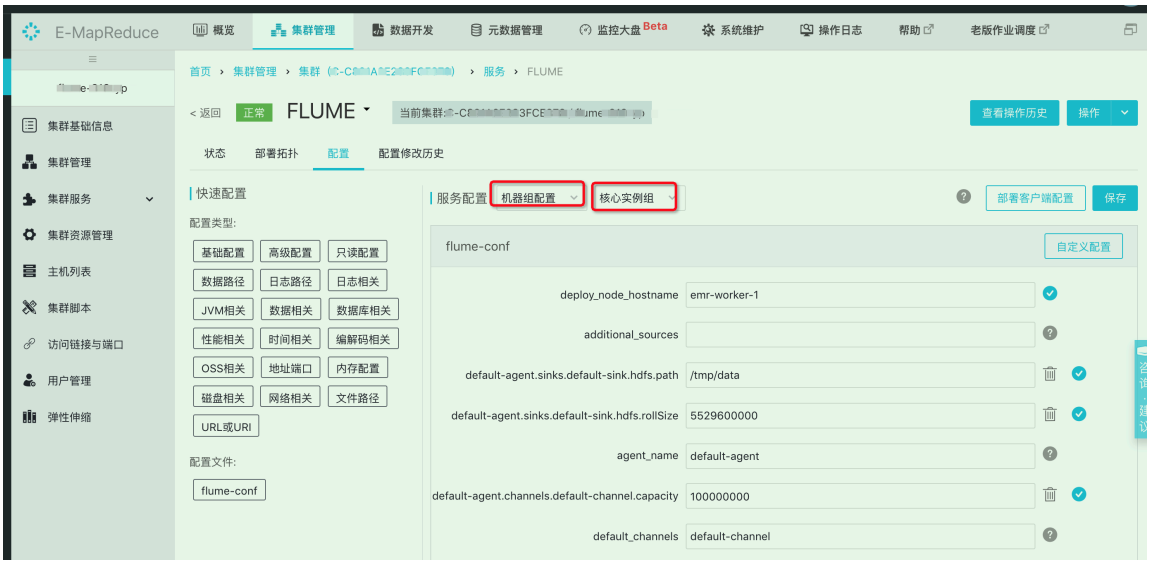
2. 在配置页面通过自定义配置添加如下配置：

default-agent.sinks.default-sink.hdfs.path	对于高可用集群，使用 hdfs://emr-cluster/path 形式的地址
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	根据实际设置
default-agent.channels.default-channel.transactionCapacity	51200
default-agent.channels.default-channel.dataDirs	channel 存储 event 数据的路径
default-agent.channels.default-channel.checkpointDir	存储 checkpoint 的路径

default-agent.channels.default-channel.capacity	根据 hdfs roll 进行设置

3. 保存配置后启动 Flume agent

4. 单击查看操作历史，显示操作成功后，部署拓扑页面可以看到 emr-worker-1 节点的 flume 已经是 started 状态。



emr-worker-1 节点启动成功后，开始启动第二个 worker 节点。同样的方式，例如在 worker-2 节点启动 flume，修改配置项：

deploy_node_hostname	节点的 hostname
default-agent.sinks.default-sink.hdfs.path	对于高可用集群，使用 hdfs://emr-cluster/path 形式的地址

5. 保存配置后，启动 All Components，指定机器为 emr-worker-2。

- Master 实例配置并启动 Flume Agent

  例如在 emr-header-1 节点进行操作，选择服务配置。

配置 agent 如下：

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

新增配置如下：

配置项	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
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

配置项	值
default-agent.channels.default-channel.capacity	根据实际情况设置
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
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
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

查看同步结果

使用 HDFS 命令，可以看到同步的数据被写入 FlumeData.${timestamp} 形式的文件中，其中 timestamp 为文件创建的时间戳。

```
[root@emr-header-1 ~]# hdfs dfs -ls /tmp/emr-worker-1/data
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/StaticLogge
rBinder.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 174 items
-rw-r--r--   3 root hadoop   34641327 2019-03-27 10:49 /tmp/emr-worker-1/data/FlumeData.1553654928352
-rw-r--r--   3 root hadoop   30963637 2019-03-27 10:49 /tmp/emr-worker-1/data/FlumeData.1553654958983
-rw-r--r--   3 root hadoop   29282243 2019-03-27 10:50 /tmp/emr-worker-1/data/FlumeData.1553654989023
-rw-r--r--   3 root hadoop   31138131 2019-03-27 10:50 /tmp/emr-worker-1/data/FlumeData.1553655019874
-rw-r--r--   3 root hadoop   29995885 2019-03-27 10:51 /tmp/emr-worker-1/data/FlumeData.1553655049922
-rw-r--r--   3 root hadoop   30238902 2019-03-27 10:51 /tmp/emr-worker-1/data/FlumeData.1553655079981
-rw-r--r--   3 root hadoop   31127038 2019-03-27 10:52 /tmp/emr-worker-1/data/FlumeData.1553655110987
-rw-r--r--   3 root hadoop   30121183 2019-03-27 10:52 /tmp/emr-worker-1/data/FlumeData.1553655141136
-rw-r--r--   3 root hadoop   30124697 2019-03-27 10:53 /tmp/emr-worker-1/data/FlumeData.1553655171175
-rw-r--r--   3 root hadoop   30624989 2019-03-27 10:53 /tmp/emr-worker-1/data/FlumeData.1553655202156
-rw-r--r--   3 root hadoop   30122940 2019-03-27 10:54 /tmp/emr-worker-1/data/FlumeData.1553655232325
-rw-r--r--   3 root hadoop   30122940 2019-03-27 10:54 /tmp/emr-worker-1/data/FlumeData.1553655262387
```

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

集群与服务管理页面提供了 Flume agent 的监控信息。通过在集群与服务管理页面单击 Flume 服务进行访问，如下图所示：

11.2 Flume 配置说明

E-MapReduce（以下简称 EMR） 从 3.16.0 版本开始支持 Apache Flume。本文介绍使用 Flume 将数据从 EMR Kafka 集群同步至 EMR Hadoop 集群的 HDFS、Hive和 HBase，以及阿里云的 OSS。Flume 的使用说明见 #unique_162。

准备工作

- 创建 Hadoop 集群时，在可选服务中选择 Flume。
- 创建 Kafka 集群，并创建名称为 flume-test 的 topic，用于生成数据。

 **说明:**

- 如果创建的是 Hadoop 高安全集群，消费标准 Kafka 集群的数据，参照兼容 MIT Kerberos 认证在 Hadoop 集群配置 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，参见 Kerberos Kafka Source 部分；
- 如果创建的 Hadoop 集群和 Kafka 集群都是高安全集群，参照跨域互信进行配置，参见跨域互信使用 Flume；

Kafka->HDFS

- 配置 Flume

  创建配置文件*flume.properties*，添加如下配置。其中，配置项 `a1.sources.source1.kafka.bootstrap.servers` 为 Kafka 集群 broker 的 host 和端口号，`a1.sources.source1.kafka.topics` 为 Flume 消费 Kafka 数据的 topic，`a1.sinks.k1.hdfs.path` 为 Flume 向 HDFS 写入数据的路径：

```
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 = /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
```

   **说明:**

配置项 a1.sinks.k1.hdfs.path 如果使用 URL 的形式，对于高可用集群，使用 hdfs://emr-cluster，例如：

```
a1.sinks.k1.hdfs.path = hdfs://emr-cluster/tmp/flume/test-data
```

对于标准集群，使用hdfs://emr-header-1:9000，例如：

```
a1.sinks.k1.hdfs.path = hdfs://emr-header-1:9000/tmp/flume/test-data
```

- 启动服务

  Flume 的默认配置文件放在 */etc/ecm/flume-conf* 下，使用该配置启动 Flume Agent：

  ```
  flume-ng agent --name a1 --conf /etc/ecm/flume-conf  --conf-file flume.properties
  ```

  启动 Agent 后，因为使用了 */etc/ecm/flume-conf* 下的 *log4j.properties*，会在当前路径下生成日志 *logs/flume.log*，可根据实际使用对 *log4j.properties* 进行配置。

- 测试

  在 Kafka 集群使用 *kafka-console-producer.sh* 输入测试数据 abc

  Flume 会在 HDFS 中以当前时间的(毫秒)时间戳生成文件 FlumeData.xxxx，查看文件内容，会看到在 Kafka 中输入的数据。

Kafka->Hive

- 创建 Hive 表

  Flume 使用事务操作将数据写入 Hive，需要在创建 Hive 表时设置 transactional 属性，如创建 flume_test 表：

  ```
  create table flume_test (id int, content string)
  clustered by (id) into 2 buckets
  stored as orc  TBLPROPERTIES('transactional'='true');
  ```

- 配置 Flume

  创建配置文件 *flume.properties*，添加如下配置。其中，配置项 a1.sources.source1.kafka.bootstrap.servers 填写 Kafka 集群 broker 的 host 和端口号，a1.sinks.k1.hive.metastore 为 Hive metastore URI，配置为 *hive-site.xml* 中配置项 hive.metastore.uris 的值：

  ```
  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
```

- **启动 Flume**

```
flume-ng agent --name a1 --conf /etc/ecm/flume-conf  --conf-file
flume.properties
```

- **生成数据**

  在 Kafka 集群中使用*kafka-console-producer.sh*，以逗号为分隔符输入测试数据 1，a。

- **检测数据写入**

  查询 Hive 事务表需要在客户端进行配置：

```
hive.support.concurrency – true
hive.exec.dynamic.partition.mode – nonstrict
hive.txn.manager – org.apache.hadoop.hive.ql.lockmgr.DbTxnManager
```

  配置好后查询 flume_test 表中的数据

Kafka->HBase

- **创建 HBase 表**

  创建 HBase 表 flume_test 及列簇 column

- 配置 Flume

  创建配置文件 *flume.properties*，添加如下配置。其中，配置项 a1.sources.source1.kafka.bootstrap.servers 为 Kafka 集群 broker 的 host 和端口号，a1.sinks.k1.table 为 HBase 表名，a1.sinks.k1.columnFamily 为列簇名：

```
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 = 1000
a1.channels.c1.transactionCapacity = 100

# Bind the source and sink to the channel
a1.sources.source1.channels = c1
a1.sinks.k1.channel = c1
```

- 启动服务

```
flume-ng agent --name a1 --conf /etc/ecm/flume-conf  --conf-file
flume.properties
```

- 测试

  在 Kafka 集群使用 *kafka-console-producer.sh* 生成数据后，就可以在 HBase 查到数据

Kafka->OSS

- 创建 OSS 路径

  创建 OSS Bucket 及目录，如 *oss://flume-test/result*

- 配置 Flume

  Flume 向 OSS 写入数据时，需要占用较大的 JVM 内存，可以改小 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，例如设置为 1 G：

    ```
    export JAVA_OPTS="-Xmx1g"
    ```

  创建配置文件 *flume.properties*，添加如下配置。其中，配置项 *a1.sources.source1.kafka.bootstrap.servers* 填写 Kafka 集群 broker 的 host 和端口号，*a1.sinks.k1.hdfs.path* 为 OSS 路径：

  ```
  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
```

- 启动 Flume

  如果配置 Flume 时修改了 OSS 缓存大小，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/*"
  ```

- 测试

  在 Kafka 集群使用 *kafka-console-producer.sh* 生成数据后，在 OSS 的 *oss://flume-test/result* 路径下会以当前时间的(毫秒)时间戳为后缀生成文件 *FlumeData.xxxx*。

Kerberos Kafka source

消费高安全 Kafka 集群的数据时，需要做额外的配置：

- 参见兼容 MIT Kerberos 认证在 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；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*，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";
    ```

```
};
```

其中，${realm} 替换为 Kafka 集群的 Kerberos realm。获取方式为：

■ 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*，最后的数字串 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*。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*。长域名的形式例如 *emr-header-1.cluster-123456*。

跨域互信使用 Flume

在配置了跨域互信后，其他配置如下：

· 参见兼容 MIT Kerberos 认证在 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。

· 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";
```

```
};
```

其中，${realm} 替换为 Kafka 集群的 Kerberos realm。获取方式为：

■ 在 Kafka 集群执行命令hostname，得到形式为*emr-header-1.cluster-xxx*的主机名，如*emr-header-1.cluster-123456*，最后的数字串 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*。添加如下内容：

```
export JAVA_OPTS="$JAVA_OPTS -Djava.security.auth.login.config=/
etc/ecm/flume-conf/flume_jaas.conf""
```

11.3 使用 LogHub Source 将非 E-MapReduce 集群的数据同步至 E-MapReduce 集群的 HDFS

本文介绍使用 EMR-Flume 实时同步 Log Service 的数据至 EMR 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

E-MapReduce 从 3.20.0 版本开始对 EMR-Flume 加入了 Log Service Source。借助 Log Service 的 Logtail 等工具，可以将需要同步的数据实时采集并上传到 LogHub，再使用 EMR-Flume 将 LogHub 的数据同步至 EMR 集群的 HDFS。 有关采集数据到 Log Service 的 LogHub 的详细方法及步骤参见#unique_166。

准备工作

创建 Hadoop 集群，在可选软件中选择 Flume，详细步骤请参见#unique_22。

配置 Flume

- 配置 Source

配置项	值	说明
type	org.apache.flume.source.loghub.LogHubSource	
endpoint	LogHub 的 Endpoint	如果使用 VPC/经典网络的 endpoint，要保证与 EMR 集群在同一个地区；如果使用公网 endpoint，要保证运行 Flume agent 的节点有公网 IP。

配置项	值	说明
project	LogHub 的 project	
logstore	LogHub 的 logstore	
accessKeyId	阿里云的 AccessKey ID	
accessKey	阿里云的 AccessKey	
useRecordTime	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	consumer_1	消费组名称，默认值为consumer_1

其他配置项说明如下：

- `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；可以设置的其他值为 begin 或 special。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，参见#unique_167；或者更改配置consumerGroup 为新的消费组。
- `heartbeatInterval` 和 `fetchInOrder`

  heartbeatInterval 表示消费组与服务端维持心跳的间隔，单位是毫秒，默认为 30 秒；fetchInOrder 表示相同 key 的数据是否按序消费，默认值为 false。
- batchSize 和 batchDurationMillis

  通用的 source batch 配置，表示触发 event 写入 channel 的阈值。
- backoffSleepIncrement 和 maxBackoffSleep

  通用的 source sleep 配置，表示 LogHub 没有数据时触发 sleep 的时间和增量。

- 配置 channel 和 sink

  此处使用 memory channel 和 HDFS sink。

  - HDFS Sink 配置如下：

配置项	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：

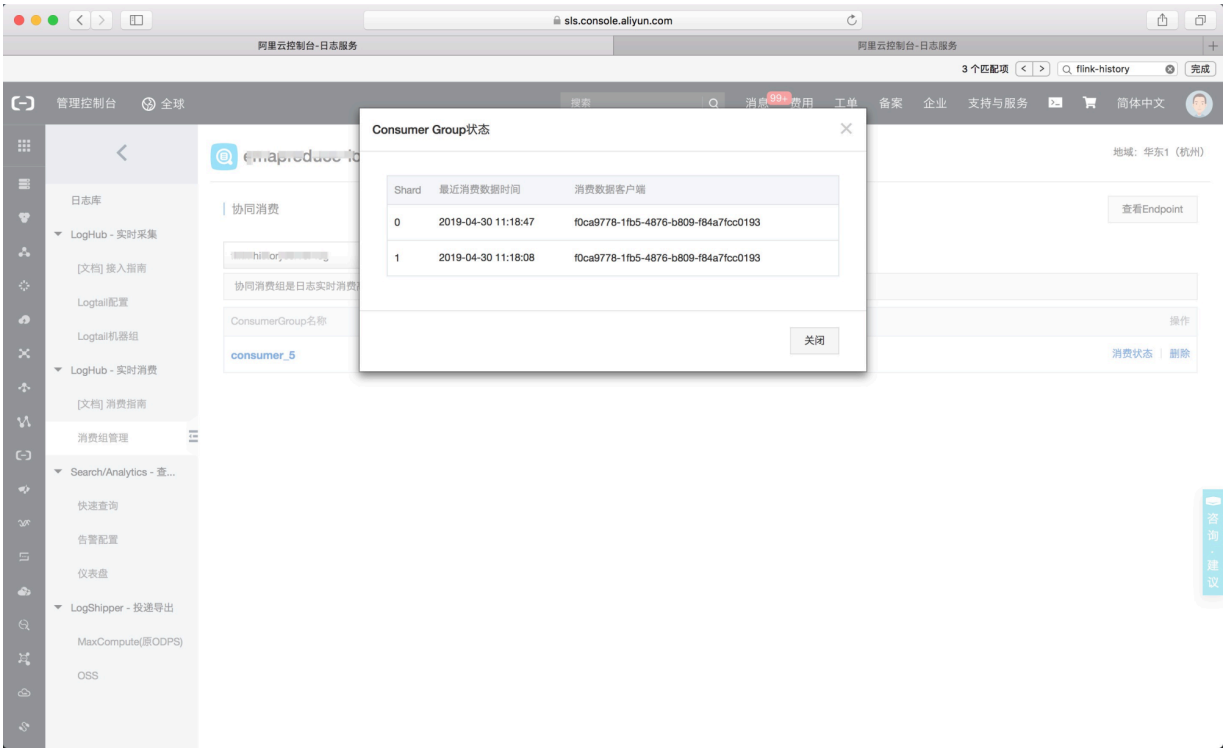
配置项	值
capacity	2000
transactionCapacity	2000

运行 Flume agent

在 阿里云 E-Mapreduce 控制台页面启动 Flume agent 的具体操作参见 #unique_168。成功启动后，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 ~]#
```

查看 Log Service 上的消费组状态，详细步骤请参见#unique_167：


sls.console.aliyun.com
阿里云控制台-日志服务
阿里云控制台-日志服务
管理控制台
全球
Consumer Group状态
Shard
最近消费数据时间
消费数据客户端
0
2019-04-30 11:18:47
f0ca9778-1fb5-4876-b809-f84a7fcc0193
1
2019-04-30 11:18:08
f0ca9778-1fb5-4876-b809-f84a7fcc0193
关闭
协同消费
查看Endpoint
consumer_5
消费状态
删除

12 Sqoop 使用说明

Sqoop 是一款用来在不同数据存储软件之间进行数据传输的开源软件，它支持多种类型的数据储存软件。

安装 Sqoop

注意:

目前，E-MapReduce 从版本 1.3 开始都会默认支持 Sqoop 组件，您无需再自行安装，可以跳过本节。

若您使用的 E-MapReduce 的版本低于 1.3，则还没有集成 Sqoop，如果需要使用请参考如下的安装方式。

1. 从官网下载 Sqoop 1.4.6 版本

   若下载的 *sqoop-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz* 无法打开，也可以使用镜像地址 *http://mirror.bit.edu.cn/apache/sqoop/1.4.6/sqoop-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz* 进行下载，请执行如下命令：

   ```
   wget http://mirror.bit.edu.cn/apache/sqoop/1.4.6/sqoop-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz
   ```

2. 执行如下命令，将下载下来的 *sqoop-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz* 解压到 Master 节点上：

   ```
   tar zxf sqoop-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz
   ```

3. 要从 MySQL 导数据需要安装 MySQL driver。请从官网下载最新版本，或执行如下命令进行下载（此处以版本 5.1.38 为例）：

   ```
   wget https://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-5.1.38.tar.gz
   ```

4. 解压后将 jar 包放到 Sqoop 目录下的 lib 目录下。

数据传输

常见的使用场景如下所示：

- MySQL -> HDFS
- HDFS -> MySQL
- MySQL -> Hive
- Hive -> MySQL

- 使用 SQL 作为导入条件

> **注意:**
> 在执行如下的命令前，请先切换你的用户为 Hadoop。

```
su hadoop
```

- 从 MySQL 到 HDFS

  在集群的 Master 节点上执行如下命令：

```
sqoop import --connect jdbc:mysql://<dburi>/<dbname> --username <
username> --password <password> --table <tablename> --check-column
 <col> --incremental <mode> --last-value <value> --target-dir <hdfs-
dir>
```

  参数说明：

  - dburi：数据库的访问连接，例如：*jdbc:mysql://192.168.1.124:3306/* 如果您的访问连接中含有参数,那么请用单引号将整个连接包裹住，例如：*jdbc:mysql://192.168.1.124:3306/mydatabase?useUnicode=true*
  - dbname：数据库的名字，例如：user。
  - username：数据库登录用户名。
  - password：用户对应的密码。
  - tablename：MySQL 表的名字。
  - col：要检查的列的名称。
  - mode：该模式决定Sqoop如何定义哪些行为新的行。取值为 append 或 lastmodified。
  - value：前一个导入中检查列的最大值。
  - hdfs-dir：HDFS 的写入目录，例如：*/user/hive/result*。

  更加详细的参数使用请参考 Sqoop Import。

- 从 HDFS 到 MySQL

  需要先创建好对应 HDFS 中的数据结构的 MySQL 表，然后在集群的 Master 节点上执行如下命令，指定要导的数据文件的路径：

  ```
  sqoop export --connect jdbc:mysql://<dburi>/<dbname> --username <
  username> --password <password> --table <tablename> --export-dir <
  hdfs-dir>
  ```

  参数说明：

  - dburi：数据库的访问连接，例如：*jdbc:mysql://192.168.1.124:3306/*。如果您的访问连接中含有参数,那么请用单引号将整个连接包裹住，例如：*jdbc:mysql://192.168.1.124:3306/mydatabase?useUnicode=true*
  - dbname：数据库的名字，例如：user。
  - username：数据库登录用户名。
  - password：用户对应的密码。
  - tablename：MySQL 的表的名字。
  - hdfs-dir：要导到 MySQL 去的 HDFS 的数据目录，例如：*/user/hive/result*。

  更加详细的参数使用请参考 Sqoop Export。

- 从 MySQL 到 Hive

  在集群的 Master 节点上执行如下命令后，从MySQL数据库导入数据的同时，也会新建一个 Hive 表：

  ```
  sqoop import --connect jdbc:mysql://<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-dir <hdfs-
dir> --hive-table <hive-tablename>
```

参数说明：

- dburi：数据库的访问连接，例如：*jdbc:mysql://192.168.1.124:3306/* 如果您的访问连接中含有参数,那么请用单引号将整个连接包裹住，例如：*jdbc:mysql://192.168.1.124:3306/mydatabase?useUnicode=true*
- dbname：数据库的名字，例如：user。
- username：据库登录用户名。
- password：用户对应的密码。
- tablename：MySQL 的表的名字。
- col：要检查的列的名称。
- mode：该模式决定Sqoop如何定义哪些行为新的行。取值为append或lastmodified。由Sqoop导入数据到Hive不支持append模式。
- value：前一个导入中检查列的最大值。
- hdfs-dir：要导到 MySQL 去的 HDFS 的数据目录，例如：*/user/hive/result*。
- hive-tablename：对应的 Hive 中的表名，可以是 xxx.yyy。

更加详细的参数使用请参考 Sqoop Import。

- 从 Hive 到 MySQL

  请参考上面的从 HDFS 到 MySQL 的命令，只需要指定 Hive 表对应的 HDFS 路径就可以了。

- 从 MySQL 到 OSS

  类似从 MySQL 到 HDFS，只是 —target-dir 不同。在集群的 Master 节点上执行如下命令：

```
sqoop import --connect jdbc:mysql://<dburi>/<dbname> --username <
username> --password <password> --table <tablename> --check-column <
col> --incremental <mode> --last-value <value> --target-dir <oss-dir
> --temporary-rootdir <oss-tmpdir>
```

注意:

- OSS 地址中的 host 有内网地址、外网地址和 VPC 网络地址之分。如果用经典网络，需要指定内网地址，杭州是 *oss-cn-hangzhou-internal.aliyuncs.com*，VPC 要指定 VPC 内网，杭州是 *vpc100-oss-cn-hangzhou.aliyuncs.com*。

- 目前同步到 OSS 不支持—delete-target-dir，用这个参数会报错 Wrong FS。如果要覆盖以前目录的数据，可以在调用 sqoop 前用hadoop fs -rm -r osspath先把原来的 OSS 目录删了。

```
sqoop import --connect jdbc:mysql://<dburi>/<dbname> --username <
username> --password <password> --table <tablename> --check-column <
col> --incremental <mode> --last-value <value> --target-dir <oss-dir
> --temporary-rootdir <oss-tmpdir>
```

参数说明：

- dburi：数据库的访问连接，例如：*jdbc:mysql://192.168.1.124:3306/* 如果您的访问连接中含有参数,那么请用单引号将整个连接包裹住，例如：*jdbc:mysql://192.168.1.124:3306/mydatabase?useUnicode=true*
- dbname：数据库的名字，例如：user。
- username：数据库登录用户名。
- password：用户对应的密码。
- tablename：MySQL 表的名字。
- col：要检查的列的名称。
- mode：该模式决定Sqoop如何定义哪些行为新的行。取值为append或lastmodified。
- value：前一个导入中检查列的最大值。
- oss-dir：OSS 的写入目录，例如：*oss://<accessid>:<accesskey>@<bucketname>.oss-cn-hangzhou-internal.aliyuncs.com/result*。
- oss-tmpdir：临时写入目录。指定 append 模式的同时，需要指定该参数。如果目标目录已经存在于 HDFS 中，则 Sqoop 将拒绝导入并覆盖该目录的内容。采用append模式后，Sqoop 会将数据导入临时目录，然后将文件重命名为正常目标目录。

更加详细的参数使用请参见 Sqoop Import。

- 从 OSS 到 MySQL

  类似 MySQL 到 HDFS，只是 —export-dir 不同。需要创建好对应 OSS 中的数据结构的 MySQL 表

  然后在集群的 Master 节点上执行如下：指定要导的数据文件的路径：

  ```
  sqoop export --connect jdbc:mysql://<dburi>/<dbname> --username <
  username> --password <password> --table <tablename> --export-dir <
  oss-dir>
  ```

  参数说明：

  - dburi：数据库的访问连接，例如：*jdbc:mysql://192.168.1.124:3306/* 如果您的访问连接中含有参数,那么请用单引号将整个连接包裹住，例如：*jdbc:mysql://192.168.1.124:3306/mydatabase?useUnicode=true*
  - dbname：数据库的名字，例如：user。
  - username：数据库登录用户名。
  - password：用户对应的密码。
  - tablename：MySQL 表的名字。
  - oss-dir：OSS 的写入目录，例如：*oss://<accessid>:<accesskey>@<bucketname>.oss-cn-hangzhou-internal.aliyuncs.com/result*。
  - oss-tmpdir：临时写入目录。指定 append 模式的同时，需要指定该参数。如果目标目录已经存在于 HDFS 中，则Sqoop将拒绝导入并覆盖该目录的内容。采用 append 模式后，Sqoop 会将数据导入临时目录，然后将文件重命名为正常目标目录。

  > 说明:
  > OSS 地址 host 有内网地址，外网地址，VPC 网络地址之分。如果 用经典网络，需要指定内网地址，杭州是 *oss-cn-hangzhou-internal.aliyuncs.com*，VPC 需要指定 VPC 内网，杭州是 *vpc100-oss-cn-hangzhou.aliyuncs.com*。

  更加详细的参数使用请参见，Sqoop Export

- 使用 SQL 作为导入条件

  除了指定 MySQL 的全表导入，还可以写 SQL 来指定导入的数据，如下所示：

  ```
  sqoop import --connect jdbc:mysql://<dburi>/<dbname> --username <
  username> --password <password> --query <query-sql> --split-by <sp
  ```

```
-column> --hive-import --hive-table <hive-tablename> --target-dir <
hdfs-dir>
```

参数说明：

- dburi：数据库的访问连接，例如：*jdbc:mysql://192.168.1.124:3306/* 如果您的访问连接中含有参数,那么请用单引号将整个连接包裹住，例如：*jdbc:mysql://192.168.1.124:3306/mydatabase?useUnicode=true*
- dbname：数据库的名字，例如：user。
- username：数据库登录用户名。
- password：用户对应的密码。
- query-sql：使用的查询语句，例如：`SELECT * FROM profile WHERE id>1 AND \$CONDITIONS`。记得要用引号包围，最后一定要带上 `AND \$CONDITIONS`。
- sp-column：进行切分的条件,一般跟 MySQL 表的主键。
- hdfs-dir：要导到 MySQL 去的 HDFS 的数据目录，例如：*/user/hive/result*。
- hive-tablename：对应的 Hive 中的表名，可以是 xxx.yyy。

更加详细的参数使用请参见 Sqoop Query Import。

集群和其他数据库的网络配置请参见#unique_170。

13 组件授权

13.1 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/创建文件夹等。

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

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

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)。如：

```
#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
```

13.2 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。

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(如 HDFS 的 permission 检查 /yarn 提交作业到队列控制)

说明:

一般设置了 HDFS permission 检查 /yarn 队列资源控制，可以不设置服务级别的授权控制，用户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/group，设置相关参数时，user 和 group 可同时设置，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/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/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，如可执行`yarn rmadmin/yarn kill`等命令，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/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/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。如：

    ```
    #queue-name=root
      <property>
          <name>yarn.scheduler.capacity.root.acl_submit_applicati
    ons</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/
    testgrp组提交作业
      </property>
    ```

- yarn.scheduler.capacity.${queue-name}.acl_administer_queue

  - 设置某些 user/group 管理队列，比如 kill 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_admini
    ster_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。
- 配置示例
  - yarn-site.xml

```
<property>
        <name>yarn.acl.enable</name>
        <value>true</value>
 </property>
<property>
        <name>yarn.admin.acl</name>
        <value>has</value>
 </property>
```

  - 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（如 kill）。
  - 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.DefaultRes
ourceCalculator</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_admini
ster_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_applicati
ons</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_applicati
ons</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>
```

13.3 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）两种授权机制。

 说明:

两种授权机制可以同时配置，不冲突。

Storage Based Authorization （针对 HiveMetaStore）

场景:

如果集群的用户直接通过 HDFS/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。详见 Hive 官方文档

添加配置

在集群的配置管理页面选择Hive > 配置 > hive-site.xml > 自定义配置

```
<property>
<name>hive.metastore.pre.event.listeners</name>
    <value>org.apache.hadoop.hive.ql.security.authorization.Authorizat
ionPreEventListener</value>
</property>
<property>
<name>hive.security.metastore.authorization.manager</name>
    <value>org.apache.hadoop.hive.ql.security.authorization.StorageBas
edAuthorizationProvider</value>
</property>
<property>
<name>hive.security.metastore.authenticator.manager</name>
    <value>org.apache.hadoop.hive.ql.security.HadoopDefaultMetasto
reAuthenticator</value>
</property>
```

重启 HiveMetaStore

在集群配置管理页面重启 HiveMetaStore

HDFS 权限控制

EMR 的 Kerberos 安全集群已经设置了 Hive 的 warehouse 的 HDFS 相关权限。

对于非 Kerberos 安全集群，用户需要做如下步骤设置 hive 基本的 HDFS 权限：

- 打开 HDFS 的权限
- 配置 Hive 的 warehouse 权限

```
hadoop fs -chmod 1771 /user/hive/warehouse
也可以设置成，1 表示 stick bit(不能删除别人创建的文件/文件夹)
hadoop fs -chmod 1777 /user/hive/warehouse
```

有了上述设置基础权限后，可以通过对 warehouse 文件夹授权，让相关用户/用户组能够正常创建表/读写表等：

```
sudo su has
      #授予test对warehouse文件夹rwx权限
      hadoop fs -setfacl -m user:test:rwx /user/hive/warehouse
      #授予hivegrp对warehouse文件夹rwx权限
      hadoo fs -setfacl -m group:hivegrp:rwx /user/hive/warehouse
```

经过 HDFS 的授权后，让相关用户/用户组能够正常创建表/读写表等，而且 不同的账号创建的 hive 表在 HDFS 中的数据只能自己的账号才能访问。

验证

- test 用户建表 testtbl

```
hive> create table testtbl(a string);
FAILED: Execution Error, return code 1 from org.apache.hadoop.hive
.ql.exec.DDLTask. MetaException(message:Got exception: org.apache.
hadoop.security.AccessControlException Permission denied: user=test
, access=WRITE, inode="/user/hive/warehouse/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)
```

上面显示错误没有权限，需要给 test 用户添加权限

```
#从root账号切到has账号
su has
#给test账号添加acl，对warehouse目录的rwx权限
 hadoop fs -setfacl -m user:test:rwx /user/hive/warehouse
```

test 账号再创建 database，成功

```
hive> create table testtbl(a string);
OK
Time taken: 1.371 seconds
#查看hdfs中testtbl的目录，从权限可以看出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
hadoop fs -ls /user/hive/warehouse
drwxr-x---   - test hadoop          0 2017-11-25 14:51 /user/hive/
warehouse/testtbl
#插入一条数据
```

```
hive>insert into table testtbl select "hz"
```

- foo 用户访问 testtbl

```
#drop table
hive> drop table testtbl;
FAILED: Execution Error, return code 1 from org.apache.hadoop.hive.
ql.exec.DDLTask. MetaException(message:Permission denied: user=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)
    at org.apache.hadoop.hdfs.server.namenode.FSPermissionChecker.
checkPermission(FSPermissionChecker.java:190)
#alter table
hive> alter table testtbl add columns(b string);
FAILED: SemanticException Unable to fetch table testtbl. java.
security.AccessControlException: Permission denied: user=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)
    at org.apache.hadoop.hdfs.server.namenode.FSPermissionChecker.
checkPermission(FSPermissionChecker.java:190)
    at org.apache.hadoop.hdfs.server.namenode.FSDirectory.checkPermi
ssion(FSDirectory.java:1720)
#select
hive> select * from testtbl;
FAILED: SemanticException Unable to fetch table testtbl. java.
security.AccessControlException: Permission denied: user=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)
```

可见 foo 用户不能对 test 用户创建的表做任何的操作，如果想要授权给 foo，需要通过 HDFS 的授权来实现

```
su has
#只授权读的权限，也可以根据情况授权写权限(比如alter)
#备注：-R 将testtbl文件夹下的文件也设置可读
hadoop fs -setfacl -R -m user:foo:r-x /user/hive/warehouse/testtbl
#可以select成功
hive> select * from testtbl;
OK
hz
Time taken: 2.134 seconds, Fetched: 1 row(s)
```

说明:

一般可以根据需求新建一个 hive 用户的 group，然后通过给 group 授权，后续将新用户添加到 group 中，同一个 group 的数据权限都可以访问。

SQL Standards Based Authorization

- 场景

  如果集群的用户不能直接通过 hdfs/hive client 访问，只能通过 `HiveServer (beeline/jdbc)` 等)来执行 hive 相关的命令，可以使用 SQL Standards Based Authorization 的授权方式。

  如果用户能够使用 hive shell 等方式，即使做下面的一些设置操作，只要用户客户端的 hive-site.xml 没有相关配置，都可以正常访问 hive。

  详见 Hive文档

- 添加配置

  - 配置是提供给 HiveServer
  - 在集群与服务管理页面选择 Hive > 配置 > hive-site.xml > 自定义配置

  ```
  <property>
  <name>hive.security.authorization.enabled</name>
   <value>true</value>
  </property>
   <property>
  <name>hive.users.in.admin.role</name>
   <value>hive</value>
  </property>
   <property>
  <name>hive.security.authorization.createtable.owner.grants</name>
   <value>ALL</value>
  </property>
  ```

- 重启 HiveServer2

  在集群配置管理页面重启 HiveServer2

- 权限操作命令

  具体命令操作详见Hive 文档

- 验证

  - 用户 foo 通过 beeline 访问 test 用户的 testtbl 表

  ```
  2: jdbc:hive2://emr-header-1.cluster-xxx:10> select * from testtbl
  ;
  Error: Error while compiling statement: FAILED: HiveAccess
  ControlException 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)
  ```

  - grant 权限

  ```
  切到 test 账号执行 grant 给 foo 授权 select 操作
  hive> grant select on table testtbl to user foo;
  ```

```
 OK
 Time taken: 1.205 seconds
```

- foo 可以正常 select

```
0: jdbc:hive2://emr-header-1.cluster-xxxxx:10> select * from
testtbl;
INFO  : OK
+------------+--+
| testtbl.a  |
+------------+--+
| hz         |
+------------+--+
1 row selected (0.787 seconds)
```

- 回收权限

```
切换到 test 账号，回收权限 foo 的 select 权限
  hive> revoke select from user foo;
    OK
    Time taken: 1.094 seconds
```

- foo 无法 select testtbl 的数据

```
0: jdbc:hive2://emr-header-1.cluster-xxxxx:10> select * from
testtbl;
Error: Error while compiling statement: FAILED: HiveAccess
ControlException 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)
```

13.4 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。

说明:

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，详见 Kerberos 简介。

添加配置

在 HBase 集群的集群与服务管理页面选择 HBase > 配置 > hbase-site > 自定义配置

添加如下几个参数:

```
<property>
     <name>hbase.security.authorization</name>
     <value>true</value>
</property>
<property>
     <name>hbase.coprocessor.master.classes</name>
```

```
      <value>org.apache.hadoop.hbase.security.access.AccessController</
value>
</property>
<property>
      <name>hbase.coprocessor.region.classes</name>
 <value>org.apache.hadoop.hbase.security.token.TokenProvider,org.
apache.hadoop.hbase.security.access.AccessController</value>
</property>
<property>
  <name>hbase.coprocessor.regionserver.classes</name>
  <value>org.apache.hadoop.hbase.security.access.AccessController,org.
apache.hadoop.hbase.security.token.TokenProvider</value>
</property>
```

重启 HBase 集群

在 HBase 集群的集群与服务管理页面选择 HBase > 操作 > RESTART All Components

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
    - 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
    - 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/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
  - 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/group 的授权方式一样，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
- user_permissions 查看权限

13.5 Kafka 授权

如果没有开启 Kafka 认证（如 Kerberos 认证或者简单的用户名密码），即使开启了 Kafka 授权，用户也可以伪造身份访问服务。所以建议创建高安全模式，即支持Kerberos 的 Kafka 集群，详见Kerberos 简介。

说明:

本文的权限配置只针对E-MapReduce的高安全模式集群，即Kafka以Kerberos的方式启动。

添加配置

1. 在集群管理页面，在需要添加配置的Kafka集群后单击详情。
2. 在左侧导航栏中单击集群与服务管理页签，然后单击服务列表中的 Kafka。
3. 在上方单击配置页签。
4. 在服务配置列表的右上角单击自定义配置，添加如下几个参数:

key	value	备注
authorizer.class.name	kafka.security.auth.SimpleAclAuthorizer	无
super.users	User:kafka	User:kafka 是必须的，可添加其它用户用分号(;)隔开

说明:

zookeeper.set.acl 用来设置 Kafka 在 ZooKeeper 中数据的权限，E-MapReduce 集群中已经设置为 true，所以此处不需要再添加该配置。该配置设置为 true 后，在 Kerberos 环境中，只有用户名称为 Kafka 且通过Kerberos 认证后才能执行 kafka-topics.sh 命令（kafka-topics.sh会直接读写/修改 ZooKeeper 中的数据）。

重启 Kafka 集群

1. 在集群管理页面，在需要添加配置的 Kafka 集群后单击详情。
2. 在左侧导航栏中单击集群与服务管理页签，然后单击服务列表中的 Kafka 右侧的操作。
3. 在下拉菜单中选择 RESTART All Components，输入记录信息后单击确定。

授权（ACL）

- 基本概念

  Kafka 官方文档定义：

  ```
  Kafka acls are defined in the general format of "Principal P is [Allowed/Denied] Operation O From Host H On Resource R"
  ```

  即 ACL 过程涉及 Principal、Allowed/Denied、Operation、Host 和 Resource。

  - Principal：用户名

安全协议	value
PLAINTEXT	ANONYMOUS
SSL	ANONYMOUS
SASL_PLAINTEXT	mechanism 为 PLAIN 时，用户名是 client_jaas.conf 指定的用户名，mechanism 为 GSSAPI 时，用户名为 client_jaas.conf 指定的 principal

安全协议	value
SASL_SSL	

- Allowed/Denied：允许/拒绝。
- Operation： 操作，包括 Read、Write、Create、DeleteAlter、Describe、ClusterAction、AlterConfigs、DescribeConfigs、IdempotentWrite 和 All。
- Host： 针对的机器。
- Resource： 权限作用的资源对象，包括 Topic、Group、Cluster 和 TransactionalId。

Operation/Resource 的一些详细对应关系，如哪些 Resource 支持哪些 Operation 的授权，详见 KIP-11 - Authorization Interface。

· 授权命令

我们使用脚本 kafka-acls.sh （/usr/lib/kafka-current/bin/kafka-acls.sh） 进行Kafka授权，您可以直接执行 `kafka-acls.sh --help`查看如何使用该命令。

操作示例

在已经创建的 E-MapReduce 高安全 Kafka 集群的 master 节点上进行相关示例操作。

1. 新建用户 test，执行以下命令。

```
useradd test
```

2. 创建 topic。

第一节添加配置的备注中提到 zookeeper.set.acl=true，kafka-topics.sh 需要在 kafka 账号下执行，而且 kafka 账号下要通过 Kerberos 认证。

```
# kafka_client_jaas.conf中已经设置了kafka的Kerberos认证相关信息
export KAFKA_HEAP_OPTS="-Djava.security.auth.login.config=/etc/ecm/
kafka-conf/kafka_client_jaas.conf"
# zookeeper地址改成自己集群的对应地址(执行`hostnamed`后即可获取)
kafka-topics.sh --create --zookeeper emr-header-1:2181/kafka-1.0.0
 --replication-factor 3 --partitions 1 --topic test
```

3. test 用户执行 kafka-console-producer.sh。

a. 创建 test 用户的 keytab 文件，用于 zookeeper/kafka 的认证。

```
su root
sh /usr/lib/has-current/bin/hadmin-local.sh /etc/ecm/has-conf -k /
etc/ecm/has-conf/admin.keytab
HadminLocalTool.local: #直接按回车可以看到一些命令的用法
HadminLocalTool.local: addprinc #输入命令按回车可以看到具体命令的用法
HadminLocalTool.local: addprinc -pw 123456 test #添加test的
principal,密码设置为123456
```

```
HadminLocalTool.local: ktadd -k /home/test/test.keytab test #导出
keytab文件，后续可使用该文件
```

b. 添加 kafka_client_test.conf。

如文件放到 */home/test/kafka_client_test.conf*，内容如下：

```
KafkaClient {
com.sun.security.auth.module.Krb5LoginModule required
useKeyTab=true
storeKey=true
serviceName="kafka"
keyTab="/home/test/test.keytab"
principal="test";
};
// Zookeeper client authentication
Client {
com.sun.security.auth.module.Krb5LoginModule required
useKeyTab=true
useTicketCache=false
serviceName="zookeeper"
keyTab="/home/test/test.keytab"
principal="test";
};
```

c. 添加 producer.conf。

如文件放到 `/home/test/producer.conf`，内容如下：

```
security.protocol=SASL_PLAINTEXT
sasl.mechanism=GSSAPI
```

d. 执行 kafka-console-producer.sh。

```
su test
export KAFKA_HEAP_OPTS="-Djava.security.auth.login.config=/home/
test/kafka_client_test.conf"
kafka-console-producer.sh --producer.config /home/test/producer.
conf --topic test --broker-list emr-worker-1:9092
```

由于没有设置 ACL，所以上述会报错：

```
org.apache.kafka.common.errors.TopicAuthorizationException: Not
authorized to access topics: [test]
```

e. 设置 ACL。

同样`kafka-acls.sh`也需要kafka账号执行。

```
su kafka
export KAFKA_HEAP_OPTS="-Djava.security.auth.login.config=/etc/ecm
/kafka-conf/kafka_client_jaas.conf"
```

```
kafka-acls.sh --authorizer-properties zookeeper.connect=emr-header
-1:2181/kafka-1.0.0 --add --allow-principal User:test --operation
Write --topic test
```

f. 再执行 kafka-console-producer.sh。

```
su test
export KAFKA_HEAP_OPTS="-Djava.security.auth.login.config=/home/
test/kafka_client_test.conf"
kafka-console-producer.sh --producer.config /home/test/producer.
conf --topic test --broker-list emr-worker-1:9092
```

正常情况下显示如下：

```
[2018-02-28 22:25:36,178] INFO Kafka commitId : aaa7af6d4a11b29d (
org.apache.kafka.common.utils.AppInfoParser)
>alibaba
>E-MapReduce
>
```

4. test 用户执行`kafka-console-consumer.sh`。

上面成功执行 kafka-console-producer.sh，并往 topic 里面写入一些数据后，就可以执行`kafka-console-consumer.sh`进行消费测试.

a. 添加 consumer.conf。

如文件放到*/home/test/consumer.conf*，内容如下：

```
security.protocol=SASL_PLAINTEXT
sasl.mechanism=GSSAPI
```

b. 执行 kafka-console-consumer.sh。

```
su test
#kafka_client_test.conf跟上面producer使用的是一样的
export KAFKA_HEAP_OPTS="-Djava.security.auth.login.config=/home/
test/kafka_client_test.conf"
kafka-console-consumer.sh --consumer.config consumer.conf  --topic
 test --bootstrap-server emr-worker-1:9092 --group test-group  --
from-beginning
```

由于未设置权限，会报以下错误：

```
org.apache.kafka.common.errors.GroupAuthorizationException: Not
authorized to access group: test-group
```

c. 设置 ACL。

```
su kafka
export KAFKA_HEAP_OPTS="-Djava.security.auth.login.config=/etc/ecm
/kafka-conf/kafka_client_jaas.conf"
# test-group权限
kafka-acls.sh --authorizer-properties zookeeper.connect=emr-header
-1:2181/kafka-1.0.0 --add --allow-principal User:test --operation
Read --group test-group
# topic权限
```

```
kafka-acls.sh --authorizer-properties zookeeper.connect=emr-header
-1:2181/kafka-1.0.0 --add --allow-principal User:test --operation
Read --topic test
```

d. 再执行kafka-console-consumer.sh。

```
su test
# kafka_client_test.conf跟上面producer使用的是一样的
export KAFKA_HEAP_OPTS="-Djava.security.auth.login.config=/home/
test/kafka_client_test.conf"
kafka-console-consumer.sh --consumer.config consumer.conf  --topic
 test --bootstrap-server emr-worker-1:9092 --group test-group  --
from-beginning
```

正常输出：

```
alibaba
E-MapReduce
```

13.6 RANGER

13.6.1 Ranger 简介

Apache Ranger 提供集中式的权限管理框架，可以对 Hadoop 生态中的 HDFS/Hive/YARN/Kafka/Storm/Solr 等组件进行细粒度的权限访问控制，并且提供了 Web UI 方便管理员进行操作。

创建集群

在 E-MapReduce 控制台创建 EMR-2.9.2/EMR-3.9.0 及以上的集群，勾选 Ranger 组件即可。

如果已经创建 EMR-2.9.2/EMR-3.9.0 及以上的集群，可以在集群的集群与服务管理页面添加 Ranger 服务。

说明：

- 开启 Ranger 后，设置安全控制策略之前，不会对应用程序产生影响和限制。
- Ranger 中设置的用户策略为集群 Hadoop 账号。

Ranger UI

在安装了 Ranger 的集群后，单击管理，然后在左侧导航栏中选择访问链接与端口，可以通过快捷链接访问 Ranger UI。

单击链接后会进入 Ranger UI 登录界面，默认的用户名/密码是 admin/admin。

修改密码

管理员首次登录后，需要修改 admin 的密码。

修改完 admin 密码后，单击右上角 admin，选择下拉菜单中的 Log Out，然后使用新的密码登录即可。

组件集成 Ranger

经过上述步骤，可以将集群中的相关组件集成到 Ranger，进行相关权限的控制，详情请参见：

- HDFS 配置
- #unique_180
- HBase 配置

13.6.2 HDFS 配置

前面简介中介绍了 E-MapReduce 中创建启动 Ranger 服务的集群，以及一些准备工作，本节介绍 HDFS 集成 Ranger 的一些步骤流程。

HDFS 集成 Ranger

- Enable HDFS Plugin

  1. 在集群管理页面，在需要操作的集群后单击管理
  2. 在服务列表中单击 Ranger 进入 Ranger 配置页面
  3. 在 Ranger 配置页面，单击右侧的操作，下拉菜单选择Enable HDFS PLUGIN。
  4. 在弹出框输入执行 Commit 记录，然后单击确定。

     单击右上角查看操作历史查看任务进度，等待任务完成。

- 重启 NameNode

  上述任务完成后，需要重启 NameNode 才生效。

  1. 在 Ranger 配置页面中，单击左上角 Ranger 后的倒三角，从 Ranger 切换到 HDFS。
  2. 单击右上角操作，下拉菜单中选择 RESTART NameNode。
  3. 在弹出框输入执行 Commit 记录，然后单击确定。
  4. 单击右上角的查看操作历史查看任务进度，等待重启任务完成。

- Ranger UI 添加 HDFS service

  参见Ranger 简介进入Ranger UI。

  在 Ranger UI 页面添加 HDFS Service。

  - 标准集群

    如果您创建的是标准集群（可到详情页面查看安全模式），请参考下图进行配置：

  - 高安全集群

    如果您创建的是高安全集群（可到详情页面查看安全模式），请参考下图进行配置：

权限配置示例

上面一节中已经将 Ranger 集成到 HDFS，现在可以进行相关的权限设置。例如给用户 test 授予 `/user/foo`路径的写/执行权限：

单击上图中的 emr-hdfs 进入配置页面，配置相关权限。

按照上述步骤设置添加一个 Policy 后，就实现了对 test 的授权，然后用户 test 就可以对 `/user/foo` 的 HDFS 路径进行访问了。

> 说明:
> 添加 Policy 1 分钟左右，HDFS 才会生效。

13.6.3 Hive 配置

Ranger 简介中介绍了 E-MapReduce 中创建和启动 Ranger 服务的集群，以及一些准备工作，本节介绍 Hive 集成 Ranger 的步骤流程。

Hive 集成 Ranger

- Hive 访问模型

  用户可以通过三种方式访问 Hive 数据，包括 HiveServer 2、Hive Client 和 HDFS。

  - HiveServer 2 方式
    - 场景：用户只能通过 HiveServer2 访问 Hive 数据，不能通过 Hive Client 和 HDFS 访问。
    - 方式：使用 beeline 客户端或者 JDBC 代码通过 HiveServer 2 去执行相关的 Hive 脚本。
    - 权限设置：

      Hive 官方自带的 SQL Standards Based Authorization 就是针对 HiveServer2 使用场景进行权限控制的。

      Ranger 中对 Hive 的表/列级别的权限控制也是针对 HiveServer2 的使用场景，如果用户还可以通过 Hive Client 或者 HDFS 访问 Hive 数据，仅仅对表/列层面做权限控制还不够，还需要下面两种方式的一些设置进一步控制权限。
  - Hive Client 方式
    - 场景：用户可以通过 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 进行一些 DDL 操作（如 Alter Table Add Columns等），也会通过提交 MapReduce 作业读取 HDFS 中的数据进行处理。

      Hive 官方自带的 Storage Based Authorization 就是针对 Hive Client 使用场景进行的权限控制，它会根据 SQL 中表的 HDFS 路径的读写权限来决定该用户是否可以进行相关的 DDL/DML 操作，如`ALTER TABLE test ADD COLUMNS(b STRING)`

      Ranger 中可以对 Hive 表的 HDFS 路径进行权限控制，加上 HiveMetaStore 配置 Storage Based Authorization，从而可以实现对 Hive Client 访问场景的权限控制。

      说明：

Hive Client 场景的 DDL 操作权限实际也是通过底层 HDFS 权限控制，所以如果用户有 HDFS 权限，则也会有对应表的 DDL 操作权限（如 Drop Table/Alter Table 等）。

- HDFS 方式
  - 场景： 用户可以直接访问 HDFS。
  - 方式： HDFS 客户端/代码等。
  - 权限设置:

    用户可以直接访问 HDFS，则需要对 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。

· Enable Hive Plugin

1. 在集群配置页面，在你想操作的集群后的操作栏中单击管理。
2. 在服务列表中单击 Ranger 进入 Ranger 配置界面。
3. 在 Ranger 配置页面，单击右侧的操作下拉菜单，选择 Enable Hive PLUGIN，然后单击确定。
4. 在弹出框输入备注信息，然后单击确定。

   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:

Enable Hive Plugin 并重启 Hive 后，则 HiveServer2 场景 /HiveClient 场景（Storage Based Authorization）的都进行了相关的配置，HDFS 的权限可参见 HDFS 配置进行开启。

· 重启 Hive

上述任务完成后，需要重启 Hive 才生效。

1. 在 Ranger 配置页面中，单击左上角 Ranger 后的倒三角，从 Ranger 切换到 Hive。
2. 单击右上角操作，从下拉菜单中选择 RESTART All Components，然后单击确定。
3. 单击右上角的查看操作历史查看任务进度，等待重启任务完成。

- Ranger UI添加Hive service

  参见Ranger 简介进入 Ranger UI。

  在 Ranger UI 页面添加 Hive Service。

  - 填写说明

    上述截图中相关的固定填写内容如下：

名称	值
Service Name	emr-hive
jdbc.driverClassName	org.apache.hive.jdbc.HiveDriver

  - 相关的变量值:

名称	值
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
policy.download.auth.users	hadoop（非高安全集群）/hive（高安全集群）

    ${master1_fullhost} 为 master 1 的长域名，可登录 master 1 执行`hostname`命令获取，${master1_fullhost} 中的数字即为 $id 的值。

权限配置示例

上面一节中已经将 Ranger 集成到 Hive，现在可以进行相关的权限设置。例如给用户 foo 授予表 testdb.test 的 a 列 Select 权限。

单击上图中的 emr-hive 进入配置页面，配置相关权限。

按照上述步骤设置添加一个 Policy 后，就实现了对 foo 的授权，然后用户 foo 就可以对 testdb.test 的表进行访问了。

说明:

添加 Policy 1 分钟左右后，Hive 才会生效。

13.6.4 HBase 配置

前面简介中介绍了 E-MapReduce 中创建启动 Ranger 服务的集群，以及一些准备工作，本节介绍 HBase 集成 Ranger 的一些步骤流程。

Enable HBase Plugin

1. 在集群管理页面，在你想操作的集群后的操作栏中单击管理
2. 在服务列表中单击 Ranger 进入 Ranger 配置页面
3. 在Ranger配置页面，单击右侧的操作下拉菜单，选择 Enable HBase PLUGIN。
4. 在弹出框输入执行 Commit 记录，然后单击确定
5. 单击右上角查看任务进度，等待任务完成。

Ranger UI 添加 HBase service

参见 Ranger 简介的介绍进入Ranger UI。

在 Ranger 的 UI 页面添加 Hase Service：

说明:

${id}：可登录机器执行`host`命令，hostname 中的数字即为 ${id} 的值。

重启 HBase

上述任务完成后，需要重启 HBase 才生效。重启 HBase，请按照以下步骤操作：

1. 在 Ranger 配置页面中，单击左上角 Ranger 后的倒三角，从 Ranger 切换到 HBase。
2. 单击右上角操作下拉菜单，选择 RESTART All Components。
3. 在弹出框输入执行 Commit 记录，然后单击确定。
4. 单击右上角查看操作历史查看任务进度，等待重启任务完成。

设置管理员账号

用户需要设置管理员账号的权限（admin 权限），用于执行一些管理命令，如`balance/compaction/flush/split`等等。

上图中已经存在权限策略，您只需要单击右侧的编辑按钮，将 user 改成自己想要设置的账号即可，另外也可以修改其中的权限（如只保留Admin权限）。hbase 账号必须要默认设置为管理员账号。

若使用 Phoenix，则还需在 ranger 的 HBase 中新增如下策略：

Table	SYSTEM.*
Column Family	`*`
Column	`*`
Groups	`public`
Permissions	`ReadWrite`, `Create`, `Admin`

权限配置示例

上面一节中已经将 Ranger 集成到 HBase，可以进行相关的权限设置。例如给用户 test 授予表 foo_ns:test 的`Create/Write/Read`权限。

单击上图中的 emr-hbase 进入配置页面，配置相关权限。

User/Group 会自动从集群中同步过来，大约需要一分钟，用户可以事先将它们添加到集群。

按照上述步骤设置添加一个 Policy 后，就实现了对test用户的授权，然后 test 用户就可以对 foo_ns:test 表进行访问了。

说明:

添加Policy 1 分钟左右后，HBase 才会生效。

13.6.5 Kafka 配置

从 EMR-3.12.0 版本开始，E-MapReduce Kafka 支持用 Ranger 进行权限配置。前面简介中介绍了 E-MapReduce 中创建启动 Ranger 服务的集群，以及一些准备工作，本节介绍 Kafka 集成 Ranger 的一些步骤流程。

Kafka 集成 Ranger

- Enable Kakfa Plugin

  1. 在集群管理页面，在需要操作的集群后单击管理
  2. 在服务列表中单击Ranger进入Ranger配置页面
  3. 在集群配置管理页面的 Ranger 服务下，单击右侧的操作下拉菜单，选择 Enable Kafka PLUGIN。

  4. 在弹出框输入执行 Commit 记录，然后单击确定。

     单击右上角的查看操作历史查看任务进度，等待任务完成 100%。

- 重启 Kafka broker

  上述任务完成后，需要重启 broker 才能生效。

  1. 在 Ranger 配置页面中，单击左上角 Ranger 后的倒三角，从 Ranger 切换到 Kafka。
  2. 单击右上角操作下拉菜单，选择 RESTART Broker。
  3. 在弹出框输入执行 Commit 记录，然后单击确定。
  4. 单击右上角查看操作历史查看任务进度，等待重启任务完成。

- Ranger UI 页面添加 Kafka Service

  参见 Ranger 简介的介绍进入 Ranger UI 页面。

  在 Ranger 的 UI 页面添加 Kafka Service

  配置 Kafka Service

权限配置示例

上面一节中已经将 Ranger 集成到 Kafka，现在可以进行相关的权限设置。

说明:
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：

按照上述步骤设置添加一个 Policy 后，就实现了对 test 的授权，然后用户 test 就可以对 test 的 topic 进行写入操作。

说明:
添加 Policy 需要 1 分钟左右才会生效。

13.6.6 Hive 数据脱敏

Ranger 支持对 Hive 数据的脱敏处理（Data Masking），它对 select 的返回结果进行脱敏处理，对用户屏蔽敏感信息。

说明:
该功能只针对 HiveServer2 的场景（如 beeline/jdbc/Hue 等途径执行的 select），对于使用 Hive Client（如 hive -e 'select xxxx'）不支持。

Hive 组件配置 Ranger

请参见文档： Hive 配置

配置 Data Mask Policy

在 Ranger UI 的`emr-hive`的 service 页面可以对用户访问 Hive 数据进行脱敏处理：

- 支持多种脱敏处理方式，比如显示开始的 4 个字符/显示最后的 4 个字符/Hash 处理等
- 配置 Mask Policy 时不支持通配符，例如 policy 中 table/column 不能配置为 *
- 每个 policy 只能配置一个列的 mask 策略，多个列需要配置各自的 mask policy

配置 Policy 流程如下所示：

配置完成后保存即可。

测试数据脱敏

- 场景：

  用户 test 在 select 表 testdb1.testtbl 中列 a 的数据时，只显示最开始的 4 个字符。

- 流程：

  1. 配置 policy

     在上节的最后一个截图，其实就是配置了该场景的一个 policy，可参考上图(其中脱敏方式选择了 show first 4)。

  2. 脱敏验证

     test 用户使用 beeline 连接 HiveServer2，执行`select a from testdb1.testtbl`

     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`来脱敏处理。

14 Kerberos认证

14.1 Kerberos 简介

E-MapReduce 从 2.7.x/3.5.x 版本开始支持创建安全类型的集群，即集群中的开源组件以 Kerberos 的安全模式启动，在这种安全环境下只有经过认证的客户端（Client）才能访问集群的服务（Service，如 HDFS）。

前置

目前 E-MapReduce 版本中支持的 Kerberos 的组件列表如下所示：

组件名称	组件版本
YARN	2.7.2
SPARK	2.1.1/1.6.3
HIVE	2.0.1
TEZ	0.8.4
ZOOKEEPER	3.4.6
HUE	3.12.0
ZEPPELIN	0.7.1
OOZIE	4.2.0
SQOOP	1.4.6
HBASE	1.1.1
PHOENIX	4.7.0

说明:

Kafka/Presto/Storm 目前版本不支持 Kerberos。

创建安全集群

在集群创建页面的软件配置下打开安全按钮即可。

。

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，它作为一个独立的第三方的身份认证服务，可以为其它服务提供身份认证功能，且支持 SSO （即客户端身份认证后，可以访问多个服务如 HBase/HDFS 等）。

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，如 HDFS/YARN/HBase 等
- 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。

  若身份认证通过则客户端会拿到一个 TGT（Ticket Granting Ticket），后续就可以拿该 TGT 去访问集成了 Kerberos 的服务。
- Service 对 Client 身份认证

  当 2.1 中用户拿到 TGT 后，就可以继续访问 Service 服务。它会使用 TGT 以及需要访问的服务名称(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

EMR 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
  - 登录阿里云 E-MapReduce 控制台，选择集群管理 > 管理 > Has，执行查看/修改配置/重启等操作。
  - 非 HA 集群部署在 emr-header-1，HA 集群部署在 emr-header-1/emr-header-2 两个节点

- 支持四种身份认证方式

  HasServer 可同时支持以下 4 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 hadoop.security.authentication.use.has 设置为 false。
    如果有通过控制台的执行计划跑作业，则不能修改 master 节点上面 /etc/ecm/hadoop-conf/core-site.xml 中的值，否则执行计划的作业认证就不通过而失败，可以使用下面的方式
    export HADOOP_CONF_DIR=/etc/has/hadoop-conf临时 export 环境变量，该路径下的 hadoop.security.authentication.use.has 已经设置为 false。
    ```

    访问方式：Service的客户端包完全可使用开源的，如 HDFS 客户端等。#unique_188

  - 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 hadoop.security.authentication.use.has 设置为true，/etc/has/has-client.conf中auth_type设置为RAM.
    如果有通过控制台的执行计划跑作业，则不能修改 master 节点上面 /etc/ecm/hadoop-conf/core-site.xml 以及 /etc/has/has-client.conf 中的值，否则执行计划的作业认证就不通过而失败，可以使用下面的方式
    export HADOOP_CONF_DIR=/etc/has/hadoop-conf; export HAS_CONF_DIR=/path/to/has-client.conf 临时 export 环境变量，其中 HAS_CONF_DIR 文件夹下的 has-client.conf 的auth_type设置为 RAM。
    ```

    访问方式： 客户端需要使用集群中的软件包（如 Hadoop/HBase 等），#unique_189

  - LDAP 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 hadoop.security.authentication.use.has 设置为true，/etc/has/has-client.conf 中 auth_type 设置为 LDAP。
    如果有通过控制台的执行计划跑作业，则不能修改 master 节点上面 /etc/ecm/hadoop-conf/core-site.xml 以及 /etc/has/has-client.conf 中的值，否则执行计划的作业认证就不通过而失败，可以使用下面的方式
    ```

```
export HADOOP_CONF_DIR=/etc/has/hadoop-conf; export HAS_CONF_DIR=/
path/to/has-client.conf 临时 export 环境变量，其中 HAS_CONF_DIR 文件夹
下的 has-client.conf 的 auth_type 设置为 LDAP。
```

访问方式：客户端需要使用集群中的软件包（如 Hadoop/HBase 等），LDAP 认证。

- 执行计划认证

  如果用户有使用 EMR 控制台的执行计划提交作业，则 emr-header-1 节点的配置必须不能被修改（默认配置）。

  客户端配置：

  ```
  emr-header-1 上面的 /etc/ecm/hadoop-conf/core-site.xml 中 hadoop.
  security.authentication.use.has 设置为 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，#unique_191

  。

- 其他

  登陆 master 节点访问集群

  集群管理员也可以登陆 master 节点访问集群服务，登陆 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，方便做一些排查问题或者运维等。

  ```
  >sudo su has
  >hadoop fs -ls /
  ```

  说明：
  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 MITKerberos 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/
  ```

14.2 兼容 MIT Kerberos 认证

本文将通过 HDFS 服务介绍兼容 MIT Kerberos 认证流程。

兼容 MIT Kerberos 的身份认证方式

EMR 集群中 Kerberos 服务端启动在 master 节点，涉及一些管理操作需在 master 节点（emr-header-1）的 root 账号执行。

下面以 test 用户访问 HDFS 服务为例介绍相关流程。

- Gateway 上执行`hadoop fs -ls /`
  - 配置 krb5.conf

```
Gateway 上面使用root账号
 scp root@emr-header-1:/etc/krb5.conf /etc/
```

  - 添加 principal
    - 登录集群 emr-header-1 节点（必须是 header-1，HA 不能在 header-2 上操作），切换到 root 账号。
    - 进入 Kerberos 的 admin 工具。
    -

```
  sh /usr/lib/has-current/bin/hadmin-local.sh /etc/ecm/has-conf
 -k /etc/ecm/has-conf/admin.keytab
  HadminLocalTool.local: #直接按回车可以看到一些命令的用法
  HadminLocalTool.local: addprinc #输入命令按回车可以看到具体命令的用法
  HadminLocalTool.local: addprinc -pw 123456 test #添加test的princippal，密码设置为123456
```

  - 导出 keytab 文件

    登录集群 emr-header-1（必须是 header-1，HA 不能在header-2操作），导入 keytab 文件。

    使用 Kerberos 的 admin 工具可以导出 principal 对应的 keytab 文件。

```
HadminLocalTool.local: ktadd -k /root/test.keytab test #导出keytab文件，后续可使用该文件
```

  - kinit 获取 Ticket

    在执行 hdfs 命令的客户端机器上面，如 Gateway。

    - 添加 linux 账号 test

```
useradd test
```

    - 安装 MITKerberos 客户端工具。

      可以使用 MITKerberos tools 进行相关操作（如 kinit/klist 等），详细使用方式参见 MITKerberos 文档

```
yum install krb5-libs krb5-workstation -y
```

    - 切到 test 账号执行 kinit

```
su test
#如果没有 keytab文件，则执行
kinit #直接回车
Password for test: 123456 #即可
#如有有keytab文件，也可执行
kinit -kt test.keytab test
#查看ticket
```

```
klist
```

说明:

MITKerberos 工具使用实例

- **导入环境变量生效，执行：**

```
export HADOOP_CONF_DIR=/etc/has/hadoop-conf
```

- **执行 hdfs 命令**

  获取到 Ticket 后，就可以正常执行 hdfs 命令了。

```
hadoop fs -ls /
      Found 5 items
     drwxr-xr-x     - hadoop hadoop          0 2017-11-12 14:23 /
apps
     drwx------      - hbase  hadoop           0 2017-11-15 19:40
 /hbase
     drwxrwx--t+    - hadoop hadoop         0 2017-11-15 17:51 /
spark-history
     drwxrwxrwt    - hadoop hadoop          0 2017-11-13 23:25 /tmp
     drwxr-x--t      - hadoop hadoop          0 2017-11-13 16:12 /
user
```

说明:

跑 yarn 作业，需要提前在集群中所有节点添加对应的 linux 账号（详见下文中[EMR集群添加test账号]）。

- **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,配置从emr集群上复制一份
   conf.addResource(new Path("/etc/ecm/hadoop-conf/hdfs-site.xml
"));
   conf.addResource(new Path("/etc/ecm/hadoop-conf/core-site.xml
"));
   //需要在程序所在linux账号下,提前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，配置从 emr 集群上复制一份
  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 master-1 上面执行相关命令获
取[文档前面有介绍命令]
  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());
  }
  }
```

附 pom 依赖:

```
<dependencies>
            <dependency>
                <groupId>org.apache.hadoop</groupId>
                <artifactId>hadoop-common</artifactId>
                <version>2.7.2</version>
            </dependency>
            <dependency>
                <groupId>org.apache.hadoop</groupId>
                <artifactId>hadoop-hdfs</artifactId>
                <version>2.7.2</version>
            </dependency>
        </dependencies>
```

14.3 RAM 认证

E-MapReduce（以下简称 EMR）集群中的 Kerberos 服务端除了可以支持第一种 MIT Kerberos 兼容的使用方式，也可以支持 Kerberos 客户端使用 RAM 作为身份信息进行身份认证。

RAM 身份认证

RAM产品可以创建/管理子账号，通过子账号实现对云上各个资源的访问控制。

主账号的管理员可以在 RAM 的用户管理界面创建一个子账号(子账户名称必须符合linux用户的规范)，然后将子账号的 AccessKey 下载下来提供给该子账号对应的开发人员，后续开发人员可以通过配置 AccessKey，从而通过 Kerberos 认证访问集群服务。

使用 RAM 身份认证不需要像第一部分 MIT Kerberos 使用方式一样，提前在 Kerberos 服务端添加 principle 等操作

下面以已经创建的子账号 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，在集群上跑 yarn 作业必须要在集群所有节点上面添加跑作业的用户账号，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：

```
sudo su hadoop
sh adduser.sh test 1 2
```

  附：adduser.sh 代码

```
#添加的账户名称
 user_name=$1
 #集群master节点个数,如HA集群有2个master
 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
```

- Gateway 管理员在 Gateway 机器上添加test用户

```
useradd test
```

- Gateway 管理员配置 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>
```

附：config_gateway_kerberos.sh 脚本代码

```
#EMR集群的emr-header-1的ip
 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
```

- **test 用户登录 Gateway 配置 AccessKey**

```
登录Gateway的test账号
 #执行脚本
 sh add_accesskey.sh test
```

附：add_accesskey.sh 脚本（修改一下 AccessKey）

```
user=$1
 if [[ `cat /home/$user/.bashrc | grep 'export AccessKey'` == "" ]];
then
 echo "
 #修改为test用户的AccessKeyId/AccessKeySecret
 export AccessKeyId=YOUR_AccessKeyId
 export AccessKeySecret=YOUR_AccessKeySecret
 " >>~/.bashrc
 else
    echo $user AccessKey has been added to .bashrc
 fi
```

- **test 用户执行命令**

经过以上步骤，test 用户可以执行相关命令访问集群服务了。

执行 hdfs 命令

```
[test@gateway ~]$ hadoop fs -ls /
  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
```

运行 hadoop 作业

```
[test@gateway ~]$ hadoop jar /usr/lib/hadoop-current/share/hadoop/
mapreduce/hadoop-mapreduce-examples-2.7.2.jar pi 10 1
```

运行 spark 作业

```
[test@gateway ~]$ spark-submit --conf spark.ui.view.acls=* --class
 org.apache.spark.examples.SparkPi --master yarn-client --driver-
memory 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
```

14.4 LDAP 认证

E-MapReduce 集群还支持基于 LDAP 的身份认证，通过 LDAP 来管理账号体系，Kerberos 客户端使用 LDAP 中的账号信息作为身份信息进行身份认证。

LDAP 身份认证

LDAP 账号可以是和其它服务共用，比如 Hue 等，只需要在 Kerberos 服务端进行相关配置即可。用户可以使用 EMR 集群中已经配置好的 LDAP 服务（ApacheDS），也可以使用已经存在的 LDAP 服务，只需要在 Kerberos 服务端进行相关配置即可。

下面以集群中已经默认启动的 LDAP 服务（ApacheDS）为例：

- Gateway 管理对基础环境进行配置（跟第二部分RAM中的一致，如果已经配置可以跳过）

  区别的地方只是 */etc/has/has-client.conf* 中的 *auth_type* 需要改为 LDAP

  也可以不修改 */etc/has/has-client.conf*，用户 test 在自己的账号下拷贝一份该文件进行修改 *auth_type*，然后通过环境变量指定路径，如：

  ```
  export HAS_CONF_DIR=/home/test/has-conf
  ```

- EMR 控制台配置 LDAP 管理员用户名/密码到 Kerberos 服务端（HAS）

  进入 EMR 控制台集群的配置管理-HAS软件下，将 LDAP 的管理员用户名和密码配置到对应的 *bind_dn* 和 *bind_password* 字段，然后重启 HAS 服务。

  此例中，LDAP 服务即 EMR 集群中的 ApacheDS 服务，相关字段可以从 ApacheDS 中获取。

- EMR 集群管理员在 LDAP 中添加用户信息
  - 获取 ApacheDS 的 LDAP 服务的管理员用户名和密码在 EMR 控制台集群的配置管理 / ApacheDS 的配置中可以查看 *manager_dn* 和 *manager_password*
  - 在 ApacheDS 中添加 test 用户名和密码：

```
登录集群emr-header-1节点root账号
 新建test.ldif文件,内容如下:
 dn: cn=test,ou=people,o=emr
 objectclass: inetOrgPerson
 objectclass: organizationalPerson
 objectclass: person
 objectclass: top
 cn: test
 sn: test
 mail: test@example.com
 userpassword: test1234
 #添加到LDAP, 其中-w 对应修改成密码manager_password
 ldapmodify -x -h localhost -p 10389 -D "uid=admin,ou=system" -w "
Ns1aSe" -a -f test.ldif
 #删除test.ldif
 rm test.ldif
```

  将添加的用户名/密码提供给 test 使用

- 用户 test 配置 LDAP 信息

```
登录Gateway的test账号
 #执行脚本
 sh add_ldap.sh test
```

  附： add_ldap.sh 脚本

```
user=$1
 if [[ `cat /home/$user/.bashrc | grep 'export LDAP_'` == "" ]];then
 echo "
 #修改为test用户的LDAP_USER/LDAP_PWD
 export LDAP_USER=YOUR_LDAP_USER
 export LDAP_PWD=YOUR_LDAP_USER
 " >>~/.bashrc
 else
    echo $user LDAP user info has been added to .bashrc
 fi
```

- 用户 test 访问集群服务

  执行 hdfs 命令

```
[test@iZbp1cyio18s5ymggr7yhrZ ~]$ hadoop fs -ls /
  17/11/19 13:33:33 INFO client.HasClient: The plugin type is: LDAP
  Found 4 items
  drwxr-x---   - has    hadoop          0 2017-11-18 21:12 /apps
  drwxrwxrwt   - hadoop hadoop          0 2017-11-19 13:33 /spark-
history
  drwxrwxrwt   - hadoop hadoop          0 2017-11-19 12:41 /tmp
```

```
drwxrwxrwt   - hadoop hadoop          0 2017-11-19 12:41 /user
```

跑 Hadoop/Spark 作业等

14.5 执行计划认证

E-MapReduce 集群支持执行计划认证，用户可以通过主账号授权子账号进行执行计划的访问。

主账号访问

在主账号登陆 E-MapReduce 控制台的情况下，在执行计划页面运行相关执行计划，将作业提交到安全集群上面执行，以 hadoop 用户访问作业中涉及的相关开源组件服务。

子账号访问

在 RAM 子账号登陆 E-MapReduce 控制台的情况下，在执行计划页面运行相关执行计划，将作业提交到安全集群上面执行，以 RAM 子账号对应的用户名访问作业中涉及的相关开源组件服务。

示例

- 主账号管理员根据需求创建多个子账号（如 A/B/C），参见#unique_195在 RAM 控制台页面给子账号授予 AliyunEMRFullAccess 的权限后，子账号就能正常登陆并使用 E-MapReduce 控制台上的相关功能。
- 主账号管理员将子账号提供给相关开发人员
- 开发人员完成作业的创建/执行计划的创建,然后启动运行执行计划提交作业到集群运行，最终在集群上面以该子账号对应的用户名（如 A/B/C）去访问相关的组件服务。

  说明:
  周期调度的执行计划目前统一以 hadoop 账号执行
- 组件服务使用子账户的用户名进行相关的权限控制，如子账户 A 是否有权限访问 hdfs 中的某个文件等。

14.6 跨域互信

E-MapReduce 中的 Kerberos 支持跨域访问（cross-realm），即不同的 Kerberos 集群之间可以互相访问。

下面以 Cluster-A 跨域去访问 Cluster-B 中的服务为例：

- Cluster-A 的 emr-header-1 的 hostname -> emr-header-1.cluster-1234 ; realm -> EMR.1234.COM
- Cluster-B的 emr-header-1 的 hostname -> emr-header-1.cluster-6789 ; realm -> EMR.6789.COM

- 说明:
  - hostname 可以在 emr-header-1 上面执行命令 hostname 获取
  - realm 可以在 emr-header-1 上面的 /etc/krb5.conf 获取

添加 principal

Cluster-A 和 Cluster-B 两个集群的 emr-header-1 节点分别执行以下完全一样的命令:

```
# root账号
       sh /usr/lib/has-current/bin/hadmin-local.sh /etc/ecm/has-conf -
k /etc/ecm/has-conf/admin.keytab
       HadminLocalTool.local: addprinc -pw 123456 krbtgt/EMR.6789.COM@
EMR.1234.COM
```

说明:

- 123456 是密码,可自行修改
- EMR.6789.COM 是 Cluster-B 的 realm，即被访问的集群的 realm
- EMR.1234.COM 是 Cluster-A 的 realm，即发起访问的集群 realm

配置 Cluster-A 的 /etc/krb5.conf

在 Cluster-A 集群上配置 [realms]/[domain_realm]/[capaths]，如下所示:

```
[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 = 10.81.49.3:88
    }
    EMR.6789.COM = {
                kdc = 10.81.49.7:88
    }
[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 = .
    }
```

将上述 /etc/krb5.conf 同步到 Cluster-A 所有节点

将 Cluster-B 节点的 /etc/hosts 文件中绑定信息（只需要长域名 emr-xxx-x.cluster-xxx）拷贝复制到 Cluster-A 的所有节点 /etc/hosts

```
10.81.45.89  emr-worker-1.cluster-xxx
 10.81.46.222  emr-worker-2.cluster-xx
 10.81.44.177  emr-header-1.cluster-xxx
```

说明:

- Cluster-A 上面如果要跑作业访问 Cluster-B，需要先重启 yarn
- Cluster-A 的所有节点配置 Cluster-B 的 host 绑定信息

访问 Cluster-B 服务

在 Cluster-A 上面可以用 Cluster-A 的 Kerberos 的 keytab 文件 /ticket 缓存，去访问 Cluster-B 的服务。

如访问 Cluster-B 的 hdfs 服务:

```
su has;
hadoop fs -ls hdfs://emr-header-1.cluster-6789:9000/
Found 4 items
-rw-r-----   2 has    hadoop         34 2017-12-05 18:15 hdfs://emr-
header-1.cluster-6789:9000/abc
drwxrwxrwt   - hadoop hadoop          0 2017-12-05 18:32 hdfs://emr-
header-1.cluster-6789:9000/spark-history
drwxrwxrwt   - hadoop hadoop          0 2017-12-05 17:53 hdfs://emr-
header-1.cluster-6789:9000/tmp
drwxrwxrwt   - hadoop hadoop          0 2017-12-05 18:24 hdfs://emr-
header-1.cluster-6789:9000/user
```