ALIBABA CLOUD

阿里云

数据湖分析

云原生数据湖分析（合集）

文档版本：20220706

阿里云

法律声明

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

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

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

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

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

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

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

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

目录

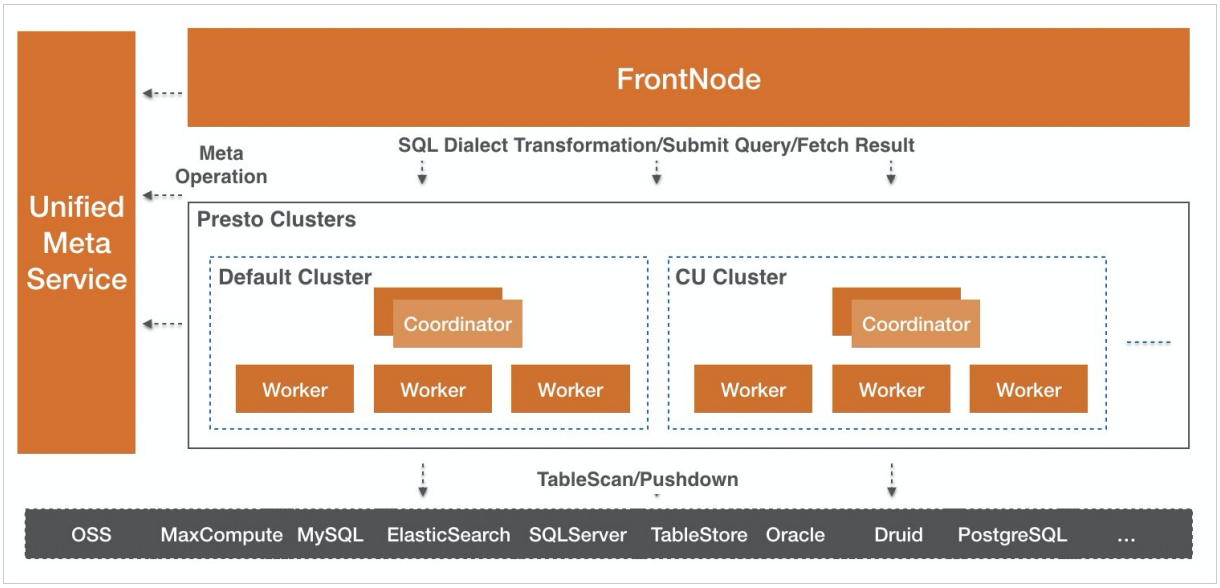
1.Serverless Presto

1.1. Serverless Presto概述

Serverless Presto是云原生数据湖团队基于Presto打造的交互式分析引擎，Presto开发的初衷就是为了解决使用Hive来进行在线分析速度太慢的问题，因此它采用全内存流水线化的执行引擎，相较于其它引擎会把中间数据落盘的执行方式，Presto在执行速度上有很大的优势，特别适合用来做Adhoc查询、BI分析、轻量级ETL等数据分析工作。

阿里云数据湖分析团队在Presto之上又进行了很多的优化，DLA支持了阿里云几乎所有的数据源比如AnalyticDB、TableStore等等；阿里云数据湖分析团队优化了Hive Connector，使得分析OSS数据时对OSS调用量大幅下降，从而提高性能且节省成本；DLA内置了企业级的权限控制体系，保护您的数据安全；内置了高可用的Coordinator方案，提高整体服务的可用性；DLA在Presto之上实现了MySQL接入协议，使得您可以使用任何兼容MySQL协议的工具来进行数据分析。

Serverless Presto的整体架构如下：



- FrontNode是用户访问的入口，它实现了MySQL协议，因此您使用任何兼容MySQL协议的客户端进行连接。
  - 您可以通过DLA的SQL执行页面直接进行库、表创建以及数据查询。
  - 您也可以使用其它客户端连接Serverless Presto服务。具体请参见创建服务访问点。

    > **说明** 登录到DMS连接数据库，需要输入数据库的账号和密码。关于如何创建账号和修改密码，请参见管理DLA账号和重置数据库账号密码

- Presto Clusters集群承担分析计算的职责。在每个Region，可以部署多个集群，一个是默认的按照扫描量计费的集群，其余的是按照CU付费的集群。
  - 您可以阅读文档计费方式概述来学习几种计费方式的差异。
  - 您可以阅读文档扫描量版本与CU版本的差异来学习扫描量版本和CU版本的具体功能差异。
  - 您可以阅读文档DLA Presto CU版本快速入门来学习如何开通、使用DLA Presto的CU版本。
  - 您可以阅读SQL参考文档学习如何创建库和表、数据查询、权限控制等其它SQL语法。每种数据源创建库、表的选项稍有不同，可以阅读连接数据源下面的文档来查看建每种数据源库表的具体写法。

- DLA的Presto Clusters集群是兼容社区Presto的，关于函数的具体定义也可以参考社区的文档：PrestoDB Functions and Operators。

- Presto Clusters集群下面可以接入各种数据源。
  - 您可以阅读连接数据源下面的文档来学习如何连接到各种数据源。

1.2. 连接数据源

1.2.1. Cassandra

云原生数据湖分析（Data Lake Analytics，DLA）支持通过CU版访问Cassandra。本文主要介绍如何通过DLA连接并查询Cassandra上的数据。

前提条件

DLA目前仅支持通过CU版访问Cassandra，请确保您已经开通了DLA CU版本，具体请参见DLA Presto CU版本快速入门。

> **说明** 创建虚拟集群时，绑定的数据源网络必须和Cassandra集群在同一个VPC下面。

操作步骤

1. 登录DLA控制台。
2. 在左侧导航栏的**Serverless Presto>SQL执行**，单击**登录DMS执行SQL**，执行以下SQL命令创建数据库和创建表。

   > **说明** 您也可以通过MySQL客户端或者程序代码等方式连接DLA，然后执行SQL命令创建库。

i. 创建数据库。

```
create database cassandra_it_db with dbproperties(
    catalog = 'cassandra',
    location = 'cds-test-1-core-001.cassandra.rds.aliyuncs.com,cds-test-1-core-002.
cassandra.rds.aliyuncs.com,cds-test-1-core-003.cassandra.rds.aliyuncs.com',
    port = '9042',
    keyspace = 'test_keyspace',
    user = 'test',
    password = 'test'
);
```

参数名称	参数说明
catalog	表示创建的是Cassandra类型的数据库。
location	需要访问的Cassandra集群的私网连接点。
port	需要访问的Cassandra集群的CQL端口。
keyspace	需要访问的Cassandra集群的keyspace。
user	需要访问的Cassandra集群的keyspace用户名。
password	需要访问的Cassandra集群的keyspace密码。

ii. 创建表。

```
create external table cassandra_it_db.test_user (
    first_name string,
    last_name string
);
```

3. 查询或访问数据。

由于只有CU的计算资源和Cassandra网络可以联通，因此所有访问Cassandra表的SQL语句都需要指定hint： `/*+cluster=your-vc-name*/` ，这样SQL才会在CU中执行。

查询数据示例如下：

```
mysql> /*+cluster=vc-test*/select * from test_user where first_name = 'james';
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| james      | bond      |
+------------+-----------+
1 row in set (1.79 sec)
```

1.2.2. OceanBase

DLA支持通过CU版访问用户自建的OceanBase，通过标准SQL语句查询OceanBase中的数据或者直接向OceanBase写入数据。本文介绍如何通过DLA读写OceanBase上的数据。

前提条件

- 目前仅支持通过CU版访问OceanBase，请确保您已开通CU版，开通方式，请参见DLA Presto CU版本快速入门。
- 虚拟集群绑定的数据源网络必须和OceanBase集群在同一个VPC下面。

准备工作

在通过DLA读写OceanBase数据前，您需要在OceanBase中创建测试表。

以下为创建OceanBase表的示例：

```
create table person (
  id bigint,
  name varchar(64),
  age int);
```

操作步骤

1. 连接DLA
2. 创建库

   ```
   CREATE DATABASE `oceanbase_test`
   WITH DBPROPERTIES (
     catalog = 'oceanbase',
     location = 'jdbc:oceanbase://oceanbaseIp:oceanbasePort/testDatabaseName',
     user = 'userName',
     password = 'password',
     vpc_id = 'vpcId',
     instance_id = 'instanceId' );
   ```

   参数说明如下：

   - CATALOG：取值为oceanbase，表示创建的是Oceanbase Schema。
   - LOCATION：填写oceanBase的地址，以及对应oceanBase中的database名。
   - USER：OceanBase数据库的用户名。
   - PASSWORD：OceanBase数据库的密码。
   - VPC_ID：OceanBase所在的网络vpcId。
   - INSTANCE_ID：部署OceanBase的instanceId。
3. 创建表

   ```
   CREATE EXTERNAL TABLE oceanbase_test.person (
     id int,
     name varchar,
     age int);
   ```

   > 注意　表名、字段的顺序和类型要和OceanBase中对应的表保持一致。

4. 访问数据由于只有CU的计算资源和OceanBase网络可以联通，因此所有访问OceanBase表的SQL语句都需要指定 `hint： /*+cluster=your-vc-name*/` ，这样SQL就会在CU中执行。

   例如：

```
mysql> /*+ cluster=vc-test */ insert into oceanbase_test.person values (1, 'james', 10)
;
+------+
| rows |
+------+
|    1 |
+------+
1 row in set (0.46 sec)
mysql> /*+ cluster=vc-test */ select id, name, age from oceanbase_test.person;
+------+-------+------+
| id   | name  | age  |
+------+-------+------+
|    1 | james |   10 |
+------+-------+------+
1 row in set (0.43 sec)
```

更多SQL信息，请参见常用SQL。

1.2.3. Kudu

DLA支持通过CU版访问用户自建的Kudu，通过标准SQL语句查询Kudu中的数据或者直接向Kudu写入数据。本文介绍如何通过DLA读写Kudu上的数据。

前提条件

- 目前仅支持通过CU版访问Kudu，请确保您已经开通了CU版。
- 虚拟集群绑定的数据源网络必须和Kudu集群在同一个VPC下面。
- 目前不支持访问开启了Kerberos认证的集群。如果您的集群开启了Kerberos认证，可以工单或钉钉咨询DLA答疑获得支持。

准备工作

通过DLA读写Kudu数据前，需要在Kudu中创建测试表。关于Kudu的表设计请参见：https://kudu.apache.org/docs/schema_design.html。

> **说明** 目前DLA不支持通过SQL在Kudu中创建新表，仅支持关联已有的表。如果需要新增一张表，需要先在Kudu中创建好，然后通过DLA的建表语句来关联。

以下代码片段是一个Java创建表的示例：

```
        String KUDU_MASTERS = "master-1:7051,master-2:7051,master-3:7051";
        String tableName = "users";
    KuduClient client = new KuduClient.KuduClientBuilder(KUDU_MASTERS).build();
    // Set up a simple schema.
    List<ColumnSchema> columns = new ArrayList<>(3);
    columns.add(new ColumnSchema.ColumnSchemaBuilder("user_id", Type.INT32)
            .key(true)
            .build());
    columns.add(new ColumnSchema.ColumnSchemaBuilder("first_name", Type.STRING).nullable(tr
ue)
            .build());
    columns.add(new ColumnSchema.ColumnSchemaBuilder("last_name", Type.STRING).nullable(tru
e)
            .build());
    Schema schema = new Schema(columns);
    // Set up the partition schema, which distributes rows to different tablets by hash.
    // Kudu also supports partitioning by key range. Hash and range partitioning can be com
bined.
    // For more information, see http://kudu.apache.org/docs/schema_design.html.
    CreateTableOptions cto = new CreateTableOptions();
    List<String> hashKeys = new ArrayList<>(1);
    hashKeys.add("user_id");
    int numBuckets = 2;
    cto.addHashPartitions(hashKeys, numBuckets);
    cto.setNumReplicas(1);
    // Create the table.
    client.createTable(tableName, schema, cto);
    System.out.println("Created table " + tableName);
```

操作步骤

1. 连接DLA。
2. 创建库

   ```
   CREATE DATABASE `kudu_test`
   WITH DBPROPERTIES (
       catalog = 'kudu',
       location = 'master-1:7051,master-2:7051,master-3:7051'
   );
   ```

   参数说明如下：

   - CATALOG：取值为kudu，表示创建的是Kudu Schema。
   - LOCATION：填写kudu master的地址，以逗号分隔。
3. 创建表

   ```
   CREATE EXTERNAL TABLE users (
   user_id int primary key,
     first_name varchar,
     last_name varchar);
   ```

> 注意　表名、字段的顺序和类型要和Kudu保持一致。

4. 访问数据

   由于只有CU的计算资源和Kudu网络可以联通，因此所有访问Kudu表的SQL语句都需要指定 `hint: /*+ cluster=your-vc-name*/` ，这样SQL就会在CU中执行。

   例如：

```
mysql> /*+ cluster=vc-test */ insert into kudu_it_db_vc.users values(1, 'Donald', 'Duck
');
+------+
| rows |
+------+
|    1 |
+------+
1 row in set (0.46 sec)
mysql> /*+ cluster=vc-test */ select user_id,first_name,last_name from kudu_it_db_vc.us
ers where user_id = 1;
+---------+------------+-----------+
| user_id | first_name | last_name |
+---------+------------+-----------+
|       1 | Donald     | Duck      |
+---------+------------+-----------+
1 row in set (0.43 sec)
```

更多SQL信息请参见：SQL参考手册。

1.2.4. HDFS

云原生数据湖分析（Data Lake Analytics，DLA）支持通过CU版访问用户自建的HDFS。本文主要介绍如何通过DLA连接并查询HDFS上的数据。

前提条件

- DLA目前仅支持通过CU版访问HDFS，请确保您已经开通了DLA CU版本，详情请参见CU版本快速入门。
- 创建虚拟集群时，绑定的数据源网络必须和HDFS集群在同一个VPC下面。
- 以下示例是一个简单的CSV文件，您可以在本地创建一个新的文本文件 `example.txt` ，在其中粘贴如下内容：

```
7,8,9
```

  然后执行如下命令将文件上传到HDFS。

```
hadoop fs -mkdir -p hdfs://172.168.199.0:9000/test/p/d=1
hadoop fs -copyFromLocal example.txt hdfs://172.168.199.0:9000/test/p/d=1/example.txt
```

  您需要把命令中 `172.168.199.0:9000` 替换成您HDFS集群的host。

> **说明**　目前不支持访问开启了Kerberos认证的集群。如果您的集群开启了Kerberos认证，可以提交工单或者钉钉咨询 DLA答疑获得支持。

操作步骤

1. 登录DLA控制台。
2. 单击左侧导航栏的**Serverless Presto>SQL执行**。
3. 单击**登录到DMS来执行SQL操作**，执行以下SQL命令创建库：

```
CREATE DATABASE `my_hdfs_db`
WITH DBPROPERTIES (
    catalog = 'hive',
    location = 'hdfs://172.168.199.0:9000/test/'
)
```

> **说明** 您也可以通过MySQL客户端或者程序代码等方式链接DLA，然后执行SQL命令创建库。

参数说明如下：

- CATALOG：取值为hive，表示创建的是hive Schema。
- LOCATION：库所在的目录。

4. 创建表。

```
CREATE EXTERNAL TABLE p (
    `a` int,
    `b` int,
    `c` int
) partitioned by (d int)
ROW FORMAT DELIMITED
    FIELDS TERMINATED BY ','
STORED AS `TEXTFILE`
LOCATION 'hdfs://172.168.199.0:9000/test/p/';
```

> **说明** HDFS表的参数、表属性和OSS表大部分都是一样的。主要的区别如下：
>
> - 由于网络连通性的问题，在创建库、表时不会对目录内容进行检查，用户需要自己保证目录的正确性。
> - 基于同样的原因，HDFS表不支持auto.create.location属性。

5. 查询或访问数据。

由于只有CU的计算资源和HDFS网络可以联通，因此所有访问HDFS表的SQL语句都需要指定 `hint: /*+ cluster=your-vc-name*/` ，这样SQL就会在CU中执行。

示例：

```
mysql> /*+ cluster=vc-test */
    -> alter table p
    -> add partition (d=1)
    -> location 'hdfs://172.168.0.6:9000/test/p/d=1';
Query OK, 0 rows affected (8.63 sec)
mysql> /*+ cluster=vc-test */
    -> alter table p
    -> drop partition (d=1) ;
Query OK, 0 rows affected (6.08 sec)
mysql> /*+ cluster=vc-test */ msck repair table p;
+-----------+-----------+
| Partition | Operation |
+-----------+-----------+
| d=1       | CREATED   |
+-----------+-----------+
1 row in set (16.47 sec)
mysql> /*+ cluster=vc-test */ select * from  p;
+------+------+------+------+
| a    | b    | c    | d    |
+------+------+------+------+
|    7 |    8 |    9 |    1 |
+------+------+------+------+
1 row in set (4.74 sec)
```

更多详细信息，请参见常用SQL。

1.2.5. HiveMetastore

云原生数据湖分析（Data Lake Analytics，DLA）支持通过CU版访问用户自建的HiveMetastore。本文主要介绍如何通过DLA连接并查询HiveMetastore里存储在HDFS的数据。

前提条件

- DLA目前仅支持通过CU版访问HiveMetastore，请确保您已经开通了DLA CU版本，具体请参见DLA Presto CU版本快速入门。

  > **说明**
  > - 创建虚拟集群时，绑定的数据源网络必须和HiveMetastore+HDFS集群在同一个VPC下面。
  > - 目前不支持访问开启了Kerberos认证的集群。如果您的集群开启了Kerberos认证，可以提交工单获取支持。

- 在HiveMetastore中已创建数据库和表，并插入数据。参考命令样例如下：

```
CREATE DATABASE testDb;
CREATE EXTERNAL TABLE if not exists testDb.testTable(
                id int,
                name string);
insert into testDb.testTable(id, name) values (1, "jack");
```

操作步骤

1. 登录DLA控制台。

2. 在左侧导航栏的**Serverless Presto>SQL执行**，单击**登录DMS执行SQL**，执行以下SQL命令创建数据库和创建表。

> **说明** 您也可以通过MySQL客户端或者程序代码等方式连接DLA，然后执行SQL命令创建库。

i. 创建数据库。

```
CREATE DATABASE `dlaDb`
WITH DBPROPERTIES (
    catalog = 'customer_hive',
    database = 'testDb',
    location = '172.16.199.34:9083',
    vpc_id = 'xxx',
    hdfs_properties = 'fs.defaultFS=hdfs://172.16.199.41:9000'
)
```

参数名称	参数说明
catalog	表示创建的数据源是用户的HiveMetastore数据，固定为customer_hive。
database	指定需要访问的数据库为用户HiveMetastore的HiveServer中的库名。
location	指定需要访问的连接点为用户HiveMetastore的HiveServer地址。
vpc_id	指定需要访问的HiveMetastore所在的vpc。

hdfs_properties	指定需要访问的HiveMetastore中默认的hdfs配置。包括以下两种形式：
	■ 非HA的HDFS，示例如下：
	`//这里的hdfs的配置为自建HDFS的地址，需要指定为IP或域名，不能为hostname。`
	`hdfs_properties='fs.defaultFS=hdfs://172.16.199.41:9000'`
	■ HA的HDFS,，示例如下：
	`//这里的hdfs的配置为自建HA HDFS的配置，需要指定为IP或域名，不能为hostname。`
	`hdfs_properties='fs.defaultFS=hdfs://emr-cluster;dfs.nameservices=emr-cluster;dfs.client.failover.proxy.provider.emr-cluster=org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider;dfs.ha.namenodes.emr-cluster=nn1,nn2;dfs.nameservice.id=emr-cluster;dfs.namenode.rpc-address.emr-cluster.nn1=172.16.199.34:8020;dfs.namenode.rpc-address.emr-cluster.nn2=172.16.199.35:8020'`

ii. 创建表。

- 通过create table来创建表。

```
CREATE EXTERNAL TABLE if not exists dlaDb.testTable(
                id int,
                name string);
```

> **说明** 表名 `testTable` 需要和HiveMetastore中对应db的表名相同。

- 通过msck来创建表。

```
msck repair database dlaDb;
```

3. 查询或访问数据。

由于只有CU的计算资源和HiveMetastore网络可以联通，因此所有访问HiveMetastore表的SQL语句都需要指定 `hint： /*+cluster=your-vc-name*/` ，这样SQL才会在CU中执行。

查询数据示例如下：

```
mysql> /*+ cluster=vc-test */ select * from  dlaDb.testTable;
+------+------+
| id   | name |
+------+------+
|    1 | jack |
+------+------+
1 row in set (1.74 sec)
```

1.2.6. Druid

Data Lake Analytics（简称DLA） 支持通过标准JDBC连接云服务器（Elastic Compute Service，简称ECS）上的自建Druid，并对其数据进行查询操作。

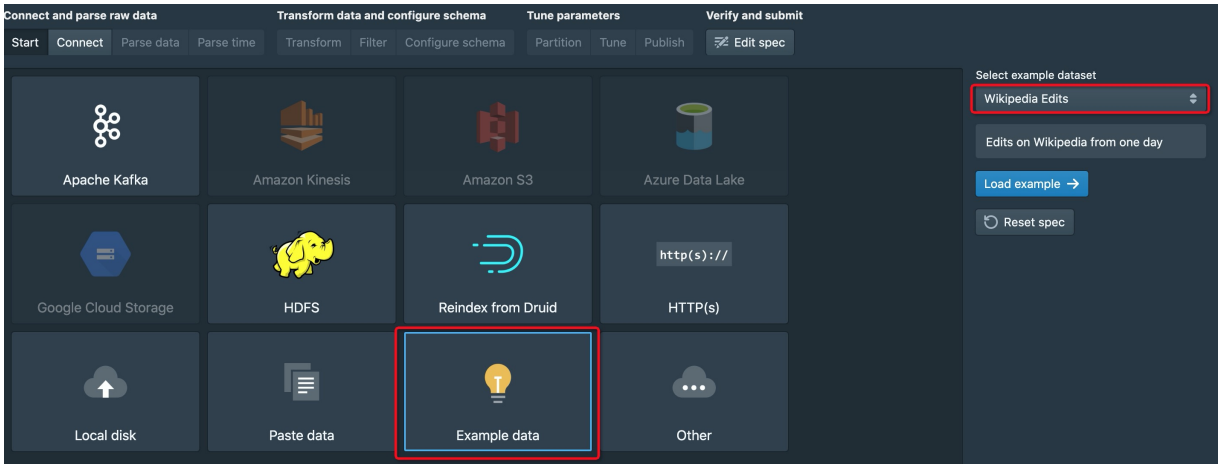
本文档介绍通过DLA连接并查询Druid数据。

前提条件

DLA将通过VPC连接ECS自建Druid，应确保DLA与ECS所属Region相同。

DLA通过JDBC接口访问Druid，需要在Druid的配置中将 `druid.sql.enable` 和 `druid.sql.avatica.enable` 这两个属性设为true。其他SQL相关参数参见Druid文档。

本文档中的示例使用Druid的示例数据Wikipedia作为测试数据，可以在Druid的Web页面上通过向导加载。



为了让DLA可以访问ECS自建Druid数据库，需要在ECS的安全组中添加安全组规则：授权100.104.0.0/16 IP地址段。

添加安全组规则时，端口选择Broker的端口（默认为8082），授权对象设置为100.104.0.0/16。

操作步骤

1. 链接DLA
2. 通过以下SQL在DLA中创建Druid Schema。

```
  CREATE DATABASE `my_druid_db`
WITH DBPROPERTIES (
 catalog = 'druid',
 location = 'jdbc:avatica:remote:url=http://BROKER:8082/druid/v2/sql/avatica/',
 user = '******',
 password = '******',
 VPC_ID = 'vpc-*****',
 INSTANCE_ID = '********'
)
```

参数说明：

- CATALOG：取值为druid，表示创建的是Druid Schema。
- LOCATION: Druid的JDBC连接地址，参见Druid文档。
- USER: Druid的用户名。
- PASSWORD: USER对应的密码。
- VPC_ID: Broker所在的VPC ID。
- INSTANCE_ID: Broker所在的ECS实例ID。

3. 通过以下SQL在my_druid_db Schema中创建wikipedia表。

```
 CREATE EXTERNAL TABLE `wikipedia` (
`__time` TIMESTAMP NULL COMMENT '',
`added` BIGINT NULL COMMENT '',
 `channel` string NULL COMMENT '',
`cityname` string NULL COMMENT '',
`comment` string NULL COMMENT '',
 `countryisocode` string NULL COMMENT '',
`countryname` string NULL COMMENT '',
`deleted` BIGINT NULL COMMENT '',
`delta` BIGINT NULL COMMENT '',
 `isanonymous` string NULL COMMENT '',
`isminor` string NULL COMMENT '',
 `isnew` string NULL COMMENT '',
`isrobot` string NULL COMMENT '',
 `isunpatrolled` string NULL COMMENT '',
`namespace` string NULL COMMENT '',
 `page` string NULL COMMENT '',
`regionisocode` string NULL COMMENT '',
 `regionname` string NULL COMMENT '',
`user` string NULL COMMENT ''
)
TBLPROPERTIES (
 TABLE_MAPPING = 'druid.wikipedia',
 COLUMN_MAPPING = 'cityname,cityName; countryisocode,countryIsoCode; countryname,countr
yName; isanonymous,isAnonymous; isminor,isMinor; isnew,isNew; isrobot,isRobot; isunpatr
olled,isUnpatrolled; regionisocode,regionIsoCode; regionname,regionName; '
);
```

也可以使用msck命令自动同步druid中的表结构：

```
 msck repair database my_druid_db;
```

表创建成功后，您就可以通过SELECT在DLA中读取Druid中的数据。

```
 select * from wikipedia;
```

1.3. 性能白皮书

1.3.1. 与开源Presto对比性能白皮书

1.3.1.1. 测试环境

本次测试针对开源自建的Presto与阿里云云原生数据湖分析DLA Presto在OSS数据源上执行查询的性能做了对比分析。本文档主要介绍了测试环境的配置要求。

环境配置要求

- 客户端ECS与服务端（Presto和DLA Presto）处于同一地域、同一可用区。本例中为华东1（杭州）可用区I。
- 客户端与服务端的网络类型均为VPC网络。
- 开源自建的Presto使用社区0.228版本。由于社区版本的Presto不支持访问OSS，需要您做如下修改来支持

访问OSS数据源：

i. 下载OSS jar包，解压后复制到Presto的*lib*和*plugin/hive-hadoop2*这两个目录下面。

ii. 在Presto的*etc/core-site.xml*中增加如下配置：

```
<property>
    <name>fs.oss.accessKeyId</name>
    <value>your ak</value>
</property>
<property>
    <name>fs.oss.accessKeySecret</name>
    <value>your sk</value>
</property>
<property>
    <name>fs.oss.credentials.provider</name>
    <value></value>
</property>
<property>
    <name>fs.oss.endpoint</name>
    <value>oss-cn-hangzhou-internal.aliyuncs.com</value>
</property>
<property>
    <name>fs.oss.impl</name>
    <value>org.apache.hadoop.fs.aliyun.oss.AliyunOSSFileSystem</value>
</property>
```

- 开源Presto集群配置如下：

配置名称	配置要求
Worker节点规格	ecs.hfg6.4xlarge（16核64 GB）
Worker节点数量	15
Coordinator节点规格	ecs.hfg6.4xlarge（16核64 GB）

- DLA使用256核1024 GB规格的虚拟集群。
- 客户端ECS使用ecs.hfg6.4xlarge（16核64 GB）规格的机型。

1.3.1.2. 测试方法

本次测试针对开源自建的Presto与阿里云云原生数据湖分析DLA Presto在OSS数据源上执行TPC-H查询的性能做了对比分析。您可以按照本文介绍自行测试对比，快速了解云原生数据湖分析（DLA）Presto引擎的性能。

1. 准备TPC-H测试数据。关于如何生成TPC-H测试数据，具体请参考TPC-H官方文档。

   TPC-H测试数据生成后，需要分别转换成Parquet和ORC格式，上传到OSS。

2. 分别在开源Presto和DLA Presto中建表。示例如下：

   对于开源Presto集群，可以在presto中建表（需要修改配置，默认不支持），也可以在hive中建表。以下是hive的建表语句。

```
CREATE SCHEMA if not exists tpch_1000x_orc
location  'oss://path/to/tpch/tpch 1000x orc/'
```

```
;
CREATE EXTERNAL TABLE tpch_1000x_orc.lineitem (
    l_orderkey bigint,
    l_partkey bigint,
    l_suppkey bigint,
    l_linenumber int,
    l_quantity double,
    l_extendedprice double,
    l_discount double,
    l_tax double,
    l_returnflag string,
    l_linestatus string,
    l_shipdate string,
    l_commitdate string,
    l_receiptdate string,
    l_shipinstruct string,
    l_shipmode string,
    l_comment string
)
STORED AS ORC
LOCATION 'oss://path/to/tpch/tpch_1000x_orc/lineitem_orc/'
;
CREATE EXTERNAL TABLE tpch_1000x_orc.customer (
    c_custkey bigint,
    c_name string,
    c_address string,
    c_nationkey int,
    c_phone string,
    c_acctbal double,
    c_mktsegment string,
    c_comment string
)
STORED AS ORC
LOCATION 'oss://path/to/tpch/tpch_1000x_orc/customer_orc/'
;
CREATE EXTERNAL TABLE tpch_1000x_orc.nation (
    n_nationkey int,
    n_name string,
    n_regionkey int,
    n_comment string
)
STORED AS ORC
LOCATION 'oss://path/to/tpch/tpch_1000x_orc/nation_orc/'
;
CREATE EXTERNAL TABLE tpch_1000x_orc.orders (
    o_orderkey bigint,
    o_custkey bigint,
    o_orderstatus string,
    o_totalprice double,
    o_orderdate string,
    o_orderpriority string,
    o_clerk string,
    o_shippriority int,
    o_comment string
```

```
)
STORED AS ORC
LOCATION 'oss://path/to/tpch/tpch_1000x_orc/orders_orc/'
;
CREATE EXTERNAL TABLE tpch_1000x_orc.part (
    p_partkey bigint,
    p_name string,
    p_mfgr string,
    p_brand string,
    p_type string,
    p_size int,
    p_container string,
    p_retailprice double,
    p_comment string
)
STORED AS ORC
LOCATION 'oss://path/to/tpch/tpch_1000x_orc/part_orc/'
;
CREATE EXTERNAL TABLE tpch_1000x_orc.partsupp (
    ps_partkey bigint,
    ps_suppkey bigint,
    ps_availqty int,
    ps_supplycost double,
    ps_comment string
)
STORED AS ORC
LOCATION 'oss://path/to/tpch/tpch_1000x_orc/partsupp_orc/'
;
CREATE EXTERNAL TABLE tpch_1000x_orc.region (
    r_regionkey int,
    r_name string,
    r_comment string
)
STORED AS ORC
LOCATION 'oss://path/to/tpch/tpch_1000x_orc/region_orc/'
;
CREATE EXTERNAL TABLE tpch_1000x_orc.supplier (
    s_suppkey bigint,
    s_name string,
    s_address string,
    s_nationkey int,
    s_phone string,
    s_acctbal double,
    s_comment string
)
STORED AS ORC
LOCATION 'oss://path/to/tpch/tpch_1000x_orc/supplier_orc/'
;
CREATE SCHEMA if not exists tpch_1000x_parquet
location  'oss://path/to/tpch/tpch_1000x_parquet/'
;
CREATE EXTERNAL TABLE tpch_1000x_parquet.lineitem (
    l_orderkey bigint,
    l_partkey bigint,
```

```
    l_suppkey bigint,
    l_linenumber int,
    l_quantity double,
    l_extendedprice double,
    l_discount double,
    l_tax double,
    l_returnflag string,
    l_linestatus string,
    l_shipdate string,
    l_commitdate string,
    l_receiptdate string,
    l_shipinstruct string,
    l_shipmode string,
    l_comment string
)
STORED AS PARQUET
LOCATION 'oss://path/to/tpch/tpch_1000x_parquet/lineitem_parquet/'
;
CREATE EXTERNAL TABLE tpch_1000x_parquet.customer (
    c_custkey bigint,
    c_name string,
    c_address string,
    c_nationkey int,
    c_phone string,
    c_acctbal double,
    c_mktsegment string,
    c_comment string
)
STORED AS PARQUET
LOCATION 'oss://path/to/tpch/tpch_1000x_parquet/customer_parquet/'
;
CREATE EXTERNAL TABLE tpch_1000x_parquet.nation (
    n_nationkey int,
    n_name string,
    n_regionkey int,
    n_comment string
)
STORED AS PARQUET
LOCATION 'oss://path/to/tpch/tpch_1000x_parquet/nation_parquet/'
;
CREATE EXTERNAL TABLE tpch_1000x_parquet.orders (
    o_orderkey bigint,
    o_custkey bigint,
    o_orderstatus string,
    o_totalprice double,
    o_orderdate string,
    o_orderpriority string,
    o_clerk string,
    o_shippriority int,
    o_comment string
)
STORED AS PARQUET
LOCATION 'oss://path/to/tpch/tpch_1000x_parquet/orders_parquet/'
;
```

```
CREATE EXTERNAL TABLE tpch_1000x_parquet.part (
    p_partkey bigint,
    p_name string,
    p_mfgr string,
    p_brand string,
    p_type string,
    p_size int,
    p_container string,
    p_retailprice double,
    p_comment string
)
STORED AS PARQUET
LOCATION 'oss://path/to/tpch/tpch_1000x_parquet/part_parquet/'
;
CREATE EXTERNAL TABLE tpch_1000x_parquet.partsupp (
    ps_partkey bigint,
    ps_suppkey bigint,
    ps_availqty int,
    ps_supplycost double,
    ps_comment string
)
STORED AS PARQUET
LOCATION 'oss://path/to/tpch/tpch_1000x_parquet/partsupp_parquet/'
;
CREATE EXTERNAL TABLE tpch_1000x_parquet.region (
    r_regionkey int,
    r_name string,
    r_comment string
)
STORED AS PARQUET
LOCATION 'oss://path/to/tpch/tpch_1000x_parquet/region_parquet/'
;
CREATE EXTERNAL TABLE tpch_1000x_parquet.supplier (
    s_suppkey bigint,
    s_name string,
    s_address string,
    s_nationkey int,
    s_phone string,
    s_acctbal double,
    s_comment string
)
STORED AS PARQUET
LOCATION 'oss://path/to/tpch/tpch_1000x_parquet/supplier_parquet/'
;
```

DLA Presto中的建表语句如下：

```
create external table if not exists `tpch_1000x_orc`.`customer` (
    `c_custkey` bigint,
    `c_name` string,
    `c_address` string,
    `c_nationkey` int,
    `c_phone` string,
    `c_acctbal` double,
    `c_mktsegment` string,
```

```
    `c_comment` string
)
STORED AS `ORC`
LOCATION 'oss://path/to/tpch/tpch_1000x_orc/customer_orc/';
CREATE EXTERNAL TABLE if not exists `tpch_1000x_orc`.`lineitem` (
    `l_orderkey` bigint,
    `l_partkey` bigint,
    `l_suppkey` bigint,
    `l_linenumber` int,
    `l_quantity` double,
    `l_extendedprice` double,
    `l_discount` double,
    `l_tax` double,
    `l_returnflag` string,
    `l_linestatus` string,
    `l_shipdate` string,
    `l_commitdate` string,
    `l_receiptdate` string,
    `l_shipinstruct` string,
    `l_shipmode` string,
    `l_comment` string
)
STORED AS `ORC`
LOCATION 'oss://path/to/tpch/tpch_1000x_orc/lineitem_orc/';
create external table if not exists `tpch_1000x_orc`.`nation` (
    `n_nationkey` int,
    `n_name` string,
    `n_regionkey` int,
    `n_comment` string
)
STORED AS `ORC`
LOCATION 'oss://path/to/tpch/tpch_1000x_orc/nation_orc/';
create external table if not exists `tpch_1000x_orc`.`orders` (
    `o_orderkey` bigint,
    `o_custkey` bigint,
    `o_orderstatus` string,
    `o_totalprice` double,
    `o_orderdate` string,
    `o_orderpriority` string,
    `o_clerk` string,
    `o_shippriority` int,
    `o_comment` string
)
STORED AS `ORC`
LOCATION 'oss://path/to/tpch/tpch_1000x_orc/orders_orc/';
create external table if not exists `tpch_1000x_orc`.`part` (
    `p_partkey` bigint,
    `p_name` string,
    `p_mfgr` string,
    `p_brand` string,
    `p_type` string,
    `p_size` int,
    `p_container` string,
    `p_retailprice` double,
```

```
    `p_comment` string
)
STORED AS `ORC`
LOCATION 'oss://path/to/tpch/tpch_1000x_orc/part_orc/';
create external table if not exists `tpch_1000x_orc`.`partsupp` (
    `ps_partkey` bigint,
    `ps_suppkey` bigint,
    `ps_availqty` int,
    `ps_supplycost` double,
    `ps_comment` string
)
STORED AS `ORC`
LOCATION 'oss://path/to/tpch/tpch_1000x_orc/partsupp_orc/';
create external table if not exists `tpch_1000x_orc`.`region` (
    `r_regionkey` int,
    `r_name` string,
    `r_comment` string
)
STORED AS `ORC`
LOCATION 'oss://path/to/tpch/tpch_1000x_orc/region_orc/';
create external table if not exists `tpch_1000x_orc`.`supplier` (
    `s_suppkey` bigint,
    `s_name` string,
    `s_address` string,
    `s_nationkey` int,
    `s_phone` string,
    `s_acctbal` double,
    `s_comment` string
)
STORED AS `ORC`
LOCATION 'oss://path/to/tpch/tpch_1000x_orc/supplier_orc/';
create database if not exists`tpch_1000x_parquet`
WITH DBPROPERTIES (
    catalog = 'oss',
    location = 'oss://path/to/tpch/tpch_1000x_parquet/'
);
create external table if not exists `tpch_1000x_parquet`.`customer` (
    `c_custkey` bigint,
    `c_name` string,
    `c_address` string,
    `c_nationkey` int,
    `c_phone` string,
    `c_acctbal` double,
    `c_mktsegment` string,
    `c_comment` string
)
STORED AS `PARQUET`
LOCATION 'oss://path/to/tpch/tpch_1000x_parquet/customer_parquet/';
CREATE EXTERNAL TABLE if not exists `tpch_1000x_parquet`.`lineitem` (
    `l_orderkey` bigint,
    `l_partkey` bigint,
    `l_suppkey` bigint,
    `l_linenumber` int,
    `l_quantity` double,
```

```
    `l_extendedprice` double,
    `l_discount` double,
    `l_tax` double,
    `l_returnflag` string,
    `l_linestatus` string,
    `l_shipdate` string,
    `l_commitdate` string,
    `l_receiptdate` string,
    `l_shipinstruct` string,
    `l_shipmode` string,
    `l_comment` string
)
STORED AS `PARQUET`
LOCATION 'oss://path/to/tpch/tpch_1000x_parquet/lineitem_parquet/';
create external table if not exists `tpch_1000x_parquet`.`nation` (
    `n_nationkey` int,
    `n_name` string,
    `n_regionkey` int,
    `n_comment` string
)
STORED AS `PARQUET`
LOCATION 'oss://path/to/tpch/tpch_1000x_parquet/nation_parquet/';
create external table if not exists `tpch_1000x_parquet`.`orders` (
    `o_orderkey` bigint,
    `o_custkey` bigint,
    `o_orderstatus` string,
    `o_totalprice` double,
    `o_orderdate` string,
    `o_orderpriority` string,
    `o_clerk` string,
    `o_shippriority` int,
    `o_comment` string
)
STORED AS `PARQUET`
LOCATION 'oss://path/to/tpch/tpch_1000x_parquet/orders_parquet/';
create external table if not exists `tpch_1000x_parquet`.`part` (
    `p_partkey` bigint,
    `p_name` string,
    `p_mfgr` string,
    `p_brand` string,
    `p_type` string,
    `p_size` int,
    `p_container` string,
    `p_retailprice` double,
    `p_comment` string
)
STORED AS `PARQUET`
LOCATION 'oss://path/to/tpch/tpch_1000x_parquet/part_parquet/';
create external table if not exists `tpch_1000x_parquet`.`partsupp` (
    `ps_partkey` bigint,
    `ps_suppkey` bigint,
    `ps_availqty` int,
    `ps_supplycost` double,
    `ps_comment` string
)
```

```
)
STORED AS `PARQUET`
LOCATION 'oss://path/to/tpch/tpch_1000x_parquet/partsupp_parquet/';
create external table if not exists `tpch_1000x_parquet`.`region` (
    `r_regionkey` int,
    `r_name` string,
    `r_comment` string
)
STORED AS `PARQUET`
LOCATION 'oss://path/to/tpch/tpch_1000x_parquet/region_parquet/';
create external table if not exists `tpch_1000x_parquet`.`supplier` (
    `s_suppkey` bigint,
    `s_name` string,
    `s_address` string,
    `s_nationkey` int,
    `s_phone` string,
    `s_acctbal` double,
    `s_comment` string
)
STORED AS `PARQUET`
LOCATION 'oss://path/to/tpch/tpch_1000x_parquet/supplier_parquet/';
```

3. 分别在开源Presto集群和DLA Presto上面执行TPC-H查询，并记录每条查询的执行时间。

> **说明**
>
> - 不要同时在开源Presto集群和DLA Presto上执行查询，避免因为OSS带宽的争抢造成测试结果不准确。
> - DLA Presto需要添加以下hint指定查询在特定的集群中执行。
>
>   ```
>   /*+cluster=your-cu-name*/
>   ```

2.ETL调度

3.监控与报警

3.1. 查看Presto监控

DLA提供了Presto虚拟集群的性能监控功能，本文介绍如何通过DLA管理控制台查看集群的资源监控。

前提条件

- 您已经成功购买DLA虚拟集群。
- 如果您是RAM用户，请确认已具备AliyunARMSFullAccess权限。

操作步骤

1. 登录Data Lake Analytics管理控制台。
2. 单击左侧导航栏中的**虚拟集群管理** 。
3. 单击目标虚拟集群**详情**。

4. 在左侧导航栏单击**监控报警**，选择**标准监控**。默认显示最近一小时的监控数据，您也可以在右上角选择时间范围。

   具体监控项说明如下。

类别	监控项	说明
集群监控	Cluster CPU Usage	Presto集群的CPU使用率。
	Cluster Memory Usage	Presto集群的内存使用率。

3.2. 查看Spark监控

DLA提供了Spark虚拟集群的性能监控功能，本文介绍如何通过DLA管理控制台查看资源监控。

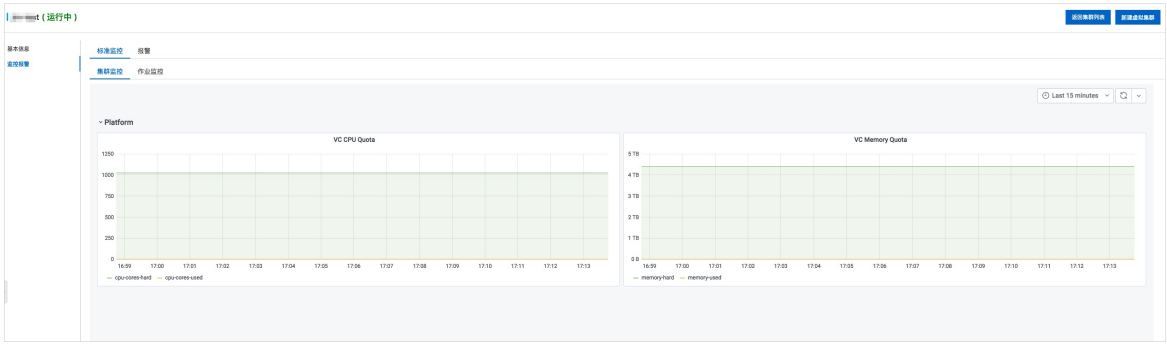
前提条件

- 您已经成功购买DLA虚拟集群。
- 如果您是RAM用户，请确认已具备AliyunARMSFullAccess权限。

查看集群监控

1. 登录Data Lake Analytics管理控制台。
2. 单击左侧导航栏中的**虚拟集群管理** 。
3. 单击目标虚拟集群**详情**。

4. 在左侧导航栏单击**监控报警**，选择**标准监控**。

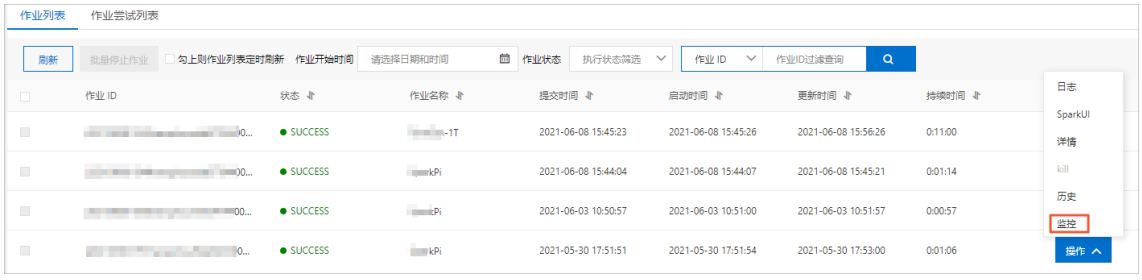


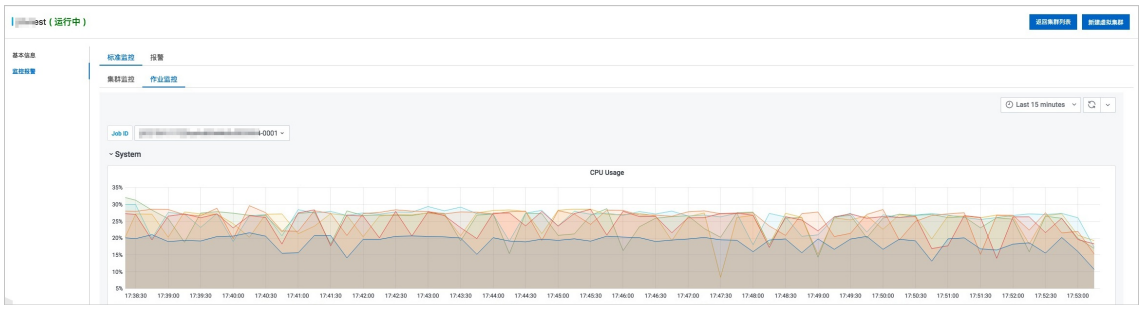
> **说明** 集群监控的监控项详情请参见监控项说明。

查看作业监控

您可以通过作业列表或者作业尝试列表查看作业监控。

- 作业列表查看作业监控。
  i. 登录Data Lake Analytics管理控制台。
  ii. 单击**Serverless Spark > 作业管理**。
  iii. 在**作业列表**中，单击目标作业**操作**，选择**监控**。





> **说明** 作业监控的监控项详情请参见监控项说明。

- 作业尝试列表查看作业监控。
  i. 登录Data Lake Analytics管理控制台。
  ii. 单击Serverless Spark > 作业管理。
  iii. 在作业尝试列表中，单击目标作业操作，选择监控。

> **说明** 作业监控的监控项详情请参见监控项说明。

监控项说明

在**标准监控**页面选择**集群监控**或者**作业监控**，具体监控项说明如下。

类别	监控项	说明
集群监控	VC CPU Quota	虚拟集群的CPU Core上限（cpu-cores-hards）和当前用量（cpu-core-used）。
	VC Memory Quota	虚拟集群的CPU内存上限（memory-hards）和当前用量（memory-used）
作业监控	CPU Usage	当前作业Driver和Executor节点的CPU使用率。
	Memory Usage	当前作业Driver和Executor节点的内存使用率。
	Network I/O	当前作业Driver和Executor节点的网络传输速度。
	Minor GC(GC time/1 min)	当前作业Driver和Executor节点每分钟 Minor GC花费时间。
	Full GC(GC time/1 min)	当前作业Driver和Executor节点每分钟Full GC花费时间。
	Streaming Processing Rate/Min	当前Streaming作业每分钟处理Records的速率。
	Streaming Processing Delay	当前Streaming作业Batch的处理延时。

类别	监控项	说明
	Streaming Scheduling Delay	当前Streaming作业Batch的调度延时。
	Structured Streaming Latency	当前Structured Streaming的作业延时。
	Structured Streaming Processing Rate	当前Structured Streaming每秒的处理速率。
	Structured Streaming Input Rate	当前Structured Streaming每秒的Input速率。

3.3. 管理报警

DLA支持配置虚拟集群以及Spark作业级别的监控告警，您可以设置报警规则，系统在监控数据满足条件时，会通知报警联系组中的所有联系人。

背景信息

监控报警是通过阿里云Prometheus监控实现的。通过阿里云Prometheus监控，您可以查看监控大盘，设置监控项，在触发监控项的报警规则时，Prometheus监控可以通过邮件、钉钉、短信、电话通知报警联系组中的所有联系人。您可以维护报警监控项对应的报警联系组，以便发生报警时，相关联系人能及时收到通知。

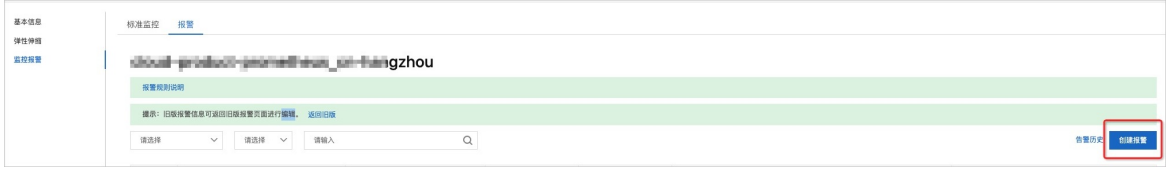
前提条件

- 您已经成功购买DLA虚拟集群。
- 如果您是RAM用户，请确认已具备AliyunARMSFullAccess权限。

添加报警

1. 登录Data Lake Analytics管理控制台。
2. 单击左侧导航栏中的**虚拟集群管理** 。
3. 单击目标虚拟集群**详情**。

4. 在左侧导航栏单击**监控报警**，选择**报警**。
5. 在右侧单击**创建报警**，进入报警配置界面。



6. 在**创建报警**面板，执行以下操作：
   i. 从**告警模板**下拉列表，选择模板。

      DLA支持的模板列表有**Presto集群CPU利用率大于90%**、**Presto集群内存利用率大于90%**、**Spark虚拟集群CPU/Memory Quota利用率大于90%**、**Spark Structure Streaming作业处理延时大于10秒**、**Spark流作业Batch处理时长大于10秒**、**Spark作业节点每分钟Full GC时间大于10秒**、**Spark作业节点内存利用率大于90%**、**Spark作业节点CPU利用率大于90%**、**Spark作业停止**。

   ii. 在**规则名称**文本框，输入规则名称，例如：Spark Structure Streaming作业处理延时大于10秒。

   iii. 在**告警表达式**文本框，输入告警表达式。以Spark Structure Streaming作业处理延时大于10秒为例，默认表达式为 `spark_structured_streaming_driver_latency / 1000 > 10` 。

      > **说明** 如果您需要对指定作业进行监控报警，请参见指定作业的报警设置。

   iv. 在**持续时间**文本框，输入时间，例如：1分钟，当告警条件连续1分钟都满足时才会发送告警。

   v. 在**告警消息**文本框，输入告警消息。

   vi. （可选）在**高级配置**的**标签**区域，单击**创建标签**可以设置报警标签，设置的标签可用作分派规则的选项。

   vii. （可选）在**高级配置**的**注释**区域，单击**创建注释**，设置键为*message*，设置值为*{{变量名}}告警信息*。设置完成后的格式为：`message:{{变量名}}告警信息` ，例如：`message:{{$labels.pod_name}}重启` 。

      您可以自定义*变量名*，也可以选择已有的标签作为*变量名*。已有的标签包括：

      - 报警规则表达式指标中携带的标签。
      - 通过报警规则创建的标签。
      - ARMS系统自带的默认标签，默认标签说明如下。

标签	说明
alertname	告警名称，格式为：告警名称_集群名称。
_aliyun_arms_alert_level	告警等级。
_aliyun_arms_alert_type	告警类型。
_aliyun_arms_alert_rule_id	告警规则对应的ID。
_aliyun_arms_region_id	地域ID。
_aliyun_arms_userid	用户ID。
_aliyun_arms_involvedObject_type	关联对象子类型，如ManagedKubernetes，ServerlessKubernetes。
_aliyun_arms_involvedObject_kind	关联对象分类，如app，cluster。
_aliyun_arms_involvedObject_id	关联对象ID。
_aliyun_arms_involvedObject_name	关联对象名称。

viii. 从**通知策略**下拉列表，选择通知策略。

如何创建通知策略，请参见通知策略。

ix. 单击**确定**。

报警配置页面显示创建的报警。

管理报警规则

1. 登录Data Lake Analytics管理控制台。
2. 单击左侧导航栏中的**虚拟集群管理** 。
3. 单击目标虚拟集群**详情**。

4. 在左侧导航栏单击**监控报警**，选择**报警**。
5. 单击**报警**页签，在右侧**操作**列按需对目标报警规则采取以下操作。
   - 如需编辑报警规则，请单击**编辑**，在**编辑报警**对话框中编辑报警规则，并单击**确认**。
   - 如需启动未启用的报警规则，请单击**开启**，然后在**状态**列中查看启动状态。
   - 如需停用已启用的报警规则，请单击**关闭**，然后在**状态**列中查看停用状态。

> **说明** 管理报警的具体操作，请参见管理报警。

3.4. 指定作业的报警设置

DLA不仅支持使用定义好的报警模板对所有作业进行监控报警，还支持对单个作业进行监控报警。本文介绍如何针对特定的作业进行监控报警。

前提条件

- 您已经成功购买DLA虚拟集群。
- 如果您是RAM用户，请确认已具备AliyunARMSFullAccess权限。
- 您已经成功创建了Spark作业。如何创建Spark作业，请参见创建和执行Spark作业。

指定作业延时触发报警

通常情况下，选择了作业延时的模板，只要有作业延时就会报警。如果您需要精确地针对特定虚拟集群的特定作业进行监控报警，可以在**创建报警**页面选择**Spark Structure Streaming作业处理延时大于10秒**模板，按下面的语法修改**告警表达式**。

```
spark_structured_streaming_driver_latency{vcName="$(vcName)",app_id=~"$(job_id).*"} / 1000
> $(latency_sec)
```

> **说明** 如何进入**创建报警**页面请参见添加报警。

告警表达式中的参数说明如下。

参数名称	参数说明
vcName	作业相关的虚拟集群名称。
job_id	作业ID。
latency_sec	作业处理延时时间，以秒为单位。

> **说明** 关于报警的更多信息，请参见管理报警。

指定作业停止触发报警

通常情况下，选择了作业停止的模板，只要有作业停止就会报警。如果您需要精确地针对特定作业进行监控报警，可以在**创建报警**页面选择**Spark作业停止**模板，按下面的语法修改**告警表达式**。

```
sum by (parent_job) (label_replace(up{pod_name=~"${job_id}.*-driver"}, "parent_job", "$1", "pod_name", "(.*?)-(.*)")) < 1
```

> **说明** 如何进入**创建报警**页面请参见添加报警。

告警表达式中的参数说明如下。

参数名称	参数说明
job_id	作业ID。

> **说明** 关于报警的更多信息，请参见管理报警。

4.SDK参考

4.1. Serverless Spark

4.1.1. 获取AccessKey

您可以为阿里云账号（主账号）和RAM用户创建一个访问密钥（AccessKey）。在调用阿里云API时您需要使用AccessKey完成身份验证。

背景信息

AccessKey包括AccessKey ID和AccessKey Secret。

- AccessKey ID：用于标识用户。
- AccessKey Secret：用于验证用户的密钥。AccessKey Secret必须保密。

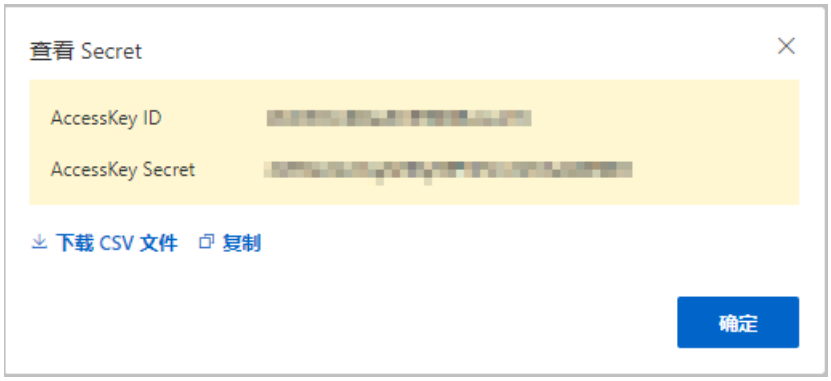
> 警告 阿里云账号AccessKey泄露会威胁您所有资源的安全。建议您使用RAM用户AccessKey进行操作，可以有效降低AccessKey泄露的风险。

操作步骤

1. 使用阿里云账号登录控制台。
2. 将鼠标置于页面右上方的账号图标，单击**AccessKey管理**。
3. 在**安全提示**对话框，选择使用阿里云账号AccessKey或RAM用户AccessKey。

   - 使用阿里云账号AccessKey
     a. 单击**继续使用AccessKey**。
     b. 在**AccessKey管理**页面，单击**创建AccessKey**。
     c. 在**手机验证**对话框，获取验证码，完成手机验证，单击**确定**。

d. 在**创建AccessKey**对话框，查看AccessKey ID和AccessKey Secret。可以单击**下载CSV文件**，下载AccessKey信息。或者单击**复制**，复制AccessKey信息。



- 使用RAM用户AccessKey
  a. 单击**开始使用子用户AccessKey**。
  b. 系统自动跳转到RAM控制台的**用户**页面，找到需要获取AccessKey的RAM用户。

> **说明** 如果没有RAM用户，请先创建RAM用户，详情请参见创建RAM用户。

  c. 单击用户登录名称。
  d. 在**认证管理**页签下的**用户AccessKey**区域，单击**创建AccessKey**。
  e. 在**手机验证**对话框，获取验证码，完成手机验证，单击**确定**。
  f. 在**创建AccessKey**页面，查看AccessKey ID和AccessKey Secret。可以单击**下载CSV文件**，下载AccessKey信息。或者单击**复制**，复制AccessKey信息。

> **说明**
> - RAM用户的AccessKey Secret只在创建时显示，不提供查询，请妥善保管。
> - 若AccessKey泄露或丢失，则需要创建新的AccessKey，最多允许为每个RAM用户创建2个AccessKey。

4.1.2. Python

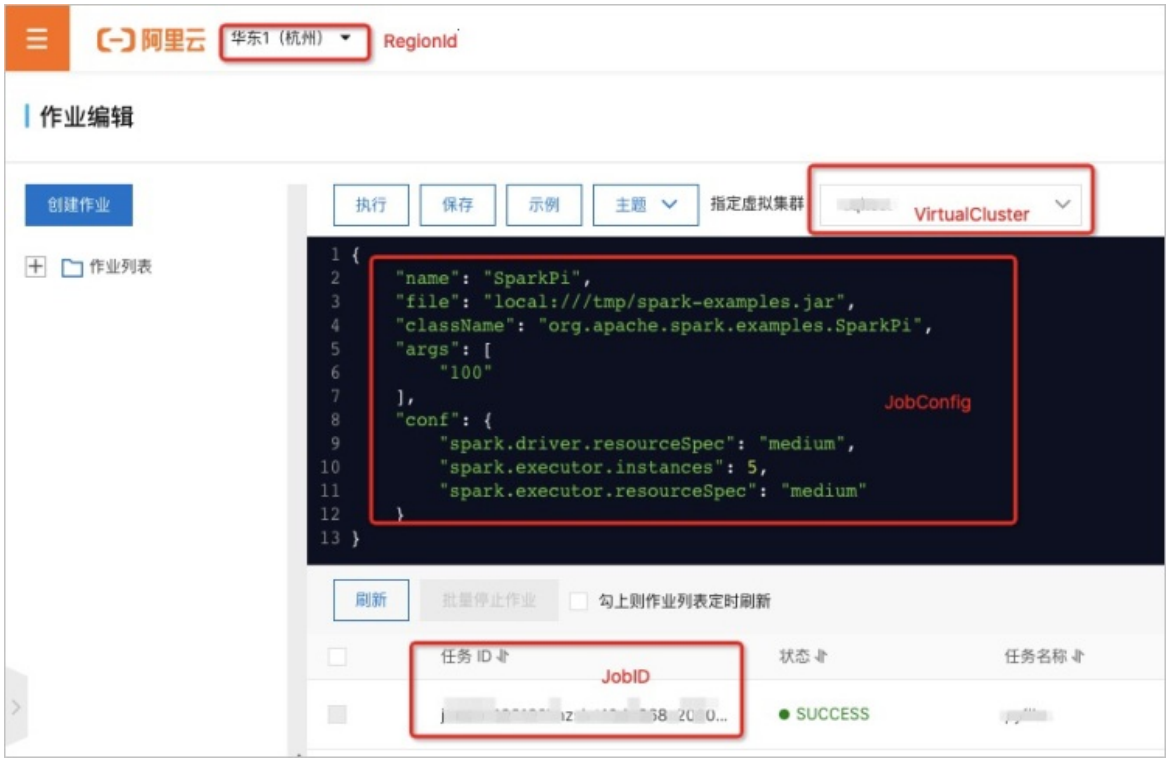
4.1.2.1. SDK安装与使用

获取开发用的SDK

可以在PIP的仓库中获取数据湖分析最新的开发SDK包，地址为Python SDK官方地址。

使用SDK提交Spark作业

1. 获取用户的AccessKey，详情请参见获取AccessKey。
2. 获取当前使用区的RegionId，阿里云各区的RegionId可以参见地域和可用区。
3. 确定执行任务的虚拟集群名称和JSON内容，您可以首先尝试在控制台提交作业，其后根据下图确定输入和返回的内容。



使用SDK提交作业的代码如下所示：

```
def submit_spark_job(region: str, access_key_id: str,
                     access_key_secret: str, cluster_name: str, job_config: str):
    """
    提交一个Spark job, 返回job_id
    :param region:              提交作业的区域ID
    :param access_key_id:       用户ak的ID
    :param access_key_secret:   用户ak的Secret
    :param cluster_name:        运行作业的Spark集群名称
    :param job_config:          定义Spark作业的JSON字符串
    :return:                    Spark任务的jobid
    :rtype:                     basestring
    :exception                  ClientException
    """
    # 创建客户端
    client = AcsClient(ak=access_key_id, secret=access_key_secret, region_id=region)
    # 初始化请求内容
    request = SubmitSparkJobRequest.SubmitSparkJobRequest()
    request.set_VcName(cluster_name)
    request.set_ConfigJson(job_config)
    # 提交Job获取结果
    response = client.do_action_with_exception(request)
    # 返回JobId
    r = json.loads(str(response, encoding='utf-8'))
    return r['JobId']
```

注意 JobConfig是一个合法的JSON格式的String, 建议您手动在控制台执行成功一些小批量的作业, 然后通过SDK来自动化提交核心业务。

4.1.2.2. Python SDK Demo

本文将演示如何使用Python SDK提交一个计算π的任务，查看任务的状态和日志，超时终止任务， 以及查看虚拟集群的历史状态。

```
"""
演示如何使用Python SDK操作数据湖分析的spark作业
author aliyun
"""
from aliyunsdkcore.client import AcsClient
from aliyunsdkopenanalytics_open.request.v20180619 import SubmitSparkJobRequest, GetJobStat
usRequest, GetJobLogRequest, \
    KillSparkJobRequest, ListSparkJobRequest, SubmitSparkSQLRequest
import json
import time
def submit_spark_sql(client: AcsClient, cluster_name, sql):
    """
    提交一个spark job, 返回job_id
    :param client:              阿里云客户端
    :param cluster_name:        运行作业的spark集群名称
    :param sql:                 sql 内容
    :return:                    spark任务的jobid
    :rtype:                     basestring
    :exception                  ClientException
```

```
    """
    # 初始化请求内容
    request = SubmitSparkSQLRequest.SubmitSparkSQLRequest()
    request.set_VcName(cluster_name)
    request.set_Sql(sql)
    # 提交job获取结果
    response = client.do_action_with_exception(request)
    # 返回job id
    r = json.loads(str(response, encoding='utf-8'))
    return r['JobId']
def submit_spark_job(client: AcsClient, cluster_name, job_config):
    """
    提交一个spark job, 返回job_id
    :param client:              阿里云客户端
    :param cluster_name:        运行作业的spark集群名称
    :param job_config:          定义spark作业的JSON字符串
    :return:                    spark任务的jobid
    :rtype:                     basestring
    :exception                  ClientException
    """
    # 初始化请求内容
    request = SubmitSparkJobRequest.SubmitSparkJobRequest()
    request.set_VcName(cluster_name)
    request.set_ConfigJson(job_config)
    # 提交job获取结果
    response = client.do_action_with_exception(request)
    # 返回job id
    r = json.loads(str(response, encoding='utf-8'))
    return r['JobId']
def get_job_status(client: AcsClient, cluster_name, job_id):
    """
    查询某个spark作业的执行状态
    :param client:              阿里云客户端
    :param cluster_name:        运行作业的spark集群名称
    :param job_id:              spark作业的id
    :return:                    spark任务的status
    :rtype:                     basestring
    :exception                  ClientException
    """
    # 初始化请求内容
    request = GetJobStatusRequest.GetJobStatusRequest()
    request.set_VcName(cluster_name)
    request.set_JobId(job_id)
    # 获取job的运行状态
    response = client.do_action_with_exception(request)
    r = json.loads(str(response, encoding='utf-8'))
    return r['Status']
def get_job_log(client: AcsClient, cluster_name, job_id):
    """
    获取spark job的日志信息
    :param client:              阿里云客户端
    :param cluster_name:        运行作业的spark集群名称
    :param job_id:              spark作业的id
    :return:                    spark作业的日志
```

```
    :rtype:                     basestring
    :exception                  ClientException
    """
    # 初始化请求内容
    request = GetJobLogRequest.GetJobLogRequest()
    request.set_VcName(cluster_name)
    request.set_JobId(job_id)
    # 获取日志信息
    response = client.do_action_with_exception(request)
    r = json.loads(str(response, encoding='utf-8'))
    return r['Data']
def kill_job(client: AcsClient, cluster_name, job_id):
    """
    杀死执行中的某个spark job
    :param client:              阿里云客户端
    :param cluster_name:        运行作业的spark集群名称
    :param job_id:              spark作业的id
    :return:                    None
    :exception                  ClientException
    """
    # 初始化请求内容
    request = KillSparkJobRequest.KillSparkJobRequest()
    request.set_VcName(cluster_name)
    request.set_JobId(job_id)
    # 杀死执行中的任务
    client.do_action_with_exception(request)
def list_job(client: AcsClient, cluster_name: str, page_number: int,
             page_size: int):
    """
    打印spark集群中历史作业详情
    :param client:              阿里云客户端
    :param cluster_name:        运行作业的spark集群名称
    :param page_number:         历史作业页码，编码是从1开始的
    :param page_size            每页返回多少作业
    :return:                    None
    :exception                  ClientException
    """
    # 初始化请求体
    request = ListSparkJobRequest.ListSparkJobRequest()
    request.set_VcName(cluster_name)
    request.set_PageNumber(page_number)
    request.set_PageSize(page_size)
    # 打印作业详情
    response = client.do_action_with_exception(request)
    r = json.loads(str(response, encoding='utf-8'))
    for job_item in r["DataResult"]["JobList"]:
        job_detail = 'JobId:{}, JobStatus:{}, ' \
                     'JobUI:{}, SumbitTime:{} \n'.format(job_item['JobId'], job_item['Statu
s'],
                                                         job_item['SparkUI'],
                                                         job_item['SubmitTime'])
        print(job_detail)
if __name__ == '__main__':
    # 必要的参数
    region = "cn-hangzhou"
```

```
region = "cn-hangzhou"
access_key_id = "xxx"
access_key_secret = "yyy"
cluster_name = "SparkCluster"
# 提交一个计算π的作业
job_config = '''
 {
     "name": "SparkPi",
     "file": "local:///tmp/spark-examples.jar",
     "className": "org.apache.spark.examples.SparkPi",
     "args": [
        "1000000"
     ],
     "conf": {
         "spark.driver.resourceSpec": "small",
         "spark.executor.instances": 1,
         "spark.executor.resourceSpec": "small"
         "spark.dla.job.log.oss.uri": "oss://test/spark-logs"
     }
}
'''
#创建一个show databases的sql语句
sql = '''
-- here is the spark conf
set spark.driver.resourceSpec=medium;
set spark.executor.instances=5;
set spark.executor.resourceSpec=medium;
set spark.app.name=sparksqltest;
set spark.sql.hive.metastore.version=dla;
-- here is your sql statement
-- add your jar
-- add jar oss://path/to/your/jar
show databases;
'''
# 创建客户端
client = AcsClient(ak=access_key_id, secret=access_key_secret, region_id=region)
#提交sql任务
#job_id: str = submit_spark_sql(client, cluster_name, sql)
# 提交任务
job_id: str = submit_spark_job(client, cluster_name, job_config)
# 轮询检查job的状态，超时则杀死作业
submit_time = time.time()
while True:
    job_status: str = get_job_status(client, cluster_name, job_id)
    # 作业已经结束
    if job_status in ["error", "success", "dead", "killed"]:
        break
    # 超时则杀死任务
    elif int(time.time() - submit_time) > 100:
        kill_job(client, cluster_name, job_id)
        break
    # 等待下一轮扫描
    else:
        time.sleep(5)
# 打印作业的详情
```

```
    print("log detail: {} \n".format(get_job_log(client, cluster_name, job_id)))
    # 查询最近的10条作业内容
    list_job(client, cluster_name, 1, 10)
```

4.1.3. Java

4.1.3.1. SDK安装与使用

本文介绍如何获取SDK以及使用SDK提交Spark作业。

获取SDK

您可以在Maven Repository中获取数据湖分析最新的SDK包，获取地址Maven SDK地址。

```
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>aliyun-java-sdk-openanalytics-open</artifactId>
    <version>1.0.13</version>
</dependency>
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>aliyun-java-sdk-core</artifactId>
    <version>4.4.6</version>
</dependency>
```

使用SDK提交Spark作业

1. 获取用户的AccessKey，详情请参见获取AccessKey。
2. 获取当前地域的RegionId，阿里云各地域的RegionId请参见地域和可用区。
3. 确定执行任务的虚拟集群名称和JSON内容，您可以首先尝试在控制台提交作业，确定输入和返回的内容。

   使用SDK提交作业的代码如下所示：

```
/**
 * 提交一个作业到数据湖分析Serverless Spark
 *
 * @param regionId           使用的数据湖分析的REGION_ID
 * @param accessKeyId        用户AccessKeyId
 * @param accessKeySecret    用户AccessKeySecret
 * @param virtualClusterName 数据湖分析虚拟集群名称
 * @param jobConfig          提交Spark作业的描述文件,需要是JSON格式
 * @return Spark JobId, 提交作业成功, 返回作业的ID, 用于后续的状态跟踪
 * @throws ClientException 提交作业可能因为网络原因等抛出错误
 */
public String submitSparkJob(String regionId,
                             String accessKeyId,
                             String accessKeySecret,
                             String virtualClusterName,
                             String jobConfig) throws ClientException {
    // 初始化阿里云平台开发Client
    DefaultProfile profile = DefaultProfile.getProfile(regionId, accessKeyId, accessKey
Secret);
    IAcsClient client = new DefaultAcsClient(profile);
    // 初始化Request, 填入集群名称和作业内容
    SubmitSparkJobRequest request = new SubmitSparkJobRequest();
    request.setVcName(virtualClusterName);
    request.setConfigJson(jobConfig);
    // 提交作业, 返回Spark作业的JobId
    SubmitSparkJobResponse response = client.getAcsResponse(request);
    return response.getJobId();
}
```

> **注意** `JobConfig` 是一个合法的JSON格式的String，建议您手动在控制台执行一些小批量的作业，然后通过SDK来自动化提交核心业务。

4.1.3.2. Java SDK Demo

本文以一个完整的程序为例，提交一个计算π的作业到数据湖分析DLA，跟踪它的状态，查询历史上的运行结果。

```
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.IAcsClient;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.openanalytics_open.model.v20180619.*;
import com.aliyuncs.profile.DefaultProfile;
import java.io.IOException;
import java.util.Arrays;
import java.util.List;
/**
 * 演示如何使用Java SDK操作数据湖分析的spark作业
 *
 * @author aliyun
 */
public class Demo {
    /**
```

```
     * 提交一个SQL作业到数据湖分析Serverless Spark
     *
     * @param virtualClusterName 数据湖分析虚拟集群名称
     * @param sql           sql内容
     * @return Spark JobId，提交作业成功，返回作业的ID，用于后续的状态跟踪
     * @throws ClientException 提交作业可能因为网络原因等抛出错误
     */
    public static String submitSparkSQL(IAcsClient client,
                                        String virtualClusterName,
                                        String sql) throws ClientException {
        // 初始化Request，填入集群名称和作业内容
        SubmitSparkSQLRequest request = new SubmitSparkSQLRequest();
        request.setVcName(virtualClusterName);
        request.setSql(sql);
        // 提交作业，返回Spark作业的JobId
        SubmitSparkSQLResponse response = client.getAcsResponse(request);
        return response.getJobId();
    }
    /**
     * 提交一个作业到数据湖分析Serverless Spark
     *
     * @param virtualClusterName 数据湖分析虚拟集群名称
     * @param jobConfig          提交Spark作业的描述文件,需要是JSON格式
     * @return Spark JobId，提交作业成功，返回作业的ID，用于后续的状态跟踪
     * @throws ClientException 提交作业可能因为网络原因等抛出错误
     */
    public static String submitSparkJob(IAcsClient client,
                                        String virtualClusterName,
                                        String jobConfig) throws ClientException {
        // 初始化Request，填入集群名称和作业内容
        SubmitSparkJobRequest request = new SubmitSparkJobRequest();
        request.setVcName(virtualClusterName);
        request.setConfigJson(jobConfig);
        // 提交作业，返回Spark作业的JobId
        SubmitSparkJobResponse response = client.getAcsResponse(request);
        return response.getJobId();
    }
    /**
     * 返回一个Spark Job当前的状态
     *
     * @param sparkJobId 用户Spark作业的ID
     * @return 返回Spark作业的状态，类型为String
     * @throws ClientException 提交作业可能因为网络原因等抛出错
     */
    public static String getSparkJobStatus(IAcsClient client,
                                           String virtualClusterName,
                                           String sparkJobId) throws ClientException {
        // 初始化Request，填入spark job id
        GetJobStatusRequest request = new GetJobStatusRequest();
        request.setJobId(sparkJobId);
        request.setVcName(virtualClusterName);
        // 提交作业，返回Spark作业的状态码
        GetJobStatusResponse response = client.getAcsResponse(request);
        return response.getStatus();
```

```
    }
    /**
     * 停止一个Spark Job
     *
     * @param sparkJobId         用户Spark作业的ID
     * @param virtualClusterName 数据湖分析虚拟集群名称
     * @return 无返回值
     * @throws ClientException 提交作业可能因为网络原因等抛出错
     */
    public static void killSparkJob(IAcsClient client,
                                    String virtualClusterName,
                                    String sparkJobId) throws ClientException {
        // 初始化Request，填入spark job id
        KillSparkJobRequest request = new KillSparkJobRequest();
        request.setVcName(virtualClusterName);
        request.setJobId(sparkJobId);
        // 提交作业，返回Spark作业的状态码
        KillSparkJobResponse response = client.getAcsResponse(request);
    }
    /**
     * 返回一个Spark Job的日志
     *
     * @param client     客户端
     * @param virtualClusterName 数据湖分析虚拟集群名称
     * @param sparkJobId         用户Spark作业的ID
     * @return 返回Spark作业的状态，类型为String
     * @throws ClientException 提交作业可能因为网络原因等抛出错
     */
    public static String getSparkJobLog(IAcsClient client,
                                String virtualClusterName,
                                String sparkJobId) throws ClientException {
        // 初始化Request，填入spark job id
        GetJobLogRequest request = new GetJobLogRequest();
        request.setJobId(sparkJobId);
        request.setVcName(virtualClusterName);
        // 提交作业，返回Spark作业的日志
        GetJobLogResponse response = client.getAcsResponse(request);
        return response.getData();
    }
    /**
     * 查询某个虚拟集群上提交的Spark作业，通过翻页可以遍历所有的历史作业信息
     *
     * @param client     客户端
     * @param pageNumber 查询的页码，从1开始
     * @param pageSize   每页返回数量
     * @throws ClientException 提交作业可能因为网络原因等抛出错
     */
    public static void listSparkJob(IAcsClient client,
                                    String virtualClusterName,
                                    int pageNumber,
                                    int pageSize) throws ClientException {
        // 初始化Request，填入spark job id
        ListSparkJobRequest request = new ListSparkJobRequest();
        request.setVcName(virtualClusterName);
        request.setPageNumber(pageNumber); // pageNumber 从1开始
```

```
        request.setPageNumber(pageNumber); // pageNumber 从1开始
        request.setPageSize(pageSize);
        // 提交作业，返回Spark作业的状态码
        ListSparkJobResponse response = client.getAcsResponse(request);
        // 获取任务列表
        List<ListSparkJobResponse.DataResult.Data> sparkJobList = response.getDataResult().
getJobList();
        for (ListSparkJobResponse.DataResult.Data job : sparkJobList) {
            System.out.println(String.format("JobName: %s, JobUI: %s, JobStatus: %s, JobCon
fig: %s",
                                             job.getJobName(),
                                             job.getStatus(),
                                             job.getSparkUI(),
                                             job.getDetail()));
        }
    }
    public static void main(String[] args) throws IOException, ClientException, Interrupted
Exception {
        // 提交任务必须的参数
        String region = "cn-hangzhou";
        String accessKeyId = "xxx";
        String accessKeySecret = "yyy";
        String virtualClusterName = "MyCluster";
        // 需要是一个合法的JSON格式的字符串
        String jobConfig=
                "{\n" +
                "    \"name\": \"SparkPi\",\n" +
                "    \"file\": \"local:///tmp/spark-examples.jar\",\n" +
                "    \"className\": \"org.apache.spark.examples.SparkPi\",\n" +
                "    \"args\": [\n" +
                "        \"100\"\n" +
                "    ],\n" +
                "    \"conf\": {\n" +
                "        \"spark.driver.resourceSpec\": \"medium\",\n" +
                "        \"spark.executor.instances\": 5,\n" +
                "        \"spark.executor.resourceSpec\": \"medium\"\n" +
                "    }\n" +
                "}";
        String sql = "-- here is the spark conf\n"
                     + "set spark.driver.resourceSpec=medium;\n"
                     + "set spark.executor.instances=5;\n"
                     + "set spark.executor.resourceSpec=medium;\n"
                     + "set spark.app.name=sparksqltest;\n"
                     + "set spark.sql.hive.metastore.version=dla;\n"
                     + "-- here is your sql statement\n"
                     + "-- add your jar\n"
                     + "-- add jar oss://path/to/your/jar\n"
                     + "show databases;";
        // 初始化阿里云平台开发Client
        DefaultProfile profile = DefaultProfile.getProfile(region, accessKeyId, accessKeySe
cret);
        IAcsClient client = new DefaultAcsClient(profile);
        // 提交任务
        // 您也可以选择提交SQL作业
        //String sparkJobId = submitSparkSQL(client, virtualClusterName, sql);
```

```
        //String sparkJobId = submitSparkSql(client, virtualClusterName, sql);
        String sparkJobId = submitSparkJob(client, virtualClusterName, jobConfig);
        // 轮询任务状态，超时未完成则杀死任务
        long startTime = System.currentTimeMillis();
        List<String> finalStatusList = Arrays.asList("error", "success", "dead", "killed");
        while (true) {
            String status = getSparkJobStatus(client, virtualClusterName, sparkJobId);
            if (finalStatusList.contains(status)) {
                System.out.println("Job went to final status");
                break;
            } else if ((System.currentTimeMillis() - startTime) > 100000) {
                // 如果超时则杀死job
                System.out.println("Kill expire time job");
                killSparkJob(client, virtualClusterName, sparkJobId);
                break;
            }
            // 打印状态，等待5秒，进入下一轮查询
            System.out.println(String.format("Job %s status is %s", sparkJobId, status));
            Thread.sleep(5000);
        }
        // 打印作业的日志
        String logDetail = getSparkJobLog(client, virtualClusterName, sparkJobId);
        System.out.println(logDetail);
        // 打印最近10条作业的明细
        listSparkJob(client, virtualClusterName, 1, 10);
    }
}
```

5.常见问题

5.1. 数据湖管理FAQ

本文汇总了数据湖管理相关的常见问题及解决方案。

- **Lakehouse相关问题**
  - 什么是Lakehouse?
  - Lakehouse数据入湖时，对线上RDS有压力吗？如何控制建仓的限流能力？
  - Lakehouse工作负载为什么运行失败，又没有Spark Log日志可以看？
- **元数据发现相关问题**
  - 为什么配置了元信息发现，并在“手动执行”之后，过了几天新的数据看不到了？
  - OSS数据源配置数仓模式和自由模式的差异以及适用场景是什么？
  - 为什么Excel导出的CSV文件没有被识别建表？
  - 为什么一个目录下面是同样Schema的JSON文件，但是没有建表？
  - 使用SLS元数据发现的批量模式，为什么有的Logstore生成了表，有的没有？
  - 为什么SLS元数据发现报错：Make sure the bucket you specified be in the same region with DLA?
  - OSS存储CSV格式文件时，为什么OSS元数据发现不出任何表？
  - OSS存储CSV格式文件时，为什么OSS元数据发现的字段类型不对？
  - OSS元数据发现，是否支持CSV文件第一行是说明，第二行是文件头，第三行才是数据？
- **多库合并/一键建仓相关问题**
  - 如何处理一键建仓连接数据库报错：Access denied for user?
  - 如何处理一键建仓连接数据库报错：because Could not create connection to database server. Attempted reconnect 3 times. Giving up?
  - 如何处理一键建仓连接数据库报错：because Encountered too many errors talking to a worker node. The node may have crashed or be under too much load. This is probably a transient issue, so please retry your query in a few minutes?
  - 为什么多库合并、一键建仓报错：No such instance with ID: rm-xxxxxx?
  - 如何处理一键建仓任务报错：because Communications link failure?
  - 哪些区支持数据湖构建服务？
  - 为什么一键建仓任务成功了，有的表没有同步过来？
  - 如何处理一键建仓任务失败，报错：mppErrorCode=30101, because Direct buffer memory...?
  - 如何处理一键建仓任务失败，报错关键字：because Failed to rename某个OSS目录?
  - 如何处理一键建仓MongoDB失败，报错：Cannot allocate slice larger than XXXXX bytes?
  - 为什么一键建仓运行了十几个小时然后失败了？

Lakehouse相关问题

什么是Lakehouse?

“Lakehouse”是基于数据湖的数仓，一种新的大数据范式，最根本出发点就是为了解决单纯Data Lake应用下的各种问题，例如不支持UPSERT，不支持多版本，不支持增量ETL，小文件太多，格式不是分析型的，元信息不统一，Schema没有约束，缺乏合理的数据索引等等。

围绕OSS对象存储等数据湖存储，构建上层可扩展的数据入湖能力，把Hudi、Delta等高效的对象管理格式和Parquet、ORC等对象格式，写入到数据湖中，并在写入过程中支持UPSERT、小文件合并、MVCC多版本、快照读等能力，用数仓的特性来解决单纯Data Lake下所不能提供的能力；在写入过程中维护海量的库表元信息，给上层的SQL和分析引擎提供统一的数据访问视图。

Lakehouse数据入湖时，对线上RDS有压力吗？如何控制建仓的限流能力？

如果是实时入湖的话，数据来自DTS，使用的是Binlog，对线上RDS无压力；
如果是建仓入湖的话会有一些压力，通过参数来控制。
详情请参见高级选项功能。

Lakehouse工作负载为什么运行失败，没有Spark Log日志可以看？

点击查看workload“详情”弹框，如果job status显示：“specified driver resource spec: [xlarge]+executor resource spec:[xlarge] * executor instances:[2] exceed the max CPU:[16.0] or max Memory:[64.0], errorCode=InvalidParameter” 类似信息时，表示您在创建workload时，设置的Spark CU使用数超过了对应Virtual Cluster的Max使用限制。

解决方案有：

- 调整Virtual Cluster的Max CU限制。
- 调整workload使用的Spark CU个数，确保任务使用的CU数小于Virtual Cluster的Max CU限制。

元数据发现相关问题

为什么配置了元数据发现，并在“手动执行”之后，发现任务执行时间点之后的新的数据看不到了？

原因：“手动执行”在手动触发执行完一次时，只能发现执行当时已存在的数据，不会发现手动执行后又新增的数据。

解决方案：您需要修改成“定时执行”模式，新增的数据才能被感知增加进来。

OSS数据源配置数仓模式和自由模式的差异以及适用场景是什么？

OSS数据源配置	使用场景	OSS路径格式要求	识别精度	性能
数仓模式	用户直接上传数据到OSS，并期望构建可分析与计算的标准数据仓库。	“库/表/文件”或者“库/表/分区/.../分区/文件”	高	高
自由模式	已存在OSS数据，但OSS的路径不清晰。期望通过元信息发现，构建可分析的库表分区。	无要求	一般	一般

为什么Excel导出的CSV文件没有被识别建表？

目前元数据发现支持的是CSV文本格式，因此需要确认Excel文件导出的是CSV文本文件。

> **说明**
>
> 识别一个CSV文件的Schema是通过采样文件，然后读取文件前1000行，需要确认前1000行的字段及分隔是否完全一致。

为什么一个目录下面是同样Schema的JSON文件，但是没有建表？

目前元数据发现只支持对只包含文件的目录进行识别，如果一个目录下既有文件又有目录，会被过滤掉。

使用SLS元数据发现的批量模式，为什么有的Logstore生成了表，有的没有？

您需要确认没有生成表的Logstore是否配置了投递到OSS。

为什么SLS元数据发现报错：Make sure the bucket you specified be in the same region with DLA?

原因：SLS是运行投递到跨region的OSS bucket上，而目前DLA 发现是不能跨region访问的。所以在SLS发现任务运行时，可能会报如下错误“OSS object [oss://test-bucket/] is invalid. Make sure the bucket you specified be in the same region with DLA. Detailed message: The bucket you are attempting to access must be addressed using the specified endpoint. Please send all future requests to this endpoint.”

解决方案：在SLS投递数据所在的OSS region上，使用“OSS元数据发现”来替代“SLS元数据发现”任务，进而实现自动建表时查询SLS所投递的数据。

OSS存储CSV格式文件时，为什么OSS元数据发现不出任何表？

这种情况一般原因有：

- 某个字段值超过了4096字节长度，导致采样中断。
- 相同目录下多个文件采样推断出不同的Schema，导致无法确定以这个目录下的哪个Schema为准 。
- 文件格式不是CSV格式。

OSS存储CSV格式文件时，为什么OSS元数据发现的字段类型不对？

这种情况一般原因有：

- 这个字段存在空值，采样推断会向上设置为string类型 。
- CSV文件数据某个字段有不同的类型值。
- CSV的分隔符格式不对，导致识别的都是string。

OSS元数据发现，是否支持CSV文件第一行是说明，第二行是文件头，第三行才是数据？

不支持这种类型。

目前OSS元数据发现支持的CSV格式为：

1. 无文件头，只有数据行的情况。
2. 有第一行作为文件头的情况。

其他非标准CSV情况都不支持。

多库合并/一键建仓相关问题

如何处理一键建仓连接数据库报错：Access denied for user?

原因：一般是由于修改了用户名或密码。

解决方案：请确认是否修改了连接数据库的用户名和密码。如果修改了用户名或者密码，可以点击**Schema管理**，找到您的一键建仓任务，点击**详细信息>配置>更新**，修改一键建仓任务的配置。

如何处理一键建仓连接数据库报错：because Could not create connection to database server. Attempted reconnect 3 times. Giving up?

原因：RDS源库压力过大。

解决方案：您可以采取如下措施。

- 调整一键建仓同步时间，与高负载的读写业务错峰运行。
- 调整一键建仓的并发高级配置，降低并发数，如：connections-per-job=10、total-allowed-connections=30，表明一个表将由10个并发连接去读，并且最大有30个连接，即最多3个表并发进行同步。您可以针对您的RDS实例规格进行适当调整，详情请参见文档高级选项功能。

如何处理一键建仓连接数据库报错：because Encountered too many errors talking to a worker node. The node may have crashed or be under too much load. This is probably a transient issue, so please retry your query in a few minutes?

原因：此时DLA公共版整体负载过高。

解决方案：您可以再次点击“执行”来重试建仓任务。若还未成功，请联系 “DLA答疑”。

为什么多库合并、一键建仓报错：No such instance with ID: rm-xxxxxx?

该报错信息表示当前建仓任务涉及的数据源端RDS实例rm-xxxxxx已经释放了。

如何处理一键建仓任务报错：because Communications link failure?

详细如报错The last packet successfully received from the server was 900,120 milliseconds ago. The last packet sent successfully to the server was 900,120 milliseconds ago.), 23 out of 110 jobs are finished。

这种情况一般有如下两种原因。

- 用户数据库有视图。
- RDS源库压力过大，RDS/MySQL 重启中断或者无响应。

解决方案：排除视图，降低并发数。

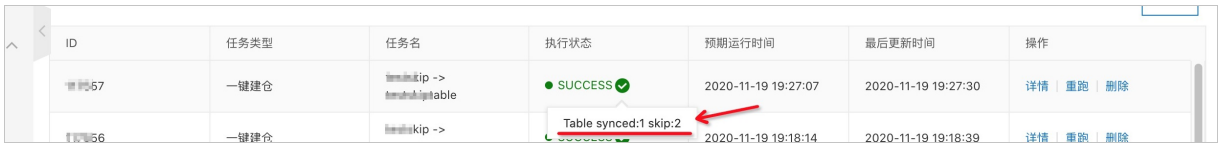
哪些区支持数据湖构建服务?

中国内地的地域都支持，中国内地以外的地域支持列表如下。

地域	数据湖构建服务
中国，香港	正常
日本，东京	正常
新加坡	正常

地域	数据湖构建服务
美国，硅谷	正常
美国，弗吉尼亚	正常
英国，伦敦	正常
德国，法兰克福	正常
澳大利亚，悉尼	正常
马来西亚，吉隆坡	正常
印度尼西亚，雅加达	正常
印度，孟买	正常

为什么一键建仓任务成功了，有的表没有同步过来？




```
        },
        "stage": "SUCCESS",
        "extraWarnTips": "The following table names are invalid!(count: 1),Skip: 用户
2.The following tables have invalid fieldNames!(count: 1),Skip: users3",
        "CopyTablesFromRds-TEMP": {
            "all": 1,
            "running": 0,
```

您可以在任务运行状态处查看跳过了多少张表，且在运行详情的extraWarnTips信息中查看跳过了哪些表。例如一键建仓会跳过表名、字段名为中文的表。

如果您发现表名和字段名都不是中文可以检查所填写的**JDBC账户是否具有该表的访问权限。**

如何处理一键建仓任务失败，报错：mppErrorCode=30101, because Direct buffer memory...?

原因：Presto公共集群负载过高。

解决方案：您可以手动再次执行重试建仓任务，如果还失败，请联系 “DLA答疑” 。

如何处理一键建仓任务失败，报错关键字：because Failed to rename <OSS Path>?

原因：建仓时，所设置的OSS路径位置是一个OSS归档型的Bucket，导致DLA无法正常重命名这个目录。

解决方案：您可以重建一个建仓任务，设置一个非归档类型的OSS bucket目录位置。

如何处理一键建仓MongoDB失败，报错：Cannot allocate slice larger than XXXXX bytes?

详细报错如下。

because Error reading from mongodb: Cannot allocate slice larger than 2147483639 bytes。

原因：Presto拉取MongoDB数据时，有的字段数值特别大。

解决方案：您可以修改建仓高级配置，通过**sensitive-columns配置**过滤掉大字段的同步。详情请参见文档高级选项功能。

为什么一键建仓运行了十几个小时然后失败了?

这种情况一般有如下两种原因。

- 您的库表中表的数量过多，有几千个表，可能会导致同步超时。
- 同步超时的表没有索引字段或者整型UNIQUE KEY字段，导致Presto计算Split任务运行超时。

解决方案：您可以调整建仓高级配置，设置并发数为**connections-per-job=1**。详情请参见文档高级选项功能。

5.2. Presto FAQ

本文汇总了使用DLA Presto的常见问题及解决方案。

- **高频问题**
  - 在哪些情况下，共享集群（Public）不计费?
  - 共享集群（Public）的算力是多大?
  - 在控制台执行SELECT语句为什么会有Limit 500的限制？能够突破这个限制吗?
  - 阿里云子账号可以在哪里执行SQL?
  - 什么是异步执行？如何使用?
  - 控制台上异步执行如何拿到执行结果?
  - 多个Hint如何一起使用?
- **虚拟集群管理相关问题**
  - 如何设置DLA Presto默认集群?
  - 如何设置链接级别的默认虚拟集群?
  - 如何把DLA Presto提交到指定的虚拟集群?
- **元数据/DLL相关问题**
  - 为什么执行 `MSCK REPAIR TABLE` 加载不了分区?

- 如何处理建库报错：Generate vpc reverse access mapping ip & mapping port failed, please check your vpc_id(vpc-xxxx) and instance_id(rm-xxx)!?
- 如何查看表的分区数目?

- **OSS/Hive相关问题**
  - 如何通过SQL查询一个OSS表里面的文件数?
  - 如何查询CSV文件中文出现乱码问题?
  - 使用Insert类型的SQL语法时，如何减小输出的文件数?
  - 支持基于.gz压缩的数据吗?
  - 能同时处理相同目录下的压缩文件和非压缩文件吗?
  - DLA的表默认会递归读取表目录下的所有子目录和文件吗?
  - 为什么同一个SQL从HDFS同步数据到OSS，在DLA中查询是11万数据量，自建的集群查询是19万数据量?
- **性能相关问题**
  - 如何解决查询报错：“Query exceeded distributed user memory limit of 2.00TB or Query exceeded per-node user memory limit of xxGB”?
  - 文件格式是ORC，为什么看扫描量是扫描了整个文件，而不是只扫描SQL里面指定的列?
  - 如何开启大查询功能?
  - 如何调整查询RDS类数据源的并发度?
- **其他问题**
  - 如何处理报错：because '3.00199E9' in column '4' is outside valid range for the datatype INTEGER?
  - 如何处理报错：Application was streaming results when the connection failed. Consider raising value of ‘net_write_timeout’ on the server.?
  - 如何处理向OTS写数据时的报错：Code: OTSParameterInvalid, Message: Invalid update row request: missing cells in request?

高频问题

在哪些情况下，共享集群（Public）不计费?

DDL是不收费的，其它情况按照扫描量收费，详情请参见按扫描量付费。

共享集群（Public）的算力是多大?

共享集群（Public）是一个所有用户共享使用的集群，每个用户的算力大致等价于一个10Core的集群。

在控制台执行SELECT语句为什么会有Limit 500的限制？能够突破这个限制吗?

原因：太多的数据展示导致页面崩溃，为了保障控制台页面的稳定性，设置了行数500的限制。

解决方案：您可以通过MySQL客户端连接DLA来突破这个限制。

RAM用户可以在哪里执行SQL?

当前DLA控制台都是使用主账号执行SQL的。RAM用户只能通过在DLA控制台生成DLA的子账号（用户名+密码），使用DMS执行SQL。

什么是异步执行？如何使用？

异步执行主要针对ETL类SQL(INSERT...SELECT....)， 这种SQL耗时较长，如果使用同步执行，那么这段时间客户端与服务器端之间的连接会始终被占用，如果有网络异常还会导致查询失败。而异步执行则是提交SQL之后立即返回一个ID，后续只需使用 `SHOW QUERY_TASK` 语句去监控这个查询的状态即可。

异步查询的提交：

```
/*+ run-async=true*/ SELECT * FROM tbl;
```

异步查询的状态查询：

```
SHOW QUERY_TASK WHERE ID = '.....'
```

控制台上异步执行如何拿到执行结果？

先执行异步执行，控制台会显示ASYNC_TASK_ID，用这个ID在同步执行里面运行 `show query_task where id = 'xxxx'` ，返回结果里面的result_file_oss_file字段为结果所在OSS的路径。

> 注意
>
> 只有当SQL执行完成后，result_file_oss_file才会有值。

多个Hint如何一起使用？

您可以参考如下代码。

```
/*+ cluster=public,run-async=true */
SELECT * FROM tbl
```

虚拟集群问题

如何设置DLA Presto默认集群？

如果需要所有的查询默认都提交到某个指定的集群，那么您可以通过以下菜单路径进行设置：**系统设置 > 设置Presto引擎的默认集群**。

如何设置链接级别的默认虚拟集群？

您可以通过在用户名后面加上 `@<集群标识>` 的方法来指定链接的集群。

如何把DLA Presto提交到指定的虚拟集群？

当您创建虚拟集群后（假设命名为dladw），您可以通过如下方式把SQL提交到指定的虚拟集群。

- 通过在SQL中指定hint的方式把SQL提交到指定的虚拟集群执行，例如：/*+cluster=dladw*/SELECT * FROM tbl;
- 通过链接用户名指定虚拟集群：支持在用户名后面加上 `@<集群标识>` 的方法来指定链接的集群。

> **说明**
>
> 您可以在集群详情页面查看集群标识。

- 设置默认集群为dladw。

元数据/DDL相关问题

为什么执行MSCK REPAIR TABLE加载不了分区？

MSCK的命令只能对目录名字符合XXX=XXX的格式加载分区。例如：oss://abc/yyy/action/year=2018/。

如何处理建库报错：Generate vpc reverse access mapping ip & mapping port failed, please check your vpc_id(vpc-xxxx) and instance_id(rm-xxx)!?

此时报错的可能原因和解决方案如下。

- 原因：您的VPC ID和Instance ID填写错误。
  解决方案：请正确进行填写。
- 原因：您的RDS实例做过数据迁移，页面显示的实例ID不是真实的ID。

  解决方案：您可以添加一个属性USE_VPC_CLOUD_INSTANCE_ID = 'true'。示例代码如下。

```
CREATE SCHEMA `test_db` WITH DBPROPERTIES (
CATALOG = 'mysql',
LOCATION = 'jdbc:mysql://rm-xxx.mysql.rds.aliyuncs.com:3306/dla_test',
USER='dla_test',   PASSWORD='xxxxx',   INSTANCE_ID = 'rm-xxx',
VPC_ID = 'vpc-xxxx',
USE_VPC_CLOUD_INSTANCE_ID = 'true' );
```

如何查看表的分区数目？

您可以通过如下命令查看表的分区数目。

```
select count(distinct <partition_key>) from tbl;
```

OSS/Hive数据源相关问题

如何通过SQL查询一个OSS表里面的文件数？

您可以通过如下命令进行查询。

```
select count(distinct `$path`) from tbl;
```

> **注意**
>
> 上面的SQL语句中您只需要修改tbl为您真实的表名，其它部分不用修改。

如何查询CSV文件的中文出现乱码问题？

原因：在Linux中使用file命令查看文件编码，之后建表的时候使用LazySimpleSerDe作为Serde。

解决方案：如果原始编码为ISO-8859，您可以进行如下配置serialization.encoding=gbk。

使用Insert类型的SQL语法时，如何减小输出的文件数？

您可以通过Hint调整以下两个参数的取值。

- table_write_worker_count：输出Task的并行度。
- task_writer_count：每个Task写文件的并行度。

输出文件数一般等于table_write_worker_count*task_writer_count。如果需要减小输出文件数，您可以把table_write_worker_count取值调整为10，task_writer_count取值调整为2。

> 说明
>
> - 由于集群规模、实际数据分布等因素都会影响输出的文件数，因此控制输出的文件数比较复杂。
> - 只有CU版本才支持输出文件数的相关配置。

支持基于.gz压缩的数据吗？

支持。目前支持GZIP和SNAPPY两种压缩算法。

能同时处理相同目录下的压缩文件和非压缩文件吗？

可以。

DLA的表默认会递归读取表目录下的所有子目录和文件吗？

是的。

为什么同一个SQL从HDFS同步数据到OSS，在DLA中查询是11万数据量，自建的集群查询是19万数据量？

原因：大部分情况是同步数据到OSS存在问题。
解决方案：您可以查看自己的OSS的数据是否正常同步。

性能相关问题

如何解决查询报错：“Query exceeded distributed user memory limit of 2.00TB or Query exceeded per-node user memory limit of xxGB”？

DLA Presto中一个查询能使用的单个节点的内存以及整个集群的总内存是有限制的，您可以通过如下方式进行解决。

- 优化JOIN以减少对于内存的需求。例如把数据量大的放在左边，数据量小的放在右边。
- 对数据进行分区。
- 分拆计算，把一个大SQL拆成多个小SQL。

文件格式是ORC，为什么看扫描量是扫描了整个文件，而不是只扫描SQL里面指定的列？

分析型的查询往往只会获取一个表里面少数几列的数据，这样执行引擎比如Presto在实际扫描底层数据的时候只需要扫描需要的列的数据。而这种节省扫描量的效果只有当底层的数据是以列存的形式存储才能达到。示例代码如下。

```
SELECT col1 FROM tbl1 WHERE col2 = 'hello;
```

但是在实际使用过程中，如果文件不是很大，或者文件不小但是表的列很多的情况，节省扫描量的效果会不起作用，因为ORC_TINY_STRIPE_THRESHOLD参数会控制Presto只扫描单个列还是整个文件。示例代码如下。

```
dataSizeSessionProperty(
    ORC_TINY_STRIPE_THRESHOLD,
    "ORC: Threshold below which an ORC stripe or file will read in its entirety",

    hiveClientConfig.getOrcTinyStripeThreshold(),
    false);
```

ORC_TINY_STRIPE_THRESHOLD参数的默认值是8MB，所以如果您的Stripe太小，Presto会读取整个文件，而不是读取一个个Stripe。如果每个Stripe太小，一次次地读取Stripe花费在网络上的开销可能比直接读取整个文件的开销还要大。

如何开启大查询功能？

当您查询的数据需要消耗比较多的内存导致一直失败时，您可以开启虚拟集群的大查询功能。开启方法，在查询中添加HINT，示例如下。

```
/*+big_query=true*/insertintotable1SELECT*FROMtable2;
```

> **说明**
>
> - 大查询功能只支持CU版本，不支持扫描量版本。
> - 大查询功能并不能让您查询任意大小的数据量和任意复杂的SQL，如果big_query还解决了不了您的问题请联系DLA答疑同学。
> - Schema的Catalog类型必须是Hive，否则会出现如下报错：big_query only support hive catalog ...。

如何调整查询RDS类数据源的并发度？

以RDS为例，使用DLA扫描线上数据时，如果RDS实例规格比较小，可能无法支撑默认的JDBC Connector并发度。您可以在DLA中引入Hint **/*+jdbc-scan-splits=2*/**，指定扫描一个数据表时的JDBC Connector并发度。

```
mysql> /*+jdbc-scan-splits=2*/select count(*) from customer;
+----------+
| count(*) |
+----------+
| 15000002 |
```

上述customer表对应RDS MySQL数据库中的customer表。DLA扫描RDS MySQL的customer表时，会使用两个线程同时扫描数据。

注意

- 待扫描目标表必须有一个自增主键，否则无法使用Hint调整查询并发度。对于有自增主键的表，您可以按照自增主键把一个查询切分成多个子查询提高查询并发度。示例代码如下。

```
-- 原始SQL
select * from customer;
-- 切分成如下的子查询
select * from customer where id >=0 and id <=10000;
select * from customer where id >=10000 and id <=20000;
```

- DLA最大并发度是500，默认并发度是1，对于一些规格配置较低或者复杂数据库，可以将默认并发度适当调低。

其它问题

如何处理报错：because '3.00199E9' in colum n '4' is outside valid range for the datatype INTEGER?

原因：底层对应的数据超出了DLA中INT类型能表示的最大范围。

解决方案：您可以使用BIGINT类型来映射。

如何处理报错：Application was streaming results when the connection failed. Consider raising value of 'net_write_timeout' on the server.?

原因：读取MySQL数据源的数据时，由于默认的net_write_timeout取值设置较小，数据还没有读取完毕，MySQL数据源就关闭了连接。

解决方案：您可以在MySQL数据源把net_write_timeout参数的取值适当调高。

如何处理向OTS写数据时报错：Code: OTSParameterInvalid, Message: Invalid update row request: missing cells in request?

原因：OTS属性列为空。

解决方案：您需要手动过滤掉属性列为空的记录。

5.3. Spark FAQ

本文汇总了使用DLA Spark的常见问题及解决方案。

常见问题

- 如何处理Spark作业报错：The VirtualCluster's name is invalid or the VirtualCluster's is not in running state?
- 如何处理Spark作业报错：User %s do not have right permission [***] to resource [***]?
- 如何处理Spark作业报错：No space left on device?
- Spark Executor出现Dead怎么处理？
- Spark访问数据源网络不通怎么办？

常见问题

如何处理Spark作业报错：The VirtualCluster's name is invalid or the VirtualCluster's is not in running state?

常见原因：虚拟集群名称（VcName）没配置，一般是由于该虚拟集群不存在或者该虚拟集群不是在运行中状态

解决方案：您需要填写正确的虚拟集群，如下所示。

- 如果该错误是DLA控制台作业管理界面报出，请先创建虚拟集群，再提交作业。
- 如果您使用的不是控制台方式，而是Aliyun OpenAPI、Spark-submit脚本或者Jupyter等，需要检查配置参数VcName对应的需求集群是否未填写或者填写错误。查找可用的虚拟集群名称的方法如下。
  1. 登录DLA控制台。
  2. 左侧导航栏单击**虚拟集群管理**。
  3. 在虚拟集群列表中选择状态是运行中的虚拟集群的名字。

如何处理Spark作业报错：User %s do not have right permission [***] to resource [***]?

常见原因：当前子账号没有调用该接口的权限。

解决方案如下。

- 配置相应权限，详情请参见快速配置子账号权限。
- 登录RAM控制台为该账号添加报错信息中对应的Resource权限。

如何处理Spark作业报错：No space left on device?

首先判断报错信息是否是Executor本地内存不足导致的，判断方法是查看SparkUI中Executor的Stderr日志信息。如果是，您可以通过添加如下配置来增加Executor本地盘的大小。

```
spark.k8s.shuffleVolume.enable: true
spark.dla.local.disk.size:  50Gi
```

> 注意
>
> - 本地盘为ESSD类型，暂时免费，后续会收费。
> - 当前本地盘最大为100Gi。
> - 如果已经配置了100Gi的本地盘，仍然提示空间不足，那么可以考虑增加Executor的数量。

Spark Executor出现Dead怎么处理？

首先进入SparkUI页面，如果Failed Stages的Failure reason中包含Failed to connect to /xx.xx.xx.xx:xxxx或者在SparkUI->Executors页面出现状态为Dead的Executor，往往是因为Spark Executor由于某种原因退出了。

> 说明
>
> 如何进入SparkUI页面，详情请参见Spark UI。

常见场景如下。

1. Driver日志出现如下报错：ERROR TaskSchedulerImpl: Lost executor xx on xx.xx.xx.xx:The executor with id xx exited with exit code 137.

原因：Executor进程内存用超了。容器的整体内存使用超过了容器最大允许使用内存上限，导致Spark进程被kill-9强制杀掉了。Spark Executor除了JVM本身使用内存外，往往还包括堆外内存（Shuffle、Cache），以及Python UDF等使用的内存。

解决方案：调大参数spark.executor.memoryOverhead（单位MB）。该参数表示容器内部非Spark Executor进程可使用的内存容量，默认是Executor容器总内存容量的30%。比如您当前配置的Executor规格是Medium（2C8G），那么默认的MemoryOverhead是2.4G，您可以调大该配置如下：

```
spark.executor.memoryOverhead = 4000。
```

2.日志中出现java.lang.OutOfMemoryError的日志。

原因：在SparkUI->Executors页面查看状态为Dead的Executor的stderr/stdout日志链接查找具体报错原因。

解决方案如下。

- 优化应用，避免大内存占用。
- 调大Executor资源规格，比如Small提高到Medium。

3.除上述报错以外，其他报错的解决方案。

查看Dead状态的Executor日志，如果报错位置位于用户业务代码，需要业务开发解决。其他错误信息您可以在搜索引擎中搜索相关报错信息。

Spark访问数据源网络不通怎么办？

默认情况下，DLA Spark不能访问用户VPC，只能访问OSS、Tablestore、MaxCompute等服务型数据源，并且不能访问公网。

DLA Spark可以通过挂载用户VPC网卡的方式来访问用户VPC中的数据源，以及访问公网。挂载用户网卡后，网络模型与一台普通用户VPC的ECS完全一样，可以复用阿里云VPC网络产品的所有功能，包括内网、公网、专线等等。如何挂载用户网卡，详情请参见文档配置数据源网络。

如何处理Spark作业日志中报错：ClassNotFound?

常见原因：一般是由于缺少类导致。您可以通过打开所上传的JAR包（jar tvf xxx.jar | grep xxxx），确认该类是否存在。

解决方案如下。

- 如果是用户业务相关的类，需要重新打包，确保目标类被打进去。
- 如果是第三方包，您可以采用如下方式。
  1. 使用Maven的Shade或者Assembly插件，加入依赖包（Shade和Assembly可以自行查找Maven相关资料）。
  2. 使用作业配置指南中“jars”参数将第三方JAR包单独上传。

如何处理Spark作业日志中报错：NoSuchMethod?

常见原因：出现了JAR包冲突，可能是引入了跟Spark冲突的第三方包导致。

解决方案：您可以通过Provided、Relocation等Maven常见的冲突解决方式来解决，相关技术属于通用技术，您可以进行搜索或者参考dla spark demo项目进行Maven Pom配置。

为什么Spark SQL作业使用show tables或者show database查询发现显示的列表不足?

首先请确认是否使用了DLA的元数据服务。

- 如果您使用的是DLA的元数据服务，需要确认权限问题。您只能看到自己具有权限的库表，其他库表隐藏。
- 如果您使用的不是DLA的元数据服务，需要确认自建元数据服务的权限和连通的正确性。

为什么在DLA SQL中执行select * from db1.table1成功， 但在Spark中报错?

详细报错如下。

```
java.lang.NullPointerException
 at java.net.URI$Parser.parse(URI.java:3042)
 at java.net.URI.<init>(URI.java:588)
 at org.apache.spark.sql.hive.client.JianghuClientImpl$$anonfun$getTableOption
```

常见原因如下。

- 表的存储格式不对，目前只支持OSS和HDFS。
- 表名中出现了中划线 `-` ，中划线为Spark SQL保留字，不允许在表名中使用。

为什么在DLA SQL中执行select * from db1.table1有数据，但在Spark中没有?

常见原因如下。

- 表中对应的OSS或者HDFS地址有嵌套关系。例如您指定的LOCATION为 `oss://db/table/` ，指定了partiion1和parition2，则您的数据文件应当类似 `oss//db/table/partition1=a/partiion2=b/data.csv` 。DLA SQL支持多层嵌套目录 `oss//db/table/partition1=a/partiion2=b/extral_folder/data.csv` ，但Spark SQL暂不支持。
- DLA SQL的语法和Spark SQL的语法略有区别，如何调试SQL，请参见Spark SQL。

为什么Spark作业提示错误日志oss object 403?

常见原因如下。

- DLA Spark不支持跨Region读JAR包或者读文件。
- 通过参数spark.dla.rolearn指定的Role没有读取这个OSS路径的权限。
- 文件地址填写错误。
- 文件与文件之间没有用逗号隔开或者写成JSON的List。

如何处理Spark SQL读JSON外表（包含日志投递自建）时的报错ClassNotFoundException: org.apache.hadoop.hive.serde2.JsonSerDe?

解决方案具体步骤如下。

1. 下载https://repo1.maven.org/maven2/org/apache/hive/hive-serde/3.1.2/hive-serde-3.1.2.jar上传至OSS。
2. 在Spark SQL开头添加语句 `add jar oss://path/to/hive-serde-3.1.2.jar;` 。

如何处理执行Spark SQL报错：Exception in thread "main" java.io.IOException: No FileSystem for scheme: oss?

解决方案：您可以在Spark SQL中添加如下命令。

```
set spark.dla.connectors=oss;
```

作业慢如何排查?

常见的作业执行慢可以分为两大类：

- 作业异常导致，您可以通过下面方法1和方法2排查。
- 作业无异常仍然很慢，您可以通过方法3、方法4和方法5排查。

1. 查看是否有Executor状态是Dead。

查看方式：单击展开对应作业的**操作**列表，点击**SparkUI**进入Spark页面，切换到**Executors**页面，查看**Status**字段。

解决方案：具体解决办法请参见Spark Executor出现Dead怎么处理。

2. 查看Driver日志，是否有异常信息导致Task终止并重试。

查看方式：单击展开对应作业的**操作**列表，点击**SparkUI**进入Spark页面，切换到**Executors**页面，找到**Executor ID**字段中为driver的stderr日志。

解决方案：通过报错信息查看异常的具体原因，大部分跟业务逻辑相关，您可以对其进行排查或者通过搜索引擎搜索解决。

> 注意
>
> 如果是OOM异常则需要检查业务逻辑是否有大内存占用，特别是某个字段特别大的情况，如果确实需要更大内存，您可以考虑使用更大规格的Executor或Driver节点。

3. 查看是否由资源不足导致。

查看方法：单击展开对应作业的**操作**列表，点击**SparkUI**进入**Spark**页面，切换到**Stages**页面，从Stage列表中，找到执行比较慢的一些Stage，查看这些Stage的并发度（Tasks: Succeeded/Total）。

如果Stage的Total比（Executor数目）*（Executor cores）多，那么就确认存在资源不足。

比如某个Stage Tasks Total是100，spark.executor.instances=5，spark.executor.resourceSpec=medium（2C8G），那么每一轮Tasks只能跑10个，需要跑10轮。

此时需要增加作业的资源总量，调大spark.executor.instances参数，或者调大Executor规格（spark.executor.resourceSpec）。最好不要超过同时运行的Stage的Tasks总数，否则会导致资源浪费。

4. 查看是否由GC导致。

查看方式：单击展开对应作业的**操作**列表，点击**SparkUI**进入**Spark**页面，切换到**Executors**页面，查看**Task Time (GC Time)**字段。

如果某些Executor GC Time比较多时，您可以采取如下解决方案。

- 优化逻辑。
- 增大Executor或Driver的节点规格。

5. 查看Driver或Executor堆栈慢在哪里。

查看方式：单击展开对应作业的**操作**列表，点击**SparkUI**进入**Spark**页面，切换到**Executors**页面，查看**Thread Dump**字段。

> 注意
>
> 堆栈只有在作业running状态才能查看。

解决方案：多次刷新堆栈查看作业运行卡在什么地方。

- 如果集中卡在某一个或者某几个函数调用上，那么说明该部分逻辑有热点或者实现效率比较低，您可以考虑优化该部分逻辑。
- 如果卡在Spark的某些调用中，您可以通过搜索引擎搜索解决。

6.跨云服务授权

6.1. DLA服务关联角色

在本文为主要介绍DLA服务关联角色（AliyunServiceRoleForOpenAnalytics）的应用场景以及如何删除服务关联角色。

背景信息

DLA服务关联角色（AliyunServiceRoleForOpenAnalytics）是在某些情况下，为了完成DLA自身的某个功能，需要获取其他各种各样的云服务的访问权限，而提供的RAM角色。更多信息请参见服务关联角色。

应用场景

DLA作为阿里云数据湖分析产品，提供Serverless Presto和Spark的核心产品功能，需要为用户打通、连接、关联各种各样的阿里云数据源和各种云服务产品（OSS、OTS、RDS、ADS、ODPS、ECS、VPC、RAM、MQ等），从而实现数据湖的各种各样的功能。因此，DLA会在用户开通DLA服务的时候，自动化的帮助用户在DLA内部创建好服务关联角色，从而极大的提高用户体验。

查看DLA服务关联角色

1. 登录Data Lake Analytics管理控制台。
2. 在**概览**页面右上角单击**选项**按钮。
3. 在**跨云服务授权**页面查看DLA服务关联角色信息：
   - 角色名称：AliyunServiceRoleForOpenAnalytics
   - 角色权限策略：AliyunServiceRolePolicyForOpenAnalytics
   - 权限说明如下：

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": "ram:DeleteServiceLinkedRole",
      "Resource": "*",
      "Effect": "Allow",
      "Condition": {
        "StringEquals": {
          "ram:ServiceName": "openanalytics.aliyuncs.com"
        }
      }
    },
    {
      "Action": [
        "ram:ListUsers",
        "ram:GenerateCredentialReport"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
```

```
            "oss:GetBucket",
            "oss:GetBucketAcl",
            "oss:GetBucketLocation",
            "oss:GetBucketInfo",
            "oss:GetBucketLogging",
            "oss:GetBucketWebsite",
            "oss:GetBucketReferer",
            "oss:GetBucketLifecycle",
            "oss:GetBucketEncryption",
            "oss:GetBucketStat",
            "oss:GetBucketMetadata",
            "oss:GetBucketTagging",
            "oss:GetBucketVersioning",
            "oss:GetSimplifiedObjectMeta",
            "oss:GetObjectMetadata",
            "oss:GetBucketStorageCapacity",
            "oss:GetBucketEncryption",
            "oss:GetObject",
            "oss:GetObjectMeta",
            "oss:GetObjectAcl",
            "oss:GetSymlink",
            "oss:GetObjectTagging",
            "oss:GetService",
            "oss:ListObjects",
            "oss:ListMultipartUploads",
            "oss:ListParts",
            "oss:ListBuckets",
            "oss:ListVpcip",
            "oss:ListVersions",
            "oss:GetBucketCname",
            "oss:GetBucketRequestPayment",
            "oss:GetBucketVpcip",
            "oss:DoesBucketExist",
            "oss:DoesObjectExist",
            "oss:ListObjectsV2",
            "oss:SelectObject",
            "oss:HeadObject",
            "oss:PutBucket",
            "oss:PutObject",
            "oss:PutObjectTagging",
            "oss:CopyObject",
            "oss:InitiateMultipartUpload",
            "oss:UploadPart",
            "oss:UploadPartCopy",
            "oss:CompleteMultipartUpload",
            "oss:AbortMultipartUpload",
            "oss:RestoreObject",
            "oss:PostObject",
            "oss:UploadFile",
            "oss:DownloadFile",
            "oss:AppendObject",
            "oss:DeleteObject",
            "oss:DeleteObjects"
        ],
        "Resource": "*"
```

```
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "alikafka:PUB"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "rds:DescribeDBInstances",
        "rds:DescribeDBInstanceAttribute",
        "rds:DescribeDBInstanceNetInfo",
        "rds:DescribeDBInstanceHAConfig",
        "rds:DescribeDBInstanceIPArrayList",
        "rds:ModifySecurityIps",
        "dds:DescribeDBInstances",
        "dds:DescribeDBInstanceAttribute",
        "dds:DescribeSecurityIps",
        "dds:ModifySecurityIps",
        "polardb:DescribeDBClusters",
        "polardb:DescribeDBClusterAttribute",
        "polardb:DescribeDBClusterEndpoints",
        "polardb:DescribeDBClusterAccessWhitelist",
        "polardb:ModifyDBClusterAccessWhitelist"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "mns:GetQueueAttributes",
        "mns:GetTopicAttributes",
        "mns:GetSubscriptionAttributes",
        "mns:ListQueue",
        "mns:ListTopic",
        "mns:ListSubscriptionByTopic",
        "mns:SendMessage",
        "mns:PublishMessage"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "mq:PUB"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "dbs:DescribeBackupPlanList",
```

```
        "dbs:DescribeFullBackupList",
        "dbs:DescribeIncrementBackupList",
        "dbs:DescribeRestoreTaskList",
        "dbs:DescribeBackupGatewayList"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "ots:GetRow",
        "ots:BatchGetRow",
        "ots:GetRange",
        "ots:GetShardIterator",
        "ots:GetStreamRecord",
        "ots:ListStream",
        "ots:ListTable",
        "ots:ListSearchIndex",
        "ots:DescribeStream",
        "ots:DescribeTable",
        "ots:DescribeSearchIndex",
        "ots:ComputeSplitPointsBySize",
        "ots:CreateTable",
        "ots:UpdateTable",
        "ots:DeleteTable",
        "ots:PutRow",
        "ots:UpdateRow",
        "ots:DeleteRow",
        "ots:BatchWriteRow",
        "ots:CreateIndex",
        "ots:DropIndex",
        "ots:CreateSearchIndex",
        "ots:DeleteSearchIndex",
        "ots:Search"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "log:ListProject",
        "log:ListLogStores",
        "log:ListShipper",
        "log:GetCursorOrData",
        "log:BatchGetLog",
        "log:GetShipper",
        "log:GetShipperConfig",
        "log:BatchGetLog",
        "log:DeleteShipper",
        "log:CreateShipper"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
```

```
    {
      "Action": [
        "ecs:CreateNetworkInterfacePermission",
        "ecs:DeleteNetworkInterfacePermission",
        "ecs:CreateNetworkInterface",
        "ecs:DescribeNetworkInterfaces",
        "ecs:DescribeSecurityGroups"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "vpc:DescribeVSwitches",
        "vpc:DescribeVpcs"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

删除服务关联角色

当您尝试删除服务关联角色（AliyunServiceRoleForOpenAnalytics）时，您需要进行如下操作：

- 关闭当前Region和其他所有Region的DLA服务，因为DLA是以用户账号维度来判断SLR的关联性。
- 删除服务关联角色，具体操作请参见删除服务关联角色。