ALIBABA CLOUD

# 阿里云

## Databricks 数据洞察
## 最佳实践

文档版本：20220601

阿里云

# 法律声明

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

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

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

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

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

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

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

# 通用约定

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

# 目录

1.Databricks Delta vs Open-Source Delta Lake 05
2.Databricks数据洞察 vs Open-Source Delta Lake功能对比 09
3.使用Spark Structured Streaming实时处理Kafka数据 11
4.OSS数据权限隔离 16
5.Databricks Delta Lake数据入湖最佳实践 19
6.通过JDBC连接Spark Thrift Server提交Spark作业 24

# 1.Databricks Delta vs Open-Source Delta Lake

本文介绍Databricks数据洞察产品中Databricks Runtime Delta和社区开源版本Delta Lake在性能优化方面的差异点。

## Performance Optimization

### 1. Compaction

Delta Lake on Databricks can improve the speed of read queries from a table by coalescing small files into larger ones.

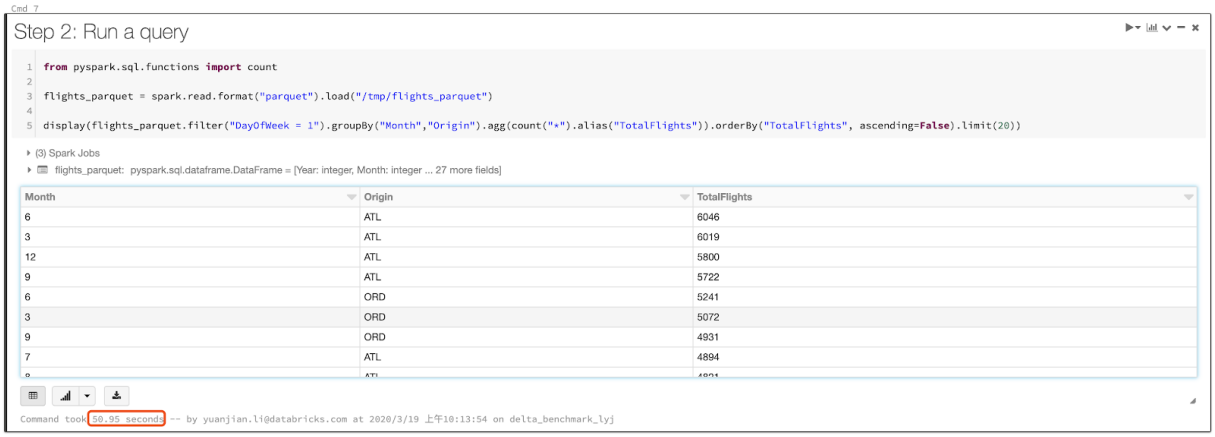
Official Document: https://docs.databricks.com/delta/optimizations/file-mgmt.html#compaction-bin-packing

Syntax

```
OPTIMIZE delta.`/data/events`
Or
OPTIMIZE events
Specify an optional partition predicate
OPTIMIZE events WHERE date >= '2017-01-01'
```

```
# Databricks notebook source
# DBTITLE 1,Clean up Parquet tables
#  %fs rm -r /tmp/flights_parquet
# DBTITLE 1,Clean up Databricks Delta tables
#  %fs rm -r /tmp/flights_delta
# DBTITLE 1,Step 0: Read flights data
flights = spark.read.format("csv") \
  .option("header", "true") \
  .option("inferSchema", "true") \
  .load("/databricks-datasets/asa/airlines/2008.csv")
# DBTITLE 1,Step 1: Write a Parquet based table using flights data
flights.write.format("parquet").mode("overwrite").partitionBy("Origin").save("/tmp/flights_
parquet")
#  Once step 1 completes, the "flights" table contains details of US flights for a year.
#  Next in Step 2, we run a query that get top 20 cities with the highest monthly total fli
ghts on the first day of week.
# DBTITLE 1,Step 2: Run a query
from pyspark.sql.functions import count
flights_parquet = spark.read.format("parquet").load("/tmp/flights_parquet")
display(flights_parquet.filter("DayOfWeek = 1").groupBy("Month","Origin").agg(count("*").al
ias("TotalFlights")).orderBy("TotalFlights", ascending=False).limit(20))
#  Once step 2 completes, you can observe the latency with the standard "flights_parquet" t
able.
#  In step 3 and step 4, we do the same with a Databricks Delta table. This time, before ru
nning the query, we run the `OPTIMIZE` command to ensure data is optimized for faster retri
eval.
# DBTITLE 1,Step 3: Write a Databricks Delta based table using flights data
flights.write.format("delta").mode("overwrite").partitionBy("Origin").save("/tmp/flights_de
lta")
# DBTITLE 1,Step 3 Continued: OPTIMIZE the Databricks Delta table
display(spark.sql("DROP TABLE  IF EXISTS flights"))
display(spark.sql("CREATE TABLE flights USING DELTA LOCATION '/tmp/flights_delta'"))
display(spark.sql("OPTIMIZE flights"))
# DBTITLE 1,Step 4 : Rerun the query from Step 2 and observe the latency
flights_delta = spark.read.format("delta").load("/tmp/flights_delta")
display(flights_delta.filter("DayOfWeek = 1").groupBy("Month","Origin").agg(count("*").alia
s("TotalFlights")).orderBy("TotalFlights", ascending=False).limit(20))
# The query over the Databricks Delta table runs much faster after `OPTIMIZE` is run.
# How much faster the query runs can depend on the configuration of the cluster you are run
ning on,
# however should be **15-20X faster**compared to the standard parquet table.
```

Benchmark



| Test Environment | Query over Parquet Table | Query over Delta Table after `OPTIMIZE` is run |
|---|---|---|
| **Attached cluster:**<br>delta_benchmark_lyj<br>91.50 GB \| 12 Cores \| DBR 6.4 \| Spark 2.4.5 \| Scala 2.11 | 50.95 seconds | 1.93 seconds |

## 2. Data Skipping

Data skipping information is collected automatically when you write data into a Delta table. Delta Lake on Databricks takes advantage of this information (minimum and maximum values) at query time to provide faster queries. You do not need to configure data skipping - the feature is activated whenever applicable.

Official Document: https://docs.databricks.com/delta/optimizations/file-mgmt.html#compaction-bin-packing

Blog and User Cases: https://databricks.com/blog/2018/07/31/processing-petabytes-of-data-in-seconds-with-databricks-delta.html?_ga=2.29295480.552083878.1584501563-968665100.1584501563

## 3. Z-Ordering

Z-Ordering is a technique

Syntax

```
OPTIMIZE events
WHERE date >= current_timestamp() - INTERVAL 1 day
ZORDER BY (eventType)
```

## Demo

```
All the script for data generation and test code can be found
in: https://drive.google.com/drive/u/1/folders/1ZQKoku9zFoC4qwYY3jlKlhYlvF8I2SpI
-- Dataset schema:
CREATE TABLE `conn_zorder` (`src_ip` STRING, `src_port`INT, `dst_ip` STRING, `dst_port` INT
)
USING delta
OPTIONS (
  `serialization.format`
  '1',
  path 'hdfs://rhodium-tests-1:8020/tmp/data/random/conn_zorder')
-- Optimize command:
OPTIMIZE '${baseLocation + connZorder}' ZORDER BY(src_ip, src_port, dst_ip, dst_port)
-- Query Tested:
select count(*) from conn_zorder where src_ip like '157%'and dst_ip like '216.%'
```

## Benchmark

| Environment Config | |
|---|---|
| VM type | 11 X Standard L16s |
| Local Disk (HDFS) per VM | 2.8TB |
| CPU per VM | 16 |
| RAM per VM | 128GB |
| Yarn Mem per node | 120GB |
| Yarn CPU per node | 15 vcpu |
| Total Yarn Mem | 1.2TB |
| Total Yarn vCPU | 158 |
| Driver mem | 8GB |
| Spark executors | 47 |
| Cores per executor | 3 |
| Mem per executor | 23GB |
| Shuffle partitions | 429 |
| spark.databricks.io.cache.maxDiskUsage | 500g |
| spark.databricks.io.cache.maxMetaDataCache | 10g |
| Yarn Resource Calculator | DominantResourceCalculator |
| **Iterations** | **2** |

| Product | Data Set Size | Runtime of Predicate Query on optimized Columns (s) | DBR Delta faster than OSS Parquet by factor of |
|---|---|---|---|
| DBR Delta | 10B Row, 1M files (267GB) | 11 | 16.9090909 |
| OSS Parquet | 10B Row, 1M files (267GB) | 186 | |

# 2.Databricks数据洞察 vs Open-Source Delta Lake功能对比

本文提供Databricks数据洞察中的Databricks Runtime Delta与社区开源版本Delta Lake。

## Databricks Runtime vs Apache Spark

下表中的 feature 列表来自 Databricks 官网(https://databricks.com/spark/comparing-databricks-to-apache-spark

| Feature | Apache Spark | Databricks数据洞察 |
| --- | --- | --- |
| Built-in file system optimized for cloud storage access | No | Yes |
| Spark-native fine grained resource sharing for optimum utilization | No | Yes |
| Fault isolation of compute resources | No | Yes |
| Faster writes to OSS | No | Yes |
| Compute optimization during joins and filters | No | Yes |
| Rapid release cycles | No | Yes |
| Auto-scaling compute | No | Yes |

## Databricks Delta vs Open-source Delta Lake

| Feature | Open SourceDelta Lake | Databricks Delta |
| --- | --- | --- |
| Snapshot Isolation / Transactional Guarantees | Yes | Yes |
| Efficient directory / File listing | Yes | Yes |
| Version history and time travel | Yes | Yes |
| Schema evolution & enforcement | Yes | Yes |
| Hidden partitions / Partitioning by expressions | In Roadmap | In Roadmap |
| HDFS Support | Yes | Yes |
| Object Storage Support | Yes | Yes |
| Streaming Data Sink | Yes | Yes |

| Feature | Open SourceDelta Lake | Databricks Delta |
| --- | --- | --- |
| Streaming Data Source | Yes | Yes |
| Basic Upsert (merge into) | Yes | Yes |
| Scalable Upsert (merge into) | No | Yes |
| Data skipping based on stats | No | Yes |
| Compact small files | Yes | Yes |
| Optimize (efficiently compact small files) | No | Yes |
| Auto Optimize | No | Yes |
| Native Parquet Reader | No | Yes |
| Read from Presto | Yes | Yes |
| Read from Hive | In Roadmap | In Roadmap |

# 3.使用Spark Structured Streaming实时处理Kafka数据

本文介绍如何使用阿里云 Dat abricks 数据洞察创建的集群去访问外部数据源 E-MapReduce，并运行Spark Structured Streaming作业以消费Kaf ka数据。

## 前提条件

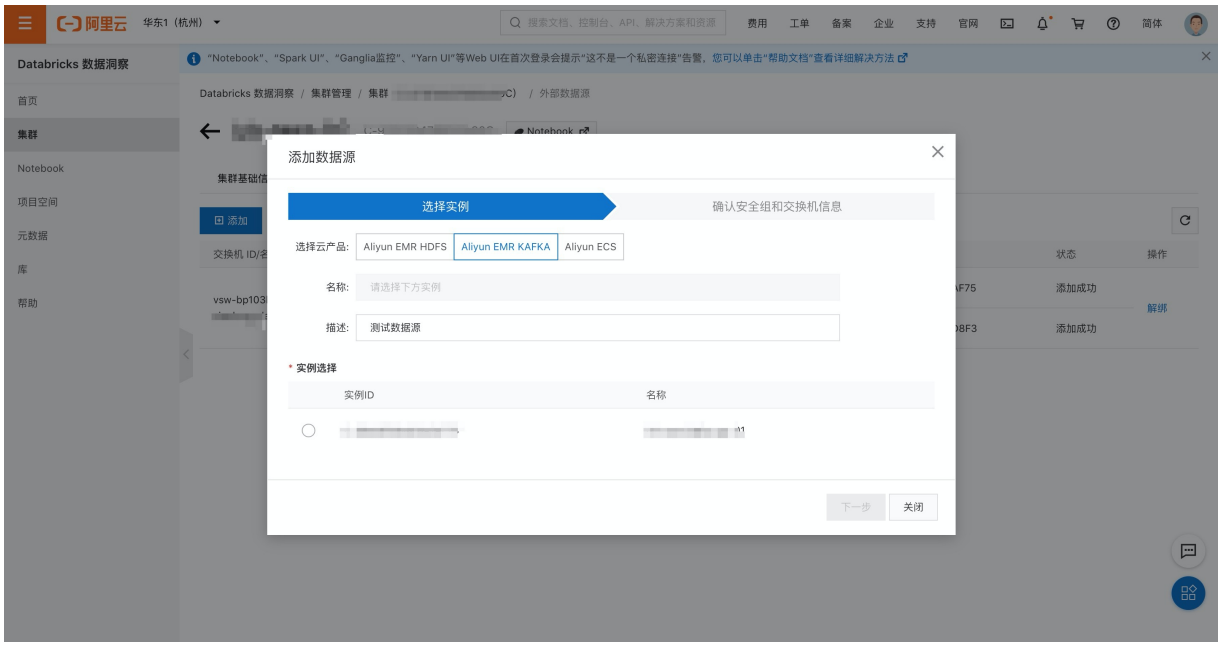
- 已注册阿里云账号，详情请参见阿里云账号注册流程。
- 已开通 E-MapReduce服务。
- 已开通对象存储 OSS服务。
- 已开通 Dat abricks数据洞察服务。

## 步骤一：创建Kafka集群和Databricks 数据洞察集群

1. 登录阿里云E-MapReduce控制台。
2. 创建Kaf ka集群。
3. 登录Dat abricks数据洞察控制台。
4. 创建集群，详情参见创建集群。

## 步骤二：Databricks 数据洞察集群添加外部数据源

1. 登录Dat abricks数据洞察控制台。
2. 单击左侧**集群**按钮，选择已创建的集群。
3. 进入集群详情页面，单击上方**数据源**按钮。
4. 在数据源页面，单击**添加**按钮，选择Aliyun EMR KAFKA
5. 填入**描述**，选择**kaf ka集群**。



## 步骤三：获取JAR包并上传到对象存储 OSS

1. 登录OSS管理控制台。
2. 创建Bucket存储空间，详情请参见存储空间。
3. 针对创建Databricks版本请选择性获取对应的Jar(DBR7+兼容高版本DBR)。
   i. （SparkStreaming-1.0-SNAPSHOT-DBR7+.jar）
   ii. (SparkStreaming-1.0-SNAPSHOT-DBR5.jar)
4. 上传JAR，详情请参见上传文件。

## 步骤四：在Kafka集群上创建Topic

本示例将创建一个分区数为10、副本数为2、名称为test的Topic。

1. 登录Kafka集群的Master节点，详情请参见使用SSH连接主节点。
2. 通过如下命令创建Topic。

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

> **说明**
>
> 创建Topic后，请保留该登录窗口，后续步骤仍将使用。

## 步骤五：运行Spark Structured Streaming作业

1. 登录Databricks数据洞察控制台。
2. 新建项目空间，详情请参见新建项目。
3. 在所属项目空间创建作业，详情请参见管理作业。
4. 执行如下作业命令，进行流式单词统计。

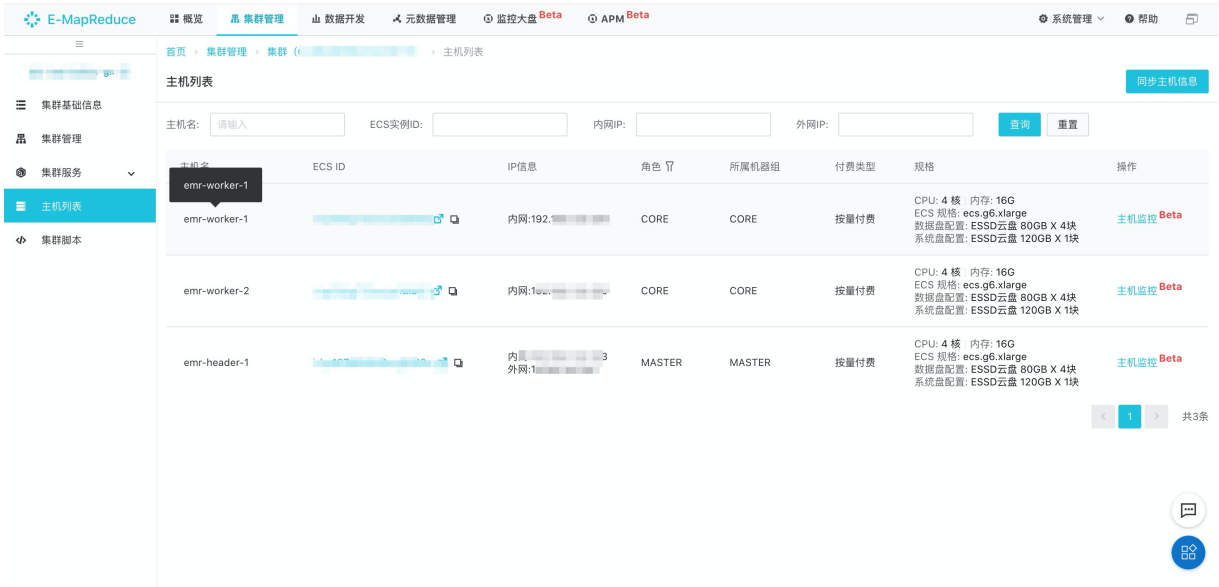
```
--deploy-mode cluster   --class demo.action.AppKafkaSparkStreaming oss://xxx/xxx/SparkStreaming-1.0-SNAPSHOT.jar
"oss://your bucket/checkpoint_dir" "192.168.xxx.xxx:9092" "log_test" "oss://your bucket/dataOutputPath"
```

关键参数说明如下：

| 参数 | 说明 |
| --- | --- |
| oss://xxx/xxx/spark-kafka-sample-1.0-SNAPSHOT-jar-with-dependencies.jar | oss对象存储上JAR包的位置 |
| 192.168.xxx.xxx:9092 | Kafka集群中任一Kafka Broker组件的内网IP地址。IP地址如下图所示。 |

| 参数 | 说明 |
| --- | --- |
| log_test | Topic名称 |
| oss://your bucket/dataOutputPath | oss对象存储上数据写入目录 |

查看Kafka集群IP




## 步骤六：使用Kafka发布消息

1. 在Kafka集群的命令行窗口，执行如下命令运行Kafka的生产者。

```
#日志结构
40.198.97.68 - - [19/Mar/2021:18:26:12 +0800] "GET /category/office HTTP/1.1" 200 58 "/item
/electronics/975" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.7 (KHTML, like Gecko
) Chrome/16.0.912.77 Safari/535.7"
104.75.85.35 - - [19/Mar/2021:18:26:12 +0800] "GET /item/networking/4019 HTTP/1.1" 200 118
"-" "Mozilla/5.0 (iPhone; CPU iPhone OS 5_0_1 like Mac OS X) AppleWebKit/534.46 (KHTML, lik
e Gecko) Version/5.1 Mobile/9A405 Safari/7534.48.3"
```

```
/usr/lib/kafka-current/bin/kafka-console-producer.sh --topic log_test --broker-list emr-wor
ker-1:9092
```

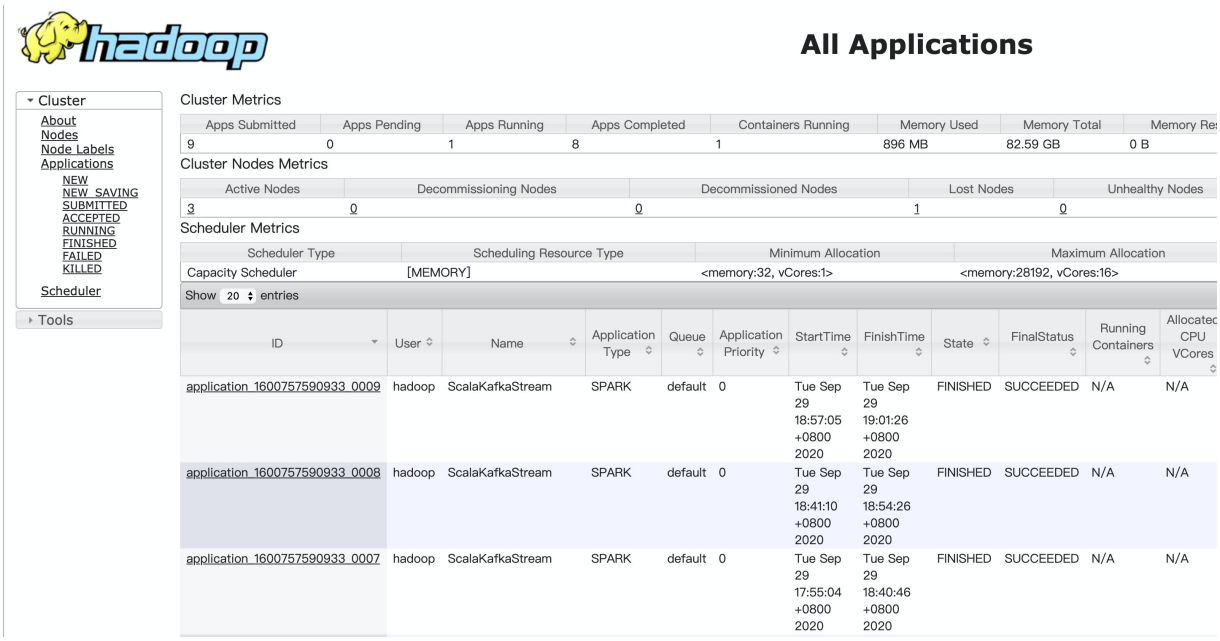
在命令行中输入4条数据

```
[root@emr-header-1 ~]# /usr/lib/kafka-current/bin/kafka-console-producer.sh --topic log-generator-topic --broker-list emr-worker-1:9092
>40.198.97.68 - - [19/Mar/2021:18:26:12 +0800] "GET /category/office HTTP/1.1" 200 58 "/item/electronics/975" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit
/535.7 (KHTML, like Gecko) Chrome/16.0.912.77 Safari/535.7"
>40.198.97.68 - - [19/Mar/2021:18:26:12 +0800] "GET /category/office HTTP/1.1" 200 58 "/item/electronics/975" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit
/535.7 (KHTML, like Gecko) Chrome/16.0.912.77 Safari/535.7"
>40.198.97.68 - - [19/Mar/2021:18:26:12 +0800] "GET /category/office HTTP/1.1" 200 58 "/item/electronics/975" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit
/535.7 (KHTML, like Gecko) Chrome/16.0.912.77 Safari/535.7"
>40.198.97.68 - - [19/Mar/2021:18:26:12 +0800] "GET /category/office HTTP/1.1" 200 58 "/item/electronics/975" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit
/535.7 (KHTML, like Gecko) Chrome/16.0.912.77 Safari/535.7"
```

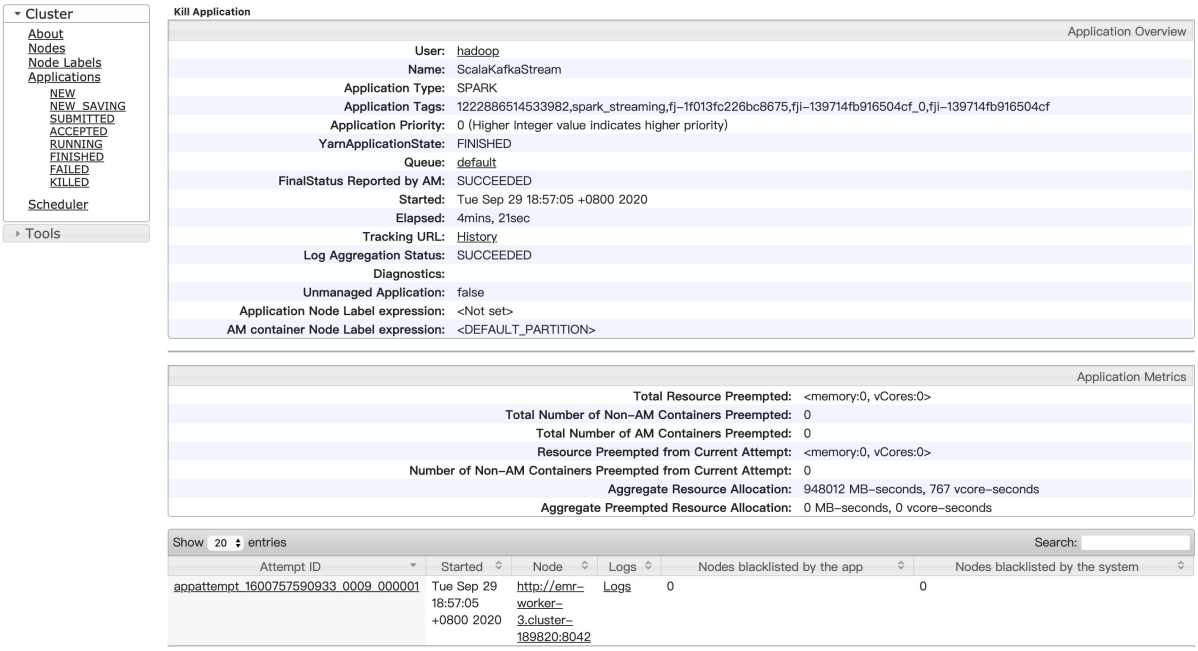
## 步骤七：查看Spark作业执行情况

通过Yarn UI查看Spark Structured Streaming作业的信息，详情请参见集群 Web UI。

1. 在Hadoop控制台，单击作业的ID。




详细信息如下。



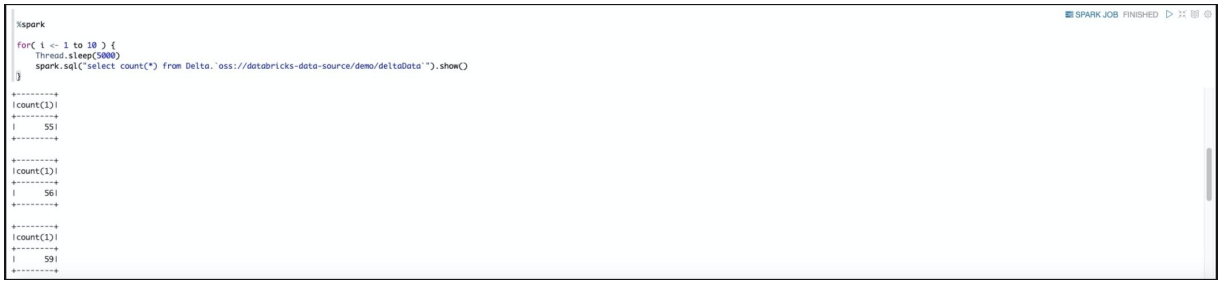

点击Logs，查看详细信息。

## 步骤八：查看数据写入情况

1. 进入Databricks 数据洞察所在集群的Notebook页面，Notebook使用详情参见（Notebook使用说明）
2. 执行代码查询数据的写入情况

```
%spark
for( i <- 1 to 10 ) {
    Thread.sleep(5000)
    spark.sql("select count(*) from Delta.`oss://your bucket/dataOutputPath`").show()
}
```

查询数据成功写入4条数据



# 4.OSS数据权限隔离

本文介绍如何使用访问控制RAM（Resource Access Management），对不同子账号的OSS数据进行隔离。

## 操作步骤

1. 云账号登录RAM控制台。
2. 创建RAM用户。
3. 新建权限策略。
   i. 在左侧导航栏中，**权限管理 > 权限策略管理**
   ii. 单击**创建权限策略。**
   iii. 填写**策略名称**
   iv. 选择**脚本配置**

   脚本配置方法请参见语法结构编辑策略内容。 本例分别按照以下两个脚本示例来创建两个权限策略：

| 测试环境（test-bucket） | 生产环境（prod-bucket） |
| --- | --- |
| `{ "Version": "1", "Statement": [ { "Effect": "Allow", "Action": [ "oss:ListBuckets" ], "Resource": [ "acs:oss:*:*:*" ] }, { "Effect": "Allow", "Action": [ "oss:Listobjects", "oss:GetObject", "oss:PutObject", "oss:DeleteObject" ], "Resource": [ "acs:oss:*:*:test-bucket", "acs:oss:*:*:test-bucket/*" ] } ] }` | `{ "Version": "1", "Statement": [ { "Effect": "Allow", "Action": [ "oss:ListBuckets" ], "Resource": [ "acs:oss:*:*:*" ] }, { "Effect": "Allow", "Action": [ "oss:Listobjects", "oss:GetObject", "oss:PutObject" ], "Resource": [ "acs:oss:*:*:prod-bucket", "acs:oss:*:*:prod-bucket/*" ] } ] }` |

按上述脚本示例进行权限隔离后，RAM用户在Databricks 数据洞察控制台的权限如下：

a. 在创建集群、创建作业和创建工作流的OSS文件页面，可以看到所有的bucket，但是只能进入被授权的bucket。

b. 只能看到被授权的bucket下的内容，无法看到其他bucket内的内容。

c. 作业中只能读写被授权的bucket，读写未被授权的bucket会报错。

4. 为RAM授权
   i. 单击左侧导航栏的**人员管理 > 用户**。
   ii. 单击待授权RAM用户所在行的**添加权限**。
   iii. 单击需要授予RAM用户的权限策略，单击**确定**。
   iv. 单击**完成**。完成授权后，权限立即生效。

# 5.Databricks Delta Lake数据入湖最佳实践

使用Spark Structured Streaming完成客户日志数据写入Delta Lake。

## 本章架构图

## 步骤一：创建Kafka集群和Databricks 数据洞察集群

1. 登录阿里云E-MapReduce控制台。
2. 创建Kafka集群，详情参见创建集群
3. 登录Databricks数据洞察控制台。
4. 创建集群，详情参见创建集群。

## 步骤二：Databricks 数据洞察集群添加外部数据源

1. 登录Databricks数据洞察控制台。
2. 单击左侧**集群**按钮，选择已创建的集群。
3. 进入集群详情页面，单击上方**数据源**按钮。
4. 在数据源页面，单击**添加**按钮，选择Aliyun EMR KAFKA
5. 填入**描述**，选择**kafka集群**。

## 步骤三：在Kafka集群上创建Topic

本示例将创建一个分区数为1、副本数为2、名称为log-generator-topic。

1. 登录Kafka集群的Master节点，详情请参见使用SSH连接主节点。
2. 通过如下命令创建Topic。

```
/usr/lib/kafka-current/bin/kafka-topics.sh --partitions 1 --replication-factor 2 --zookeeper emr-header-1:2181 /kafka-1.0.0 --topic log-generator-topic --create
```

> **说明**
>
> 创建Topic后，请保留该登录窗口，后续步骤仍将使用。

## 步骤四：执行Spark Structured Streaming作业

示例Note下载：Case-Steaming2DeltaLake

日志文件下载：access.log

1. 在Notebook中引入第三方库详情参见Java库管理

```
%spark.conf
spark.jars.packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.0.1
```

2. 创建数据库。

```
%sql
--创建数据库
create database log_data_warehouse;
--使用创建的库
use log_data_warehouse;
```

3. 根据生成的日志结果创建分区表。

说明

Apache日志发生器，生成的日志结构。

```
#日志结构
40.198.97.68 - - [19/Mar/2021:18:26:12 +0800] "GET /category/office HTTP/1.1" 200 58 "/
item/electronics/975" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.7 (KHTML, li
ke Gecko) Chrome/16.0.912.77 Safari/535.7"
104.75.85.35 - - [19/Mar/2021:18:26:12 +0800] "GET /item/networking/4019 HTTP/1.1" 200
118 "-" "Mozilla/5.0 (iPhone; CPU iPhone OS 5_0_1 like Mac OS X) AppleWebKit/534.46 (KH
TML, like Gecko) Version/5.1 Mobile/9A405 Safari/7534.48.3"
```

```
%sql
CREATE TABLE apache_logs(
ipaddr STRING,
identity STRING,
username STRING,
accesstime STRING,
request STRING,
status STRING,
size STRING,
referrer STRING,
agent STRING,
year string,
month string,
day string
)
using delta
PARTITIONED BY(year,month ,day )
```

4. Notebook执行Spark Structured Streaming 作业

```
%spark
import org.apache.spark.sql.functions._
//定义执行Structured Streaming的方法
def getquery(checkpoint_dir:String,tableName:String,servers:String,topic:String ) {
// 加载Kafka数据配置项startingOffsets=latest;
var streamingInputDF =
  spark.readStream
    .format("kafka")
    .option("kafka.bootstrap.servers", servers)
    .option("subscribe", topic)
    .option("startingOffsets", "latest")
    .option("minPartitions", "10")
    .option("failOnDataLoss", "true")
    .load()
// 定义日志的字段类型
val resDF=streamingInputDF
    .select(col("value").cast("string"))
    .withColumn("newMessage",split(col("value"), " "))
    .filter(col("newMessage").getItem(8).isNotNull)
    .select(
        col("newMessage").getItem(0).as("ipaddr"),
        col("newMessage").getItem(1).as("identity"),
        col("newMessage").getItem(2).as("username"),
        col("newMessage").getItem(3).as("accesstime"),
        col("newMessage").getItem(4).as("request"),
        col("newMessage").getItem(5).as("status"),
        col("newMessage").getItem(6).as("size"),
        col("newMessage").getItem(7).as("referrer"),
        col("newMessage").getItem(8).as("agent")
    )
    .withColumn("year",date_format(current_date(),"yyyy"))
    .withColumn("month",date_format(current_date(),"MM"))
    .withColumn("day",date_format(current_date(),"dd"))
// 将流数据动态写入到apache_logs表里
val query = resDF
      .writeStream
      .outputMode("append")
      .format("delta")
      .option("checkpointLocation", checkpoint_dir)
      .table(tableName)
  query
}
```

5. 执行作业

```
%spark
val my_checkpoint_dir="your checkpiont path"
val tableName="apache_logs"
val servers= "your kafka server"
val topic= "log-generator-topic"
val flowquery =getquery(my_checkpoint_dir,tableName,servers,topic)
```

## 步骤五：使用Kafka发布消息

1. 在Kafka集群的命令行窗口，执行如下命令运行Kafka的生产者。

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

在命令行中输入数据

```
40.198.97.68 - - [19/Mar/2021:18:26:12 +0800] "GET /category/office HTTP/1.1" 200 58 "/item/electronics/975" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.7 (KHTML, like Gecko) Chrome/16.0.912.77 Safari/535.7"
104.75.85.35 - - [19/Mar/2021:18:26:12 +0800] "GET /item/networking/4019 HTTP/1.1" 200 118 "-" "Mozilla/5.0 (iPhone; CPU iPhone OS 5_0_1 like Mac OS X) AppleWebKit/534.46 (KHTML, like Gecko) Version/5.1 Mobile/9A405 Safari/7534.48.3"
```

2.向Kafka集群手动写入三条测试数据

```
[root@emr-header-1 ~]#
[root@emr-header-1 ~]# /usr/lib/kafka-current/bin/kafka-console-producer.sh --topic log-generator-topic --broker-list emr-worker-1:9092
>40.198.97.68 - - [19/Mar/2021:18:26:12 +0800] "GET /category/office HTTP/1.1" 200 58 "/item/electronics/975" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.7 (KHTML, like Gecko) Chrome/16.0.912.77 Safari/535.7"
>40.198.97.68 - - [19/Mar/2021:18:26:12 +0800] "GET /category/office HTTP/1.1" 200 58 "/item/electronics/975" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.7 (KHTML, like Gecko) Chrome/16.0.912.77 Safari/535.7"
>40.198.97.68 - - [19/Mar/2021:18:26:12 +0800] "GET /category/office HTTP/1.1" 200 58 "/item/electronics/975" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.7 (KHTML, like Gecko) Chrome/16.0.912.77 Safari/535.7"
```

## 步骤六：查看结果

1. 进入Databricks数据洞察Notebook，动态查看数据写入情况

```
%spark
for( i <- 1 to 3 ) {
    Thread.sleep(5000)
    spark.sql("select count(1) from apache_logs").show()
}
```

查看数据写入情况

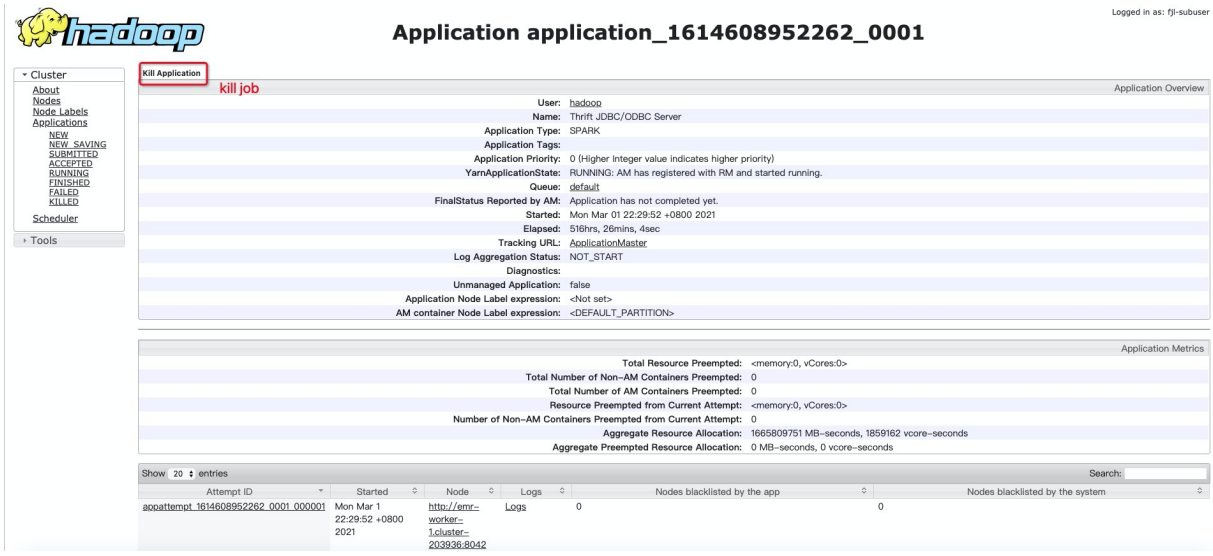
查询写入表的count数成功写入三条。

```
%spark                                                    SPARK JOB FINISHED
for( i <- 1 to 3 ) {
    Thread.sleep(5000)
    spark.sql("select count(1) from apache_logs").show()
}
+--------+
|count(1)|
+--------+
|  363240|
+--------+

+--------+
|count(1)|
+--------+
|  363243|
+--------+

+--------+
|count(1)|
+--------+
|  363243|
+--------+

Took 1 min 1 sec. Last updated by fjl-subuser at March 22 2021, 8:41:40 PM.
```

## 步骤七： 使用Yarn Applications UI页面查看作业详情或kill job

通过Yarn UI查看Spark Structured Streaming作业的信息，详情请参见集群 Web UI。

1. 进入running，选择需要kill的application。


Logged in as: fjl-subuser
Application application_1614608952262_0001
Cluster
About
Nodes
Node Labels
Applications
NEW
NEW_SAVING
SUBMITTED
ACCEPTED
RUNNING
FINISHED
FAILED
KILLED
Scheduler
Tools
Kill Application
kill job
Application Overview
User: hadoop
Name: Thrift JDBC/ODBC Server
Application Type: SPARK
Application Tags:
Application Priority: 0 (Higher Integer value indicates higher priority)
YarnApplicationState: RUNNING: AM has registered with RM and started running.
Queue: default
FinalStatus Reported by AM: Application has not completed yet.
Started: Mon Mar 01 22:29:52 +0800 2021
Elapsed: 516hrs, 26mins, 4sec
Tracking URL: ApplicationMaster
Log Aggregation Status: NOT_START
Diagnostics:
Unmanaged Application: false
Application Node Label expression: <Not set>
AM container Node Label expression: <DEFAULT_PARTITION>
Application Metrics
Total Resource Preempted: <memory:0, vCores:0>
Total Number of Non-AM Containers Preempted: 0
Total Number of AM Containers Preempted: 0
Resource Preempted from Current Attempt: <memory:0, vCores:0>
Number of Non-AM Containers Preempted from Current Attempt: 0
Aggregate Resource Allocation: 1665809751 MB-seconds, 1859162 vcore-seconds
Aggregate Preempted Resource Allocation: 0 MB-seconds, 0 vcore-seconds
Show 20 entries
Search:
Attempt ID
Started
Node
Logs
Nodes blacklisted by the app
Nodes blacklisted by the system
appattempt_1614608952262_0001_000001
Mon Mar 1 22:29:52 +0800 2021
http://emr-worker-1.cluster-203936:8042
Logs
0
0

# 6.通过JDBC连接Spark Thrift Server提交Spark作业

本文介绍通过JDBC连接Spark Thrift Servert并成功提交Spark作业。

## 前提条件

- 连接Spark Thrift Server需要校验用户名和密码，请进行用户认证配置，请参见：用户管理
- DDI集群Spark Thrift Server默认端口号为10001，请确认成功添加安全组白名单，请参见：安全组白名单

## 背景信息

JDBC连接Spark Thrift Server如下：

- Beeline：通过HiveServer2的JDBC客户端进行连接。
- Java：编写Java代码进行连接。
- Python：编写Python代码进行连接。

## Beeline客户端连接Spark Thrift Server

1. 执行如下命令，进入Beeline客户端。

```
beeline
```

返回如下信息

```
Beeline version 2.3.7 by Apache Hive
```

2. 执行如下命令，连接Spark Thrift Servert。

```
!connect jdbc:hive2://{ddi-header-ip}:10001/{db_name}
```

3. 输入用户名和密码。

```
Enter username for jdbc:hive2://ip:10001/beijing_dlf_db_test: username
Enter password for jdbc:hive2://ip:10001/beijing_dlf_db_test: ********
```

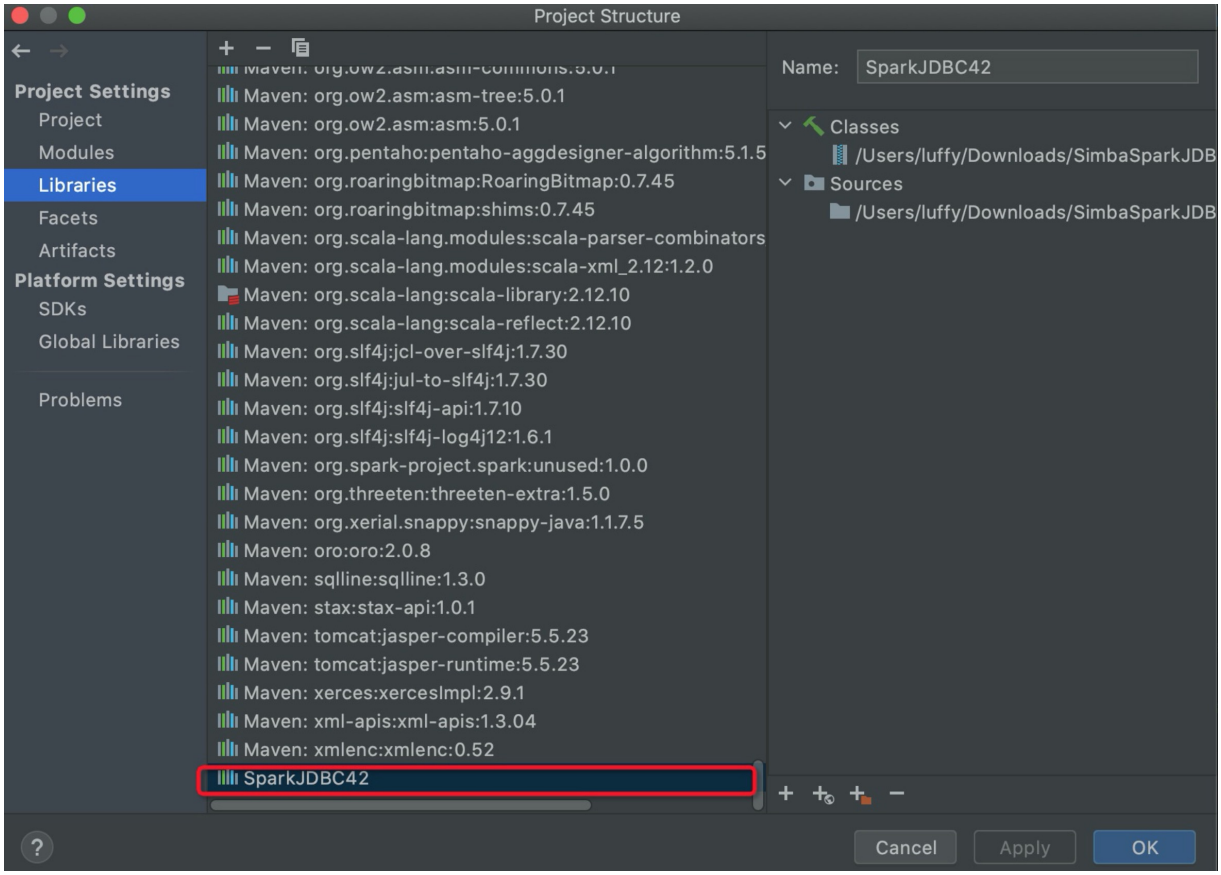
4. 查询数据，返回结果如下：

```
0: jdbc:hive2://ip:10001/beijing_d> select * from table_name limit 10;
+---------+-------------+
| action  |    date     |
+---------+-------------+
| Close   | 2016-07-27  |
| Open    | 2016-07-27  |
| Close   | 2016-07-27  |
| Close   | 2016-07-27  |
| Open    | 2016-07-27  |
| Close   | 2016-07-27  |
| Open    | 2016-07-27  |
| Open    | 2016-07-27  |
| Open    | 2016-07-27  |
| Open    | 2016-07-27  |
+---------+-------------+
10 rows selected (15.853 seconds)
```

## Java代码连接Spark Thrift Server

- 在执行本操作前，确保您已安装Java环境和Java编程工具，并且已配置环境变量
- Java代码连接Spark Thrift Server需要下载Dat abricks提供的依赖包，下载路径：Dat abricks JDBC Driver

1. 将项目依赖SparkJDBC42.jar添加到编程工具的Lib下，如图：




2. 编写代码，连接Spark Thrift Server并提交作业。

代码如下：

```
import com.simba.spark.jdbc.DataSource;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class SparkJDBC {
    private static Connection connectViaDS() throws Exception {
        Connection connection = null;
        Class.forName("com.simba.spark.jdbc.Driver");
        DataSource ds = new com.simba.spark.jdbc.DataSource();
        ds.setURL("jdbc:spark://{ip}:10001/{db_name};AuthMech=3;UID=username;PWD=ps");
        connection = ds.getConnection();
        return connection;
    }

    public static void main(String[] args) throws Exception {
        Connection connection = connectViaDS();
        System.out.println("日志打印: "+connection.getClientInfo());
        PreparedStatement tables = connection.prepareStatement("show databases ");
        ResultSet res = tables.executeQuery();
        while (res.next()){
            System.out.println("database_name :"+ res.getString(1));
        }
        res.close();
        tables.close();
        connection.close();
    }
}
```

本地测试是否正常执行。

> **注意**
>
> 执行代码过程中如果出现NoSuchFileException请联系DDI运维人员。

## Python代码连接Spark Thrift Server

1. 使用PyHive连接Spark Thrift Server需要依的赖包安装如下：

```
pip install sasl
pip install thrift
pip install thrift-sasl
pip install PyHive
```

2. 编写Python代码，连接Spark Thrift Server并提交作业。

代码如下： `vim sparkJDBC.py`

```
from pyhive import hive
conn = hive.Connection(host='host',database='db_name', port=10001, username='username',pass
word='ps',auth="LDAP")
cursor=conn.cursor()
cursor.execute('select * from table_name limit 10')
for result in cursor.fetchall():print(result)
conn.close()
```

3. 执行Python代码结果如下：

```
python sparkJDBC.py
```

执行结果：

```
('Close', '2016-07-27')
('Open', '2016-07-27')
('Close', '2016-07-27')
('Close', '2016-07-27')
('Open', '2016-07-27')
('Close', '2016-07-27')
('Open', '2016-07-27')
('Open', '2016-07-27')
('Open', '2016-07-27')
('Open', '2016-07-27')
```