ALIBABA CLOUD

Alibaba Cloud

Tablestore

Compute-Analysis

Document Version: 20220621

Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	**Danger:** Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	**Warning:** Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	**Notice:** If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	**Note:** You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.MaxCompute

1.1. Use MaxCompute to access Tablestore

This topic describes how to establish a seamless connection between Tablestore and MaxCompute that belongs to the same Alibaba Cloud account.

Background information

MaxCompute is a cost-effective and fully managed platform for petabytes of data warehousing. You can use this service to process and analyze large amounts of data in a fast and efficient manner. You can execute a simple Data Definition Language (DDL) statement to create an external table in MaxCompute. Then, you can use this table to associate MaxCompute with external data sources. This allows access to and output of data in various formats. MaxCompute tables can contain only structured data. However, external tables can contain either structured or non-structured data.

Tablestore and MaxCompute have their own type systems, and the following table lists their mappings.

Tablestore	MaxCompute
STRING	STRING
INTEGER	BIGINT
DOUBLE	DOUBLE
BOOLEAN	BOOLEAN
BINARY	BINARY

Preparations

Before you use MaxCompute to access Tablestore, make sure that the following preparations are made:

1. Activate MaxCompute.
2. Create a workspace.
3. Create an AccessKey pair for the RAM user.
4. Create a role in the RAM console to authorize MaxCompute to access Tablestore. For more information, see Cross-account authorization.
5. In the Tablestore console, create an instance and a table. For more information, see Create instances and Create tables.

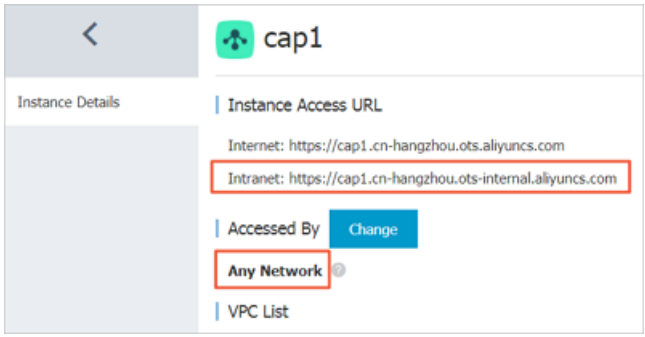
   In this example, the following content describes the information about the created instance and table:

   - Instance name: cap1
   - Table name: vehicle_track
   - Primary key information: vid(integer) and gt(integer)

- Endpoint: `https://cap1.cn-hangzhou.ots-internal.aliyuncs.com`

  > **Note** When you use MaxCompute to access Tablestore, we recommend that you use the internal network address of Tablestore.

- Set the network type to access the instance to **Any Network**.



Step 1: Install the client

1. Download the MaxCompute client package and download the package.

   > **Note** Make sure that your machine is installed with JRE 1.7 or later.

2. Edit the conf/odps_config.ini file, and configure the client, as shown in the following code:

```
access_id=********************
access_key=**********************
 # Specify the AccessKey ID and AccessKey secret of your Alibaba Cloud account. To obta
in the AccessKey ID and AccessKey secret, log on to the Alibaba Cloud Management Consol
e and view the AccessKey pair on the AccessKey Management page.
project_name=my_project
 # Specify the name of the project that you want to access.
end_point=https://service.odps.aliyun.com/api
 # Specify the endpoint of MaxCompute.
tunnel_endpoint=https://dt.odps.aliyun.com
 # Specify the link to access the MaxCompute Tunnel service.
log_view_host=http://logview.odps.aliyun.com
 # Specify the LogView URL that the client returns after a job is run. After you access
the LogView URL, you can view detailed information of the job.
https_check=true
 # Specify whether to enable access over HTTPS.
```

   > **Note** In the odps_config.ini file, the number sign (`#`) is used to add comments. The MaxCompute client uses `--` to add comments.

3. Run bin/odpscmd.bat. Enter `show tables;` .

   If the tables in the current MaxCompute projects are displayed, the preceding configurations are valid.

```
odps@ MCOTStest>show tables;

ALIYUN$document@aliyun-test.com:bank_data
ALIYUN$document@aliyun-test.com:result_table

OK
```

Step 2. Create an external table

Create a MaxCompute data table ots_vehicle_track and associate it with a Tablestore table vehicle_track.

Information of the associated Tablestore table:

- Instance name: cap1
- Table name: vehicle_track
- Primary key information: vid(int) and gt(int)
- Endpoint: `https://cap1.cn-hangzhou.ots-internal.aliyuncs.com`

```
CREATE EXTERNAL TABLE IF NOT EXISTS ots_vehicle_track
(
vid bigint,
gt bigint,
longitude double,
latitude double,
distance double ,
speed double,
oil_consumption double
)
STORED BY 'com.aliyun.odps.TableStoreStorageHandler' -- (1)
WITH SERDEPROPERTIES ( -- (2)
'tablestore.columns.mapping'=':vid, :gt, longitude, latitude, distance, speed, oil_consumpt
ion', -- (3)
'tablestore.table.name'='vehicle_track' -- (4)
)
LOCATION 'tablestore://cap1.cn-hangzhou.ots-internal.aliyuncs.com'; -- (5)
```

The following table lists the parameters.

Label	Parameter	Description
(1)	com.aliyun.odps.TableStoreStorageHandler	StorageHandler built in MaxCompute. StorageHandler processes Tablestore data. StorageHandler defines the interaction between MaxCompute and Tablestore. MaxCompute implements related logic.
(2)	SERDEPROPERITES	The operation that provides parameter options. The tablestore.columns.mapping and tablestore.table.name parameters are required for TableStoreStorageHandler.

Label	Parameter	Description
(3)	tablestore.columns.mapping	Required. The columns of the Tablestore table to be accessed by MaxCompute. The columns include primary key columns and attribute columns. Columns that contain colons (`:`) are the primary key columns of Tablestore. Examples: `:vid` and `:gt`. Other columns in the example are attribute columns. You must specify all primary key columns of the table in Tablestore when you specify the mapping. You need only to specify the attribute columns that MaxCompute accesses instead of specifying all attribute columns.
(4)	tablestore.table.name	The name of the Tablestore table to be accessed. If the specified name of the Tablestore table does not exist, an error occurs. MaxCompute does not proactively create Tablestore tables.
(5)	LOCATION	The information of the Tablestore instance to be accessed. The information includes the name of the instance and endpoint.

Step 3: Use an external table to access Tablestore data

After you create an external table, Tablestore data is imported to MaxCompute. Then, you can use MaxCompute SQL commands to access Tablestore data.

```
// Collect statistics for the average speed and fuel consumption of vehicles whose VIDs are smaller than 4 before the timestamp 1469171387.
select vid,count(*),avg(speed),avg(oil_consumption) from ots_vehicle_track where vid <4 and gt<1469171387  group by vid;
```

A similar output is returned.

```
odps@ analysis_vehicle>select vid,count(*), avg(speed),avg(oil_consumption) from ots_vehicle_track where vid <4 and gt<1469171387  group by vid;

ID = 20170306160538185gsvlupk2
Log view:
http://logview.odps.aliyun.com/logview/?h=http://service.odps.aliyun.com/api&p=analysis_vehicle&i=20170306160538185gsvlupk2&token=ajhxdXZNWWVPcTNrWkFOS0xpSGtoM2tFWGEwPSxPRFBTX
09CTzoxNjA0MzM3MzgzMTc0NjE5LDE0ODk0MjExMzgseyJTdGF0ZW1lbnQiOlt7IkFjdGlvbiI6WyJvZHBzOlJlYWQiXSwiRWZmZWN0IjoiQWxsb3ciLCJSZXNvdXJjZSI6WyJhY3M6b2RwczoqOnByb2plY3RzL2FuYWx5c2lzX3Zl
aGljbGUvaW5zdGFuY2VzLzIwMTcwMzA2MTYwNTM4MTg1Z3N2bHVwazIiXX1dLCJWZXJzaW9uIjoiMSJ9
Job Queueing.
--------------------------------------------------------------------------------
                STAGES        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  BACKUP
M1_job_0 .....    TERMINATED      1          1        0        0       0
R2_1_job_0 .....    TERMINATED      1          1        0        0       0
--------------------------------------------------------------------------------
STAGES: 02/02    [==========================>>] 100%  ELAPSED TIME: 51.18 s
--------------------------------------------------------------------------------
Summary:

+------------+------------+------------+------------+
| vid        | _c1        | _c2        | _c3        |
+------------+------------+------------+------------+
| 0          | 47         | 0.11622589796672624 | 6.5155061805308145 |
| 1          | 47         | 0.11200649789552555 | 6.5673983001829885 |
| 2          | 47         | 0.098971763146085 | 6.738738527883797 |
| 3          | 47         | 0.11503916531605494 | 6.6038902692751895 |
+------------+------------+------------+------------+
```

1.2. Cross-account authorization

This topic describes how to establish a seamless connection between Tablestore and MaxCompute across Alibaba Cloud accounts.

> **Note** To learn about how to connect Tablestore to MaxCompute within the same account, see Use MaxCompute to access Tablestore.

Preparations

Prepare two Alibaba Cloud accounts. If Account A grants access permissions to Account B, Account B can be used to access table data of Account A when you run MaxCompute. The following table describes the basic information.

> **Note** The following information is for reference only. Replace the information with actual information during operations.

Item	Tablestore	MaxCompute
Alibaba Cloud account name	Account A	Account B
UserId	12345	56789

Before you use MaxCompute to access Tablestore, make sure that the following preparations are made:

1. Activate MaxCompute and create a workspace for Account B. For more information, see MaxCompute and Create a workspace.
2. Create an AccessKey pair for each of Account A and Account B. For more information, see Create an AccessKey pair.
3. Use Account A to log on to the RAM console. On the **RAM Roles** page, create a RAM role. In the example, the created role name is AliyunODPSRoleForOtherUser.
4. In the list of RAM roles, find the AliyunODPSRoleForOtherUser role. Click the RAM role name, and set the policy content. Follow the code to set the policy content:

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

> **Note** Replace 1xxxx with your UID.

5. After the RAM role is created, view the Alibaba Cloud Resource Name (ARN) of the role on the **Basic Information** page.
6. Go back to the RAM console homepage. Go the **Policies** page. Click **Create Policy**.
7. On the **RAM Roles** page, find the AliyunODPSRoleForOtherUser role. Then, click **Add Permissions**.
8. In the Add Permissions panel, select the AliyunODPSRolePolicyForOtherUse permission. Click **OK**.
9. In the Tablestore console, create an instance and a table. For more information, see Create instances and Create tables.

   In this example, the following content describes the information about the created instance and table:

   - Instance name: cap1
   - Table name: vehicle_track
   - Primary key information: vid (integer) and gt (integer)
   - Endpoint: `https://cap1.cn-hangzhou.ots-internal.aliyuncs.com`

     > **Note** When you use MaxCompute to access Tablestore, we recommend that you use the internal network address of Tablestore.

   - Set the network type to access the instance to **Any Network**.

Use MaxCompute to access Tablestore

Repeat the steps described in Use MaxCompute to access Tablestore. You must specify a roleArn when you create an external table for cross-account access.

Account B is used to create an external table by using MaxCompute and access Tablestore by using the roleArn created in Preparations.

For more information about specific operations, see Use MaxCompute to access Tablestore. In Step 2, when you create an external table, use the following code:

```
CREATE EXTERNAL TABLE ads_log_ots_pt_external
(
vid bigint,
gt bigint,
longitude double,
latitude double,
distance double ,
speed double,
oil_consumption double
)
STORED BY 'com.aliyun.odps.TableStoreStorageHandler'
WITH SERDEPROPERTIES (
'tablestore.columns.mapping'=':vid, :gt, longitude, latitude, distance, speed, oil_consumpt
ion',
'tablestore.table.name'='vehicle_track',
'odps.properties.rolearn'='acs:ram::12345:role/aliyunodpsroleforotheruser'
)
LOCATION 'tablestore://cap1.cn-hangzhou.ots-internal.aliyuncs.com'
USING 'odps-udf-example.jar'
```

1.3. Allow MaxCompute to access Tablestore by using AccessKey pairs

In addition to account authorization, you can access data in Tablestore by using AccessKey pairs on MaxCompute.

Preparations

Get the AccessKey credentials (that is, AccessKeyId and AccessKeySecret) for the account that owns the target Tablestore resources. If the AccessKey pair is for a Resource Access Management (RAM) user, the RAM user must be granted at least the following permissions to operate the Tablestore resources:

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "ots:ListTable",
        "ots:DescribeTable",
        "ots:GetRow",
        "ots:PutRow",
        "ots:UpdateRow",
        "ots:DeleteRow",
        "ots:GetRange",
        "ots:BatchGetRow",
        "ots:BatchWriteRow",
        "ots:ComputeSplitPointsBySize"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
--You can also add other permissions.
```

Allow MaxCompute to access Tablestore by using AccessKey pairs

Unlike account authorization, you must specify the AccessKey pairs information in the `LOCATION` clause when you create an external table.

```
LOCATION 'tablestore://${AccessKeyId}:${AccessKeySecret}@${InstanceName}. ${Region}.ots-internal.aliyuncs.com'
```

Assume that the following information is what MaxCompute must access:

AccessKeyId	AccessKeySecret	Instance name	Region	Network mode
abcd	1234	cap1	cn-hangzhou	Intranet access

The statement to create the external table is:

```
CREATE EXTERNAL TABLE ads_log_ots_pt_external
(
vid bigint,
gt bigint,
longitude double,
latitude double,
distance double ,
speed double,
oil_consumption double
)
STORED BY 'com.aliyun.odps.TableStoreStorageHandler'
WITH SERDEPROPERTIES (
'tablestore.columns.mapping'=':vid, :gt, longitude, latitude, distance, speed, oil_consumpt
ion',
'tablestore.table.name'='vehicle_track'
)
LOCATION 'tablestore://abcd:1234@cap1.cn-hangzhou.ots-internal.aliyuncs.com'
```

For more information about accessing data, see the "Step 3: Use an external table to access Tablestore data" section of the Use MaxCompute to access Tablestore topic.

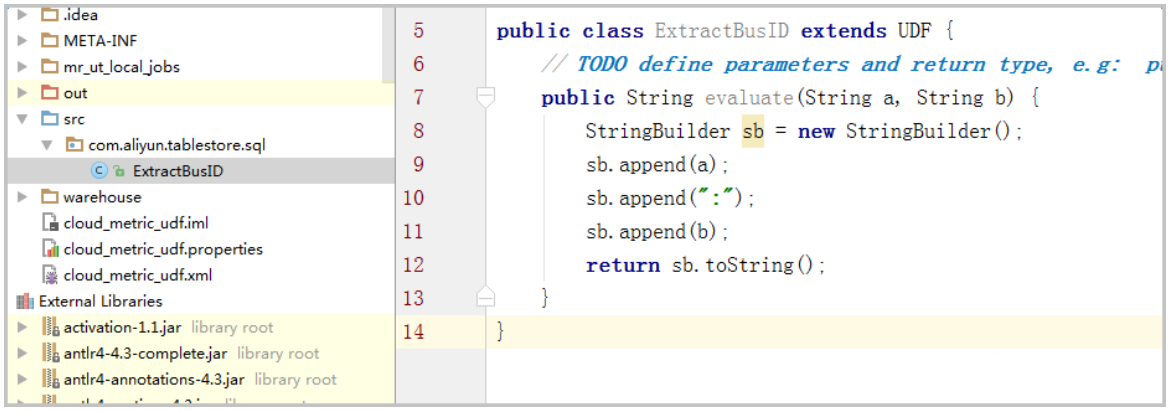
1.4. Use UDFs to process data

If your data stored in Tablestore is uniquely structured and you want to define development logic to process each row of data, such as parsing a specific JSON string, you can use User Defined Function (UDF).

Procedure

1. Install the MaxCompute-Java/MaxCompute-Studio plug-in in IntelliJ. For more information, see What is Studio. After the plug-in is installed, you can start development.

   The following figure shows a simple UDF definition, which connects two strings. MaxCompute supports more complex UDF, such as user-defined window execution logic. For more information, see Develop and debug UDF.

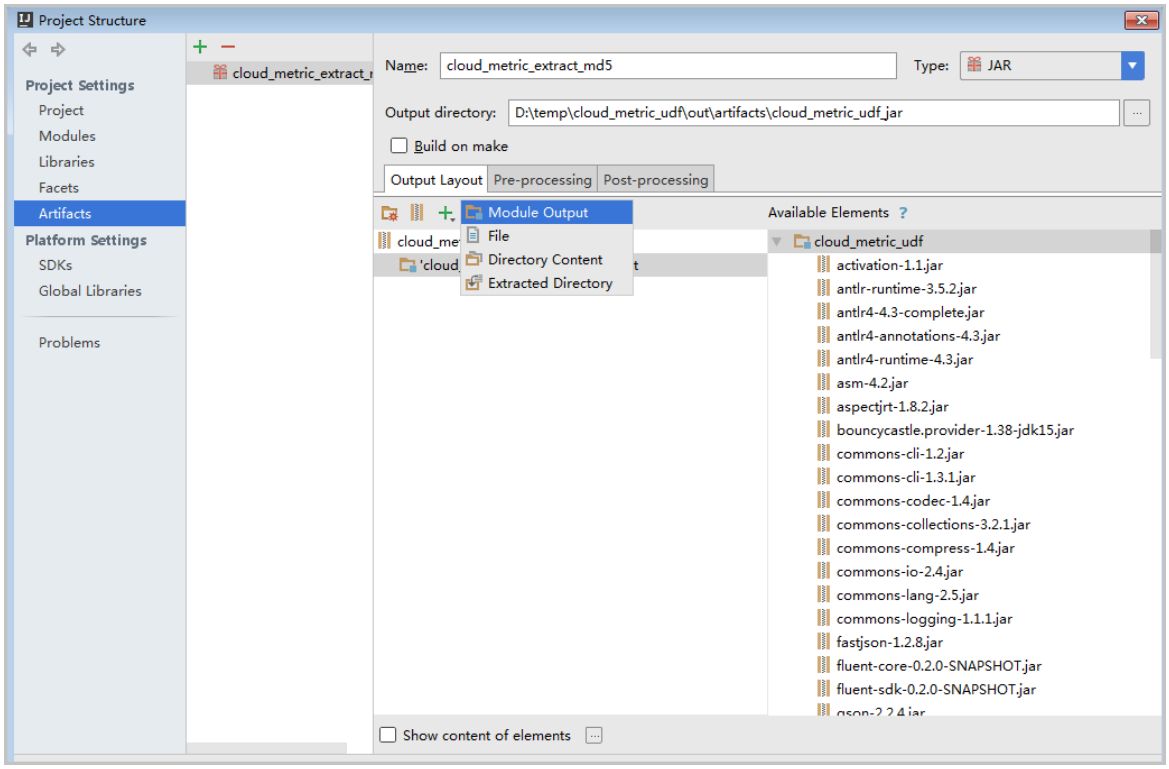


2. Install the MaxCompute-Java/MaxCompute-Studio plug-in in IntelliJ. For more information, see What is Studio. After the plug-in is installed, you can start development.

The following figure shows a simple UDF definition, which connects two strings. MaxCompute supports more complex UDF, such as user-defined window execution logic. For more information, see Develop and debug UDF.

3. Upload the package to MaxCompute after the resource is packaged.

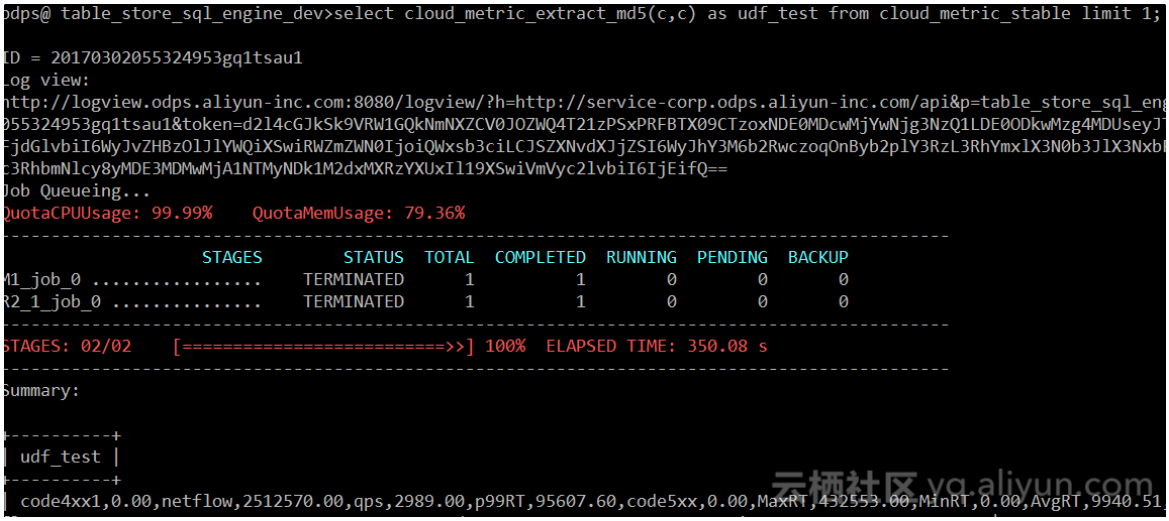
   Choose **File > Project Structure > Artifacts**. Enter the *name* and *output directory*. Then, click the **+** icon to select the output module. After the resource is packaged, use MaxCompute Project Explorer to upload the resource and create a function. Then, you can call the function in SQL.



4. Run **bin/odpscmd.bat**.

```
// Select a row of data and pass in name/name to the UDF. A connection of the two strin
gs is returned.
select cloud_metric_extract_md5(name, name) as udf_test from test_table limit 1;
```

A similar output is returned.



1.5. FAQ

This topic describes the frequently asked questions about the use of MaxCompute to access Tablestore.

- FAILED: ODPS-0010000:System internal error - fuxi job failed, WorkerPackageNotExist

  Cause: You must set odps.task.major.version to unstructured_data.

- FAILED: ODPS-0010000:System internal error - std::exception:Message: a timeout was reached

  Cause: The endpoint to access Tablestore is invalid, which causes MaxCompute to fail to access Tablestore.

- logview invalid end_point

  Cause: In the execution process, the logview URL is returned. If an error is returned when a browser is used to access the address, the configurations may be invalid. Check MaxCompute configurations.

If the problem persists, submit a ticket.

2.Spark/SparkSQL

2.1. Overview

Spark helps you perform complex computing and efficient analysis on Tablestore data by using E-MapReduce (EMR) SQL or DataFrame.

Features

In addition to basic features, Tablestore Spark connector provides the following core features for batch computing:

- Index selection: An appropriate index determines the efficiency of data queries. You can select an index that best matches filter conditions to increase the query efficiency. Tablestore allows you to choose between global secondary index and search index.
- Partition pruning: This feature allows you to filter unnecessary splits in advance by using fine-grained configurations based on filter conditions, which reduces the amount of data sent from the server.
- Projection column and Filter pushdown: This feature allows you to push down the Projection column and Filter condition to the server, which reduces the amount of data sent from each partition.
- Dynamical adjustment of the size of each split: This feature allows you to adjust the size of each split and the number of splits. Each split is bound to the partition of a Resilient Distributed Dataset (RDD), which accelerates the execution of Spark tasks.

  > **Note**
  >
  > You can call the ComputeSplitsBySize operation to obtain the splits. This operation splits all data in the table into splits based on the specified size and returns the split points between these splits and the information of the machine where the splits are distributed. This operation is used to execute plans such as concurrency plans on computing engines.

Stream computing uses change data capture (CDC) to complete streaming consumption and computing in micro-batch mode of Spark based on Overview. Meanwhile, at-least-once semantics is provided. In stream computing, one split is bound to one partition of an RDD. Partitions of a table can be scaled out to implement the linear scalability of data throughput.

Use EMR SQL or DataFrame

You can use one of the following methods to access Tablestore by using Spark: Use EMR SQL and DataFrame.

- Use EMR SQL

  This method uses standard SQL statements to access and perform operations on business data. You can use this method to implement seamless migration of existing business logic.

- Use DataFrame

  This method requires programming knowledge. However, you can combine this method with other features to execute complex business logic, which is suitable for complex and flexible scenarios.

Data access method

Tablestore provides the following methods for batch computing of Spark to access data: KV-based queries from a table or global search index and search index-based queries. This way, in addition to a wide range of query and analysis capabilities, Tablestore allows you to read and write large amounts of data.

Differences between the two data access methods:

- KV-based queries implement high efficiency when the fields specified for filtering are primary key columns. However, this method is not suitable when the fields specified for filtering are non-primary key columns because these field values often change. In addition, KV-based queries do not support geo query.
- Search index-based queries apply to the following data access scenarios.

  > **Note**
  >
  > Based on inverted indexes and column-oriented storage, search index provides query and analysis features such as full-text search, fuzzy query, geo query, and aggregation that are similar to those of Elasticsearch.

  - A few real-time data analysis scenarios that have high requirements on latencies.
  - Multiple fields specified for filtering are non-primary key columns. These fields cannot be contained by the primary key of the global secondary index or table.
  - Filtering efficiency is high when fields are used for filtering. A field can be used to filter out most data.

    For example, in `select * from table where col = 1000;`, col indicates the non-primary key column. The condition col = 1000 can be used to filter out most data.
  - Query conditions contain fields for geo query.

The following section uses a figure and SQL statement `select * from table where col1 like 'A%' or col2 = 'a';` to describe how to use the two query types.

- When you use search index to access data, you can obtain a row of data (pk1 = 1) that contains the col1 column whose value is 'A%' from the search index. You can also obtain two rows of data (pk1 = 1 and pk1 = 2) that contain the col2 column whose value is 'a' from the search index. Then, the two results are concatenated by using union to obtain data that meets conditions (`pk1 = 1,col1 = 'Alibaba Cloud',col2 = 'a'`).
- When you use KV-based queries, data is queried from the Tablestore table. The table can be queried only by using primary key columns. If the field specified in the SQL statement is not a primary key column of the table, the whole table must be scanned.

  col1 is not the primary key column of the table. Therefore, Tablestore scans the whole table to search for a row of data that contains the col2 column whose value is 'A%'. col2 is not the primary key column of the table. Therefore, Tablestore scans the whole table again to search for two rows of data that contain the col2 column whose value is ''a''. Then, the two results are concatenated by using union.

  You can also create an index table where the primary key columns are col1 and col2. However, this method reduces flexibility.

2.2. Data types

This topic describes the mappings of data and value types between Spark, Scala, as well as the search indexes and tables of Tablestore. When you use these data and value types, you must follow the mapping rules for Spark, Scala, and Tablestore.

Basic data types

The following table describes the supported basic data types.

Data type in Spark	Value type in Scala	Data type in search indexes	Data type in tables
ByteType	Byte	Long	Integer
ShortType	Short	Long	Integer
IntegerType	Int	Long	Integer
LongType	Long	Long	Integer
FloatType	Float	Double	Double
DoubleType	Double	Double	Double
StringType	String	Keyword/Text	String
BinaryType	Array[Byte]	Binary	Binary
BooleanType	Boolean	Boolean	Boolean
String JSON (geographical coordinates)	String (JSON)	*Geopoint*	String (JSON)

Geographical location data types

Search indexes support geo query. When a geo query is pushed down to the compute layer, Spark can query and analyze data based on geographical locations in addition to basic types of data.

Geo query provides the following query types: geo-distance query, geo-bounding box query, and geo-polygon query. Geo query can be used to query information about the locations of IoT devices, locations of food delivery orders, locations of employees when they clock in or out, and locations of express delivery orders. You can use the following methods to perform geo query:

- Use the search indexes of Tablestore. For more information, see Geo-distance query.
- Use Spark SQL.
  - Use geo-distance query by specifying the central point and the radius for the query.

```
select * from table where  val_geo = '{"centerPoint":"3,0", "distanceInMeter": 100000}'
and name like 'ali%'
```

  - Use geo-bounding box query.

```
select * from table where geo = '{"topLeft":"8,0", "bottomRight": "0,10"}' and id in {
123 , 321 }
```

  - Use geo-polygon query.

```
select * from table where geo = '{"points":["5,0", "5,1", "6,1", "6,10"]}'
```

The following table describes the geographical location data types supported by Spark, Scala, search indexes, and tables.

Data type in Spark	Value type in Scala	Data type in search indexes	Data type in tables
String JSON (coordinates of the central point and the radius of a circular geographical area)	String (JSON)	Geopoint	STRING (JSON)
STRING JSON (coordinates of the vertices in a rectangular geographical area)	String (JSON)	Geopoint	STRING (JSON)
String JSON (coordinates of the vertices in a polygonal geographical area)	String (JSON)	Geopoint	STRING (JSON)

2.3. E-MapReduce SQL

2.3.1. Batch computing

You can use Spark SQL on an E-MapReduce (EMR) cluster to access Tablestore. Compared with batch computing, Tablestore Spark connector allows you to select indexes, prune partitions, perform Projection column and Filter pushdown, and dynamically specify the sizes of partitions. In addition, Tablestore Spark connector uses global secondary index or search index of Tablestore to accelerate queries.

Prerequisites

- An EMR Hadoop cluster is created. For more information, see Create a cluster.

  When you create a cluster, make sure that **Assign Public IP Address** is enabled to access the cluster over the Internet and Remote Logon is enabled to log on to a remote server by using Shell.

  > **Note**
  >
  > This topic uses Shell commands. For more information about how to use the graphical interfaces of EMR to implement data development, see Manage projects.

- The emr-datasources_shaded_2.11-2.2.0-SNAPSHOT.jar package is uploaded to the EMR Header server.

Use Spark to connect to a Tablestore table and global secondary index

After Spark is connected to a Tablestore table and global secondary index, the system selects an index table based on the column conditions for queries.

1. Create a table or global secondary index at the Tablestore side.
   i. Create a Tablestore data table. For more information, see Overview.

      In the example, the name of the table is tpch_lineitem_perf. The primary key columns are l_orderkey (type: LONG) and l_linenumber (type: LONG). Part of the 14 attribute columns are l_comment (type: STRING), l_commitdate (type: STRING), l_discount (type: DOUBLE), l_extendedprice (type: DOUBLE), l_linestatus (type: STRING), l_partkey(type: LONG), l_quantity(type: DOUBLE), and l_receiptdate (type: STRING). The number of data entries are 384,016,850.

   ii. (Optional). Create a global secondary index on the data table. For more information, see Use SDKs.

      > **Note**
      >
      > To use non-primary key column columns in the query conditions, we recommend that you create a global secondary index to accelerate queries.

      Global secondary index allows you to create an index table based on specified columns. Data in the generated index table is sorted by the specified indexed columns. All data written to the base table is automatically synchronized to the index table in an asynchronous manner.

2. Create an external table in Spark SQL at the EMR cluster side.
   i. Log on to the EMR Header server.

ii. Run the following command to start the command line of Spark SQL. You can use the command line to create an external table in Spark SQL and perform SQL-related operations.

The standard parameters to start Spark is in the `--num-executors 32 --executor-memory 2g --executor-cores 2` format. You can adjust the parameter values based on the specific cluster configurations. <Version> specifies the version information of the uploaded JAR package. Example: 2.1.0-SNAPSHOT.

```
spark-sql --jars emr-datasources_shaded_2.11-<Version>.jar --master yarn --num-exec
utors 32 --executor-memory 2g --executor-cores 2
```

iii. Connect to the global secondary index when you create the external table in Spark SQL.

- Parameters

Parameter	Description
endpoint	The endpoint to access the Tablestore instance. A VPC address is used to access the EMR cluster.
access.key.id	The AccessKey ID of your Alibaba Cloud account.
access.key.secret	The AccessKey secret of your Alibaba Cloud account.
instance.name	The description of the instance.
table.name	The name of the data table.
split.size.mbs	The size of each split. Default value: 100. Unit: MB.
max.split.count	The maximum number of splits calculated for the table. The number of concurrent tasks corresponding to the number of splits. Default value: 1000.
catalog	The schema of the table.

- Examples

```
DROP TABLE IF EXISTS tpch_lineitem;
CREATE TABLE tpch_lineitem
USING tablestore
OPTIONS(
endpoint="http://vehicle-test.cn-hangzhou.vpc.tablestore.aliyuncs.com",
access.key.id="",
access.key.secret="",
instance.name="vehicle-test",
table.name="tpch_lineitem_perf",
split.size.mbs=10,
max.split.count=1000,
catalog='{"columns":{"l_orderkey":{"type":"long"},"l_partkey":{"type":"long"},"l_
suppkey":{"type":"long"},"l_linenumber":{"type":"long"},"l_quantity":{"type":"dou
ble"},"l_extendedprice":{"type":"double"},"l_discount":{"type":"double"},"l_tax":
{"type":"double"},"l_returnflag":{"type":"string"},"l_linestatus":{"type":"string
"},"l_shipdate":{"type":"string"},"l_commitdate":{"type":"string"},"l_receiptdate
":{"type":"string"},"l_shipinstruct":{"type":"string"},"l_shipmode":{"type":"stri
ng"},"l_comment":{"type":"string"}}}'
);
```

3. Perform queries by using SQL statements

   The following section provides examples on SQL queries to address different query needs:

   - Full table query
     - SQL statement: `SELECT COUNT(*) FROM tpch_lineitem;`
     - Total duration for running the SQL statement: 36.199s, 34.711s, and 34.801s. Average duration: 35.237s.
   - Primary key query
     - SQL statement:

       ```
       SELECT COUNT(*) FROM tpch_lineitem WHERE l_orderkey = 1 AND l_linenumber = 1;
       ```

     - Tablestore server: average duration of 0.585 ms for the GetRow operation
   - Non-primary key columns query without using global secondary index
     - SQL statement: `SELECT count(*) FROM tpch_lineitem WHERE l_shipdate = '1996-06-06';`
     - Total duration for running the SQL statement: 37.006s, 37.269s, and 37.17s. Average duration: 37.149s.
   - Non-primary key columns query by using global secondary index
     - SQL statement: `SELECT count(*) FROM tpch_lineitem WHERE l_shipdate = '1996-06-06';`
     - Total duration for running the SQL statement when global secondary index is enabled for the l_shipdate column: 1.686s, 1.651s, and 1.784s. Average duration: 1.707s

Use Spark to connect to a Tablestore table and search index

After Spark is connected to a Tablestore table and search index, the system selects an index table based on the column conditions for queries.

1. Create a table and search index at the Tablestore side.
   i. Create a data table. For more information, see Overview.

      In the example, the name of the table is geo_table. The primary key column is pk1 (type: STRING). The attribute columns are val_keyword1 (type: STRING), val_keyword2 (type: STRING), val_keyword3 (type: STRING), val_bool (type: BOOLEAN), val_double (type: DOUBLE), val_long1 (type: LONG), val_long2 (type: LONG), val_text (type: STRING), and val_geo (type: STRING). The number of data entries is 208,912,382.

   ii. A search index is created on the table. For more information, see Create and use search indexes.

      When you create a search index, configure mappings for the search index based on field types.

      > **Note**
      >
      > When you create a search index, select Geographical Location instead of STRING for a GEOPOINT field in the search index.

      After you create a search index, the search index synchronizes data from the table. When the search index enters the incremental state, the search index is created.

2. Create an external table by using Spark at the EMR cluster side.
   i. Log on to the EMR Header server.
   ii. Connect to the search index when you create the external table in Spark SQL.
      - Parameters

Parameter	Description
endpoint	The endpoint to access the Tablestore instance. A VPC address is used to access the EMR cluster.
access.key.id	The AccessKey ID of your Alibaba Cloud account.
access.key.secret	The AccessKey secret of your Alibaba Cloud account.
instance.name	The description of the instance.
table.name	The name of the Tablestore table.
search.index.name	The name of the search index.

Parameter	Description
max.split.count	The maximum number of concurrent tasks used by parallel scan of search index. The maximum number of concurrent tasks used by parallel scan corresponds to that of splits in Spark.
push.down.range.long	Specifies whether to push down predicates where operators (>= > < <=) are used to compare values with LONG column values. For more information, see Configure predicate pushdown for batch computing. Type: Boolean. The default value is true, which indicates that predicates where operators (>= > < <=) are used to compare values with LONG column values are pushed down. A value of false indicates that predicates where operators (>= > < <=) are used to compare values with LONG column values are not pushed down. For more information, see.
push.down.range.string	Specifies whether to push down predicates where operators (>= > < <=) are used to compare values with STRING column values. For more information, see Configure predicate pushdown for batch computing. Type: Boolean. The default value is true, which indicates that predicates where operators (>=, >, <, and <=) that are used to compare values with STRING column values are pushed down. A value of false indicates that predicates where operators (>=, >, <, and <=) are used to compare values with STRING column values are not pushed down.

- Examples

```
DROP TABLE IF EXISTS geo_table;
CREATE TABLE geo_table (
pk1 STRING, val_keyword1 STRING, val_keyword2 STRING, val_keyword3 STRING,
val_bool BOOLEAN, val_double DOUBLE, val_long1 LONG, val_long2 LONG,
val_text STRING, val_geo STRING COMMENT "geo stored in string format"
)
USING tablestore
OPTIONS(
endpoint="https://sparksearchtest.cn-hangzhou.vpc.tablestore.aliyuncs.com",
access.key.id="",
access.key.secret="",
instance.name="sparksearchtest",
table.name="geo_table",
search.index.name="geo_table_index",
max.split.count=64,
push.down.range.long = false,
push.down.range.string = false
);
```

3. Perform queries by using SQL statements

   The following section provides examples on SQL queries to address different query needs:

   - Full table query by using the search index
     - SQL statement: `SELECT COUNT(*) FROM geo_table;`
     - Duration for running the SQL statement: Actual duration: 165.208s. Average QPS: 1,264,500. Data entries for testing: 208,912,382. Number of concurrent tasks by using parallel scan: 64.

```
208912382
Time taken: 165.208 seconds, Fetched 1 row(s)
20/06/29 20:55:11 INFO [main] SparkSQLCLIDriver: Time taken: 165.208 seconds, Fetch
ed 1 row(s)
```

   - Boolean query
     - SQL statement:

```
 SELECT val_long1, val_long2, val_keyword1, val_double FROM geo_table WHERE
(val_long1 > 17183057 AND val_long1 < 27183057) AND (val_long2 > 1000 AND val_long2 <
5000) LIMIT 100;
```

     - Duration for running the SQL statement: Actual duration: 2.728s. Spark pushes the Projection column and Filter to the search index, which accelerates query efficiency.

```
21423964        4017    aaa     2501.9901650365096
21962236        2322    eio     2775.9021545044116
Time taken: 2.894 seconds, Fetched 100 row(s)
20/06/30 18:51:24 INFO [main] SparkSQLCLIDriver: Time taken: 2.894 second
```

   - Geo query

Geo query includes the following query types: geo-distance query, geo-bounding box query, and geo-polygon query. In the example, val_geo indicates the name of the GEOPOINT field. The coordinate pair of a geographical location is in the format of "latitude,longitude".

- Geo-distance query

  Syntax: val_geo = '{"centerPoint":"the coordinate pair of the central point", "distanceInMeter": the distance from the central point'}'

  SQL statement:

```
SELECT COUNT(*) FROM geo_table WHERE val_geo =
'{"centerPoint":"6.530045901643962,9.05358919674954", "distanceInMeter": 3000.0}';
```

- Geo-bounding box query

  Syntax: val_geo = '{"topLeft":"the coordinate of the upper-left corner in the rectangular area", "bottomRight": "the coordinate of the lower-right corner in the rectangular area"}'

  SQL statement:

```
SELECT COUNT(*) FROM geo_table WHERE val_geo =
'{"topLeft":"6.257664116603074,9.1595116589601", "bottomRight":
"6.153593333442616,9.25968497923747"}';
```

- Geo-polygon query

  Syntax: val_geo = '{"points":["Coordinate Pair 1", "Coordinate Pair 2", ... "Coordinate Pair n-1", "Coordinate Pair n"]}'

  SQL statement:

```
SELECT COUNT(*) FROM geo_table WHERE val_geo = '{"points":
["6.530045901643962,9.05358919674954", "6.257664116603074,9.1595116589601",
"6.160393397574926,9.256517839929597", "6.16043846779313,9.257192872563525"]}';
```

2.3.2. Stream computing

You can use Spark SQL in an E-MapReduce (EMR) cluster to access Tablestore. Then, EMR uses Change Data Capture (CDC) to complete micro-batch stream consumption and computing for Spark based on Tunnel Service. At-least-once semantics are also provided.

Prerequisites

- An EMR Hadoop cluster is created. For more information, see Create a cluster.

  When you create a cluster, make sure that **Assign Public IP Address** is enabled to access the cluster over the Internet and Remote Logon is enabled to log on to a remote server by using Shell.

  > **Note**
  >
  > This topic uses Shell commands. For more information about how to use the graphical interfaces of EMR to implement data development, see Manage projects.

- The emr-datasources_shaded_2.11-2.2.0-SNAPSHOT.jar package is uploaded to the EMR Header server.

Quick start

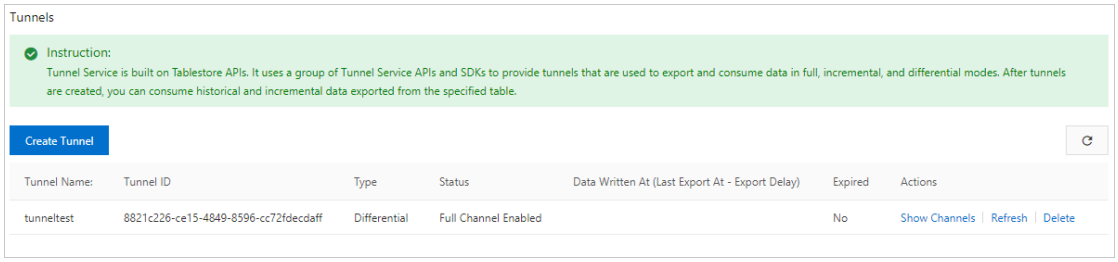
1. Create tables and tunnels in Tablestore.
   i. Create a Source table and a Sink table in the Tablestore console. For more information, see Quick start of Tablestore.

      The name of the Source table is OrderSource. The primary key columns are UserId and OrderId. The attribute columns are price and timestamp. The following figure shows an example of the Source table.

   ii. The name of the Sink table is OrderStreamSink. The primary key columns are begin and end. The attribute columns are count and totalPrice. The start time and end time are displayed in the format of yyyy-MM-dd HH:mm:ss. Example: 2019-11-27 14:54:00.

   iii. Create a tunnel for the Source table. For more information, see Quick start.

      The name, ID, and type of the tunnel are displayed in the Tunnels section. The tunnel ID is used for stream processing.



2. Create a Spark external table on the EMR cluster side.
   i. Log on to the EMR Header server.

   ii. Run the following command to start the command line of Spark SQL. You can use the command line to create an external table in Spark SQL and perform SQL-related operations.

      The standard parameters to start Spark is in the `--num-executors 32 --executor-memory 2g --executor-cores 2` format. You can adjust the parameter values based on the specific cluster configurations. <Version> specifies the version information of the uploaded JAR package. Example: 2.1.0-SNAPSHOT.

```
spark-sql --jars emr-datasources_shaded_2.11-<Version>.jar --master yarn --num-exec
utors 32 --executor-memory 2g --executor-cores 2
```

   iii. Create an external Source table, which is named order_source. This table corresponds to the OrderSource table in Tablestore.

- Parameters

Parameter	Description
endpoint	The endpoint of the Tablestore instance. The EMR cluster uses a virtual private cloud (VPC) address to access the Tablestore instance.
access.key.id	The AccessKey ID of your Alibaba Cloud account.
access.key.secret	The AccessKey secret of your Alibaba Cloud account.
instance.name	The name of the instance.
table.name	The name of the Tablestore data table.
catalog	The schema of the Tablestore data table.

- Example

```
DROP TABLE IF EXISTS order_source;
CREATE TABLE order_source
USING tablestore
OPTIONS(
endpoint="http://vehicle-test.cn-hangzhou.vpc.tablestore.aliyuncs.com",
access.key.id="",
access.key.secret="",
instance.name="vehicle-test",
table.name="OrderSource",
catalog='{"columns": {"UserId": {"type": "string"}, "OrderId": {"type": "string"}
,"price": {"type": "double"}, "timestamp": {"type": "long"}}}'
);
```

3. Perform real-time stream computing.

   Real-time stream computing collects statistics for the number of orders and the order amount within a time window and writes the aggregate results to the Tablestore data table.

   i. Create an external Sink table, which is named order_stream_sink. This table corresponds to the OrderStreamSink table in Tablestore.

      The parameter configurations for creating an external Sink table and an external Source table differ only in the catalog field.

   ii. Create a view on the order_source table.

      The ID of the tunnel that you created for the Source table in Tablestore is required when you create the view.

iii. Run a Stream SQL job to perform real-time aggregation and write the aggregate results to the OrderStreamSink table in Tablestore in real time.

```
// Create an external Sink table, which is named order_stream_sink. This table correspo
nds to the OrderStreamSink table in Tablestore.
DROP TABLE IF EXISTS order_stream_sink;
CREATE TABLE order_stream_sink
USING tablestore
OPTIONS(
endpoint="http://vehicle-test.cn-hangzhou.vpc.tablestore.aliyuncs.com",
access.key.id="",
access.key.secret="",
instance.name="vehicle-test",
table.name="OrderStreamSink",
catalog='{"columns": {"begin": {"type": "string"},"end": {"type": "string"}, "count": {
"type": "long"}, "totalPrice": {"type": "double"}}}'
);

// Create a view named order_source_stream_view on the order_source table.
CREATE SCAN order_source_stream_view ON order_source USING STREAM
OPTIONS(
tunnel.id="4987845a-1321-4d36-9f4e-73d6db63bf0f",
maxoffsetsperchannel="10000");

// Run a Stream SQL job on the order_source_stream_view. The following code provides an
example on how to aggregate the number of orders and the order amount at an interval of
30 seconds.
// Write the aggregate results to the OrderStreamSink table in Tablestore in real time.

CREATE STREAM job1
options(
checkpointLocation='/tmp/spark/cp/job1',
outputMode='update'
)
INSERT INTO order_stream_sink
SELECT CAST(window.start AS String) AS begin, CAST(window.end AS String) AS end, count(
*) AS count, CAST(sum(price) AS Double) AS totalPrice FROM order_source_stream_view GRO
UP BY window(to_timestamp(timestamp / 1000), "30 seconds");
```

You can obtain the aggregate results after you run Stream SQL. The aggregate results are saved in the OrderStreamSink table. The following figure shows an example of the aggregate results.

2.4. DataFrame

2.4.1. Batch computing

You can use Spark DataFrame to access Tablestore and perform debugging on the local computer and cluster.

Prerequisites

- You have understood the dependent packages that are used for Spark to access Tablestore. These dependent packages have been imported to the project by using maven.

- Spark related dependent packages: spark-core, spark-sql, and spark-hive
- Spark Tablestore connector: emr-tablestore-<version>.jar
- Tablestore SDK for Java: tablestore-<version>-jar-with-dependencies.jar

<version> indicates the version number of the corresponding dependent package. Use the actual version number.

- A table is created at the Tablestore side. For more information, see Create a data table.
- An AccessKey pair that consists of an AccessKey ID and an AccessKey secret is obtained. For more information, see Obtain an AccessKey pair.

Quick start

The following section describes how to use batch computing by using project samples.

1. Download the source code of the project sample from GitHub. For more information about the path to download the code, visit TableStoreSparkDemo.

   The project contains the complete dependencies and samples on how to use batch computing. For more information about the dependencies, see the pom file in the project.

2. Read the README document of the TableStoreSparkDemo project. Install Spark Tablestore connector and Tablestore SDK for Java of the latest versions to the local maven library.

3. Modify the sample code.

   TableStoreBatchSample is used in the example. Descriptions of the core sample code:

   - format("tablestore") indicates that the ServiceLoader method is used to load Spark Tablestore connector. For more information about configurations, see META-INF.services in the project.
   - instanceName indicates the name of the Tablestore instance. tableName indicates the name of the table. endpoint indicates the endpoint of the instance. accessKeyId indicates the AccessKey ID of the Alibaba Cloud account. accessKeySecret indicates the AccessKey secret of the Alibaba Cloud account.
   - catalog is a JSON string, which includes field names and types. In the following example, the table has the following fields: salt (type: Long), UserId (type: String), OrderId (type: String), price (type: Double), and timestamp (type: Long).

     In the latest version, you can use the Schema method to replace the catalog configurations. Select a method based on the version.

   - split.size.mbs: optional. This parameter indicates the size of each split. Default value: 100. Unit: GB.

     A smaller value indicates more splits and tasks for Spark.

```
    val df = sparkSession.read
    .format("tablestore")
    .option("instance.name", instanceName)
    .option("table.name", tableName)
    .option("endpoint", endpoint)
    .option("access.key.id", accessKeyId)
    .option("access.key.secret", accessKeySecret)
    .option("split.size.mbs", 100)
    .option("catalog", dataCatalog)
    // The latest version allows you to replace catalog configurations with the Schem
a method.
    //.schema("salt LONG, UserId STRING, OrderId STRING, price DOUBLE, timestamp LONG
")
    .load()

  val dataCatalog: String =
    s"""
       |{"columns": {
       |    "salt": {"type":"long"},
       |    "UserId": {"type":"string"},
       |    "OrderId": {"type":"string"},
       |    "price": {"type":"double"},
       |    "timestamp": {"type":"long"}
       | }
       |}""".stripMargin
```

Debugging

After the sample code is modified, you can perform debugging on the local computer or Spark cluster. TableStoreBatchSample is used in the example to describe the debugging process.

- Perform debugging on the local computer

  IntelliJ IDEA is used in the example.

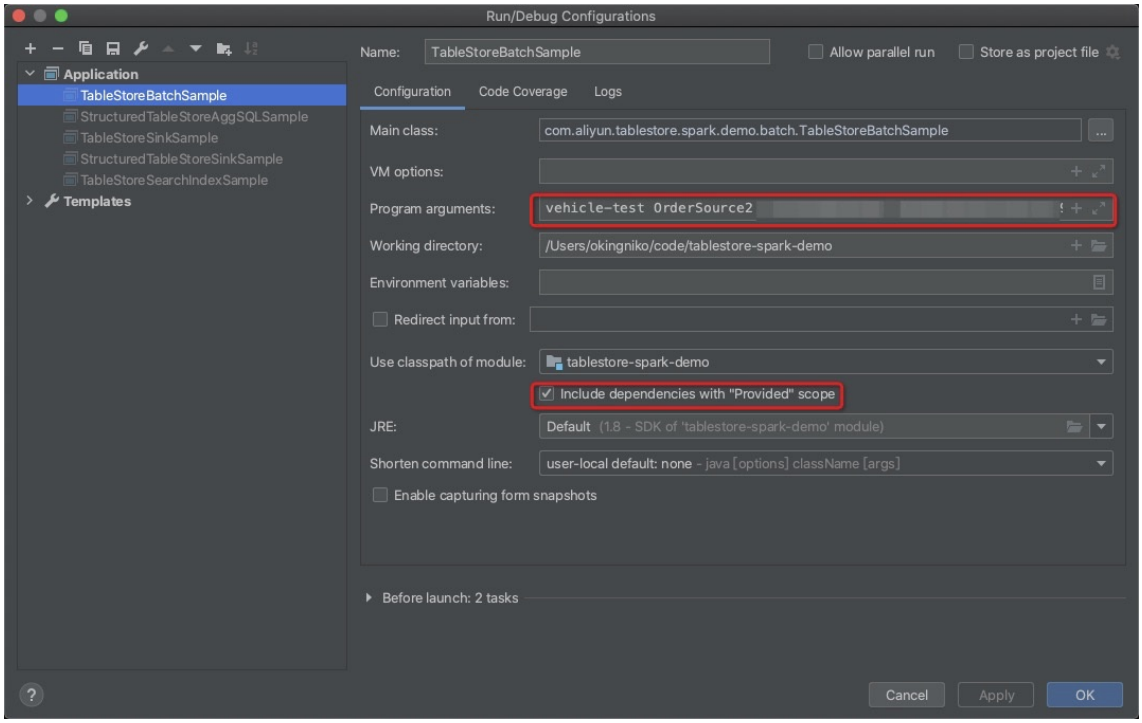
  > **Note**
  >
  > The environments used for testing are Spark 2.4.3, Scala 2.11.7, and Java SE Development Kit 8. If you encounter any problems when you debug the sample code, contact Tablestore technical support personnel.

  i. Configure system parameters such as the name of the instance, the name of the table, the endpoint of the instance, and AccessKey ID and AccessKey secret of the Alibaba Cloud account.

     You can also customize the method to load parameters.

  ii. Select **include dependencies with "provided" scope**. Click **OK**.

  iii. Run the code sample.



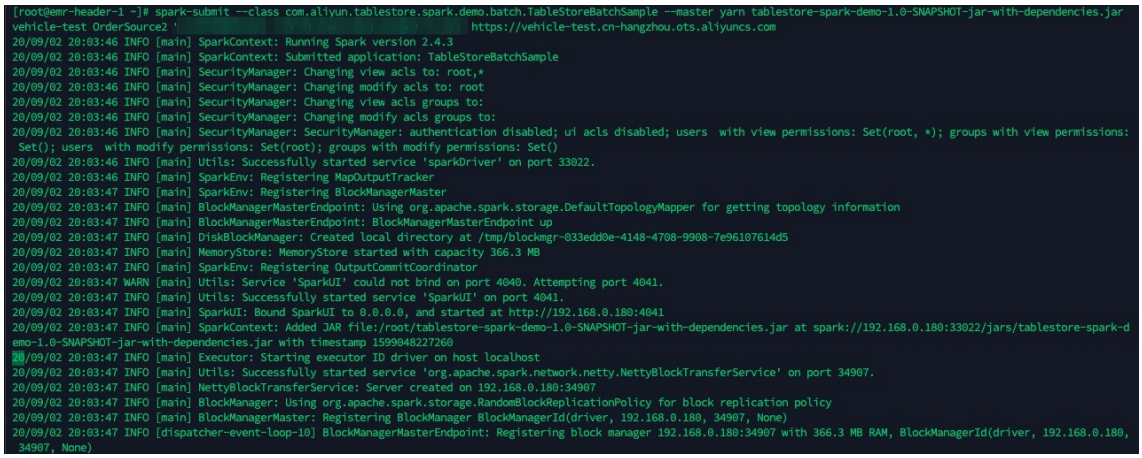
- Perform debugging on the Spark cluster

  The spark-submit method is used in the example. By default, master in the sample code is set to `local[*]`. You can remove this parameter when you run tasks on the Spark cluster. Then, import the spark-submit parameter.

  i. Run the **mvn-u clean package** command for packaging. The path of the package is `target/tablestore-spark-demo-1.0-SNAPSHOT-jar-with-dependencies.jar`.

  ii. Upload the package to the Driver node of the Spark cluster. Use spark-submit to submit a task.

```
spark-submit --class com.aliyun.tablestore.spark.demo.batch.TableStoreBatchSample --m
aster yarn tablestore-spark-demo-1.0-SNAPSHOT-jar-with-dependencies.jar <ots-instance
Name> <ots-tableName> <access-key-id> <access-key-secret> <ots-endpoint>
```



2.4.2. Stream computing

You can use Spark DataFrame to access Tablestore and perform debugging on the local computer and cluster.

Prerequisites

- You have understood the dependent packages that are used for Spark to access Tablestore. These dependent packages have been imported to the project by using maven.
  - Spark related dependent packages: spark-core, spark-sql, and spark-hive
  - Spark Tablestore connector: emr-tablestore-<version>.jar
  - Tablestore SDK for Java: tablestore-<version>-jar-with-dependencies.jar

  <version> indicates the version number of the corresponding dependent package. Use the actual version number.
- A Source table and a tunnel for the Source table are created at the Tablestore side. For more information, see Create a tunnel.
- An AccessKey pair that consists of an AccessKey ID and an AccessKey secret is obtained. For more information, see Obtain an AccessKey pair.

Quick start

You can learn about how to use stream computing by using project samples.

1. Download the source code of the project sample from GitHub. For more information about the path to download the code, visit TableStoreSparkDemo.

   The project contains the complete dependencies and samples on how to use batch computing. For more information about the dependencies, see the pom file in the project.
2. Read the README document of the TableStoreSparkDemo project. Install Spark Tablestore connector and Tablestore SDK for Java of the latest versions to the local maven library.
3. Modify the sample code.

   StructuredTableStoreAggSQLSample is used in the example. Descriptions of the core sample code:
   - format("tablestore") indicates that the ServiceLoader method is used to load Spark Tablestore connector. For more information about configurations, see META-INF.services in the project.
   - instanceName indicates the name of the Tablestore instance. tableName indicates the name of the table. endpoint indicates the endpoint of the instance. accessKeyId indicates the AccessKey ID of the Alibaba Cloud account. accessKeySecret indicates the AccessKey secret of the Alibaba Cloud account.
   - catalog is a JSON string, which includes field names and types. In the following example, the table has the following fields: UserId (type: STRING), OrderId (type: STRING), price (type: DOUBLE), and timestamp (type: LONG).
   - maxoffsetsperchannel indicates the maximum data volume read by each channel of micro-batch operations. Default value: 10000.

```
    val ordersDF = sparkSession.readStream
      .format("tablestore")
      .option("instance.name", instanceName)
      .option("table.name", tableName)
      .option("tunnel.id", tunnelId)
      .option("endpoint", endpoint)
      .option("access.key.id", accessKeyId)
      .option("access.key.secret", accessKeySecret)
      .option("maxoffsetsperchannel", maxOffsetsPerChannel) // The default value is 1
0000.
      .option("catalog", dataCatalog)
      .load()
      .createTempView("order_source_stream_view")

  val dataCatalog: String =
    s"""
       |{"columns": {
       |    "UserId": {"type":"string"},
       |    "OrderId": {"type":"string"},
       |    "price": {"type":"double"},
       |    "timestamp": {"type":"long"}
       | }
       |}""".stripMargin
```

Debugging

After the sample code is modified, you can perform debugging on the local computer or Spark cluster. StructuredTableStoreAggSQLSample is used in the example to describe the debugging process.

- Perform debugging on the local computer

  IntelliJ IDEA is used in the example.

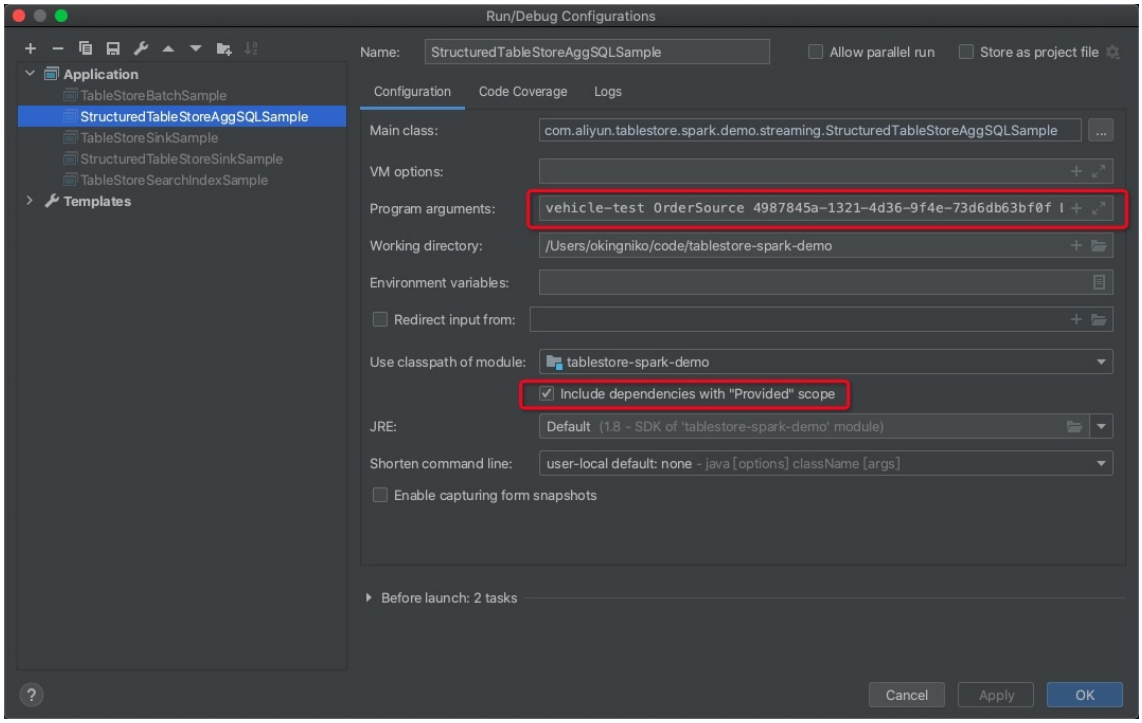
  > **Note**
  >
  > The environments used for testing are Spark 2.4.3, Scala 2.11.7, and Java SE Development Kit 8. If you encounter any problems when you debug the sample code, contact Tablestore technical support personnel.

  i. Configure system parameters such as the name of the instance, the name of the table, the endpoint of the instance, and AccessKey ID and AccessKey secret of the Alibaba Cloud account.

     You can also customize the method to load parameters.

  ii. Select **include dependencies with "provided" scope,**. Click **OK**.

  iii. Run the code sample.



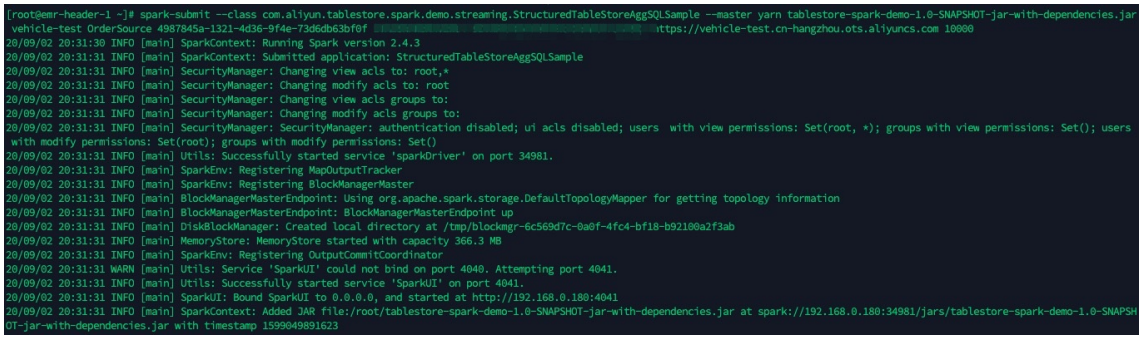
- Perform debugging on the Spark cluster

  The spark-submit method is used in the example. By default, master in the sample code is set to `local[*]`. You can remove this parameter when you run tasks on the Spark cluster. Then, import the spark-submit parameter.

  i. Run the **mvn -U clean package** command for packaging. The path of the package is `target/tablestore-spark-demo-1.0-SNAPSHOT-jar-with-dependencies.jar`.

  ii. Upload the package to the Driver node of the Spark cluster. Use spark-submit to submit a task.

```
spark-submit --class com.aliyun.tablestore.spark.demo.streaming.StructuredTableStoreAggSQLSample --master yarn tablestore-spark-demo-1.0-SNAPSHOT-jar-with-dependencies.jar <ots-instanceName> <ots-tableName> <ots-tunnelId> <access-key-id> <access-key-secret> <ots-endpoint> <max-offsets-per-channel>
```



2.5. Detail analysis

2.5.1. Configure predicate pushdown for batch computing

When you use search index for batch computing, you can customize predicate pushdown. You can configure predicate pushdown only on LONG and STRING columns.

Background information

You can use predicate pushdown when a query condition for the search index needs to filter large amounts of data, and it is time-consuming for Tablestore to join all intermediate query results. To address these needs, you can push down the value filtering of part of fields from the storage layer (Tablestore) to the compute layer (Spark), which improves query efficiency.

Take `select * from table where a = 10 and b < 999999999;` as an example. If the number of entries returned for the condition (a = 10) is only 1,000, and the number of entries returned for the condition (b < 999999999) is up to 100 million, it is time-consuming for Tablestore to join the results of the two conditions. However, you can push down the condition (b < 999999999) to the compute layer. Spark needs only to filter data based on the 1,000 entries returned from the storage layer, which reduces most pressure on the storage layer.

Supported operators

The following table lists the operators supported by Spark.

Spark	Supported	Example of SQL statement
And	Yes	select * from table where a > 1 and b < 0;
Or	Yes	select * from table where a > 1 or b < 0;
Not	Yes	select * from table where a != 1;
EqualTo	Yes	select * from table where a = 1;
Not+EqualTo	Yes	select * from table where a != 1;
IsNull	Yes	select * from table where a is null; Return rows that do not contain the a column from table.
In	Yes	select * from table where a in {1,2,3}; By default, the maximum upper limit is 1024.

Spark	Supported	Example of SQL statement
LessThan	Yes	select * from table where a < 10; You can customize predicate pushdown when the SQL statement uses LONG or STRING columns.
LessThanOrEqual	Yes	select * from table where a <=10; You can customize predicate pushdown when the SQL statement uses LONG or STRING columns.
GreaterThan	Yes	select * from table where a > 10; You can customize predicate pushdown when the SQL statement uses LONG or STRING columns.
GreaterThanOrEqual	Yes	select * from table where a >= 10; You can customize predicate pushdown when the SQL statement uses LONG or STRING columns.
StringStartsWith	Yes	select * from table where a like "tablestore%"; Return rows that contain a column whose values are prefixed with tablestore from table.
Coordinate pair of the central point, radius (STRING JSON)	Yes	select * from table where val_geo = '{"centerPoint":"3,0", "distanceInMeter": 100000}'; The geographical area is defined by the radius and coordinate pair of the central point.
Coordinate pairs of the upper-left corner and lower-right corner in the rectangular geographical area (STRING JSON)	Yes	select * from table where geo = '{"topLeft":"8,0", "bottomRight": "0,10"}'; The rectangular geographical area is defined by the coordinate pairs of the upper-left corner and lower-right corner.
Coordinate pairs of vertices in the polygon geographical area (STRING JSON)	Yes	select * from table where geo = '{"points":["5,0", "5,1", "6,1", "6,10"]}'; The polygon geographical area is defined by the coordinate pairs of multiple vertices.

Configure predicate pushdown

The parameters for predicate pushdown must be configured when you create a Spark external table. Predicate pushdown uses the following rules:

- When ADN and NOT are the only logical operators in the filter conditions, you can specify whether to enable predicate pushdown.
- When the filter conditions contain logical operator OR , all predicates are pushed down. Custom configurations for predicate pushdown such as push.down.range.long=false and push.down.range.string=false (configured by using SQL in EMR) become invalid.

The following table describes the combination of logical operators with pushdown configurations, examples of SQL statements, and expected results.

Logical operator	Pushdown configuration	Example of SQL statement	Expected result
AND	• push.down.range.long=true • push.down.range.string=true	select * from table where val_long1 > 1000 and val_long1 is null and name like 'table%' and pk in {12341,213432};	The SQL statement is executed. All predicates are pushed down.
AND only	• push.down.range.long=false • push.down.range.string=true	select * from table where val_long1 > 1 and name like 'table%';	Predicates where operators that are used to compare values with LONG column values are not pushed down. The results returned based on the predicate are filtered by the Spark compute layer. The Spark compute layer obtains the data based on the condition name like 'table%'. The condition val_long1 > 1 is executed by using the business code.
AND only	• push.down.range.long=true • push.down.range.string=false	select * from table where val_string1 > 'string1' and name like 'table%';	Predicates where operators that are used to compare values with STRING column values are not pushed down. The results returned based on the predicate are filtered by the Spark compute layer. The Spark compute layer obtains the data based on the condition name like 'table%'. The condition val_long1 > 1 is executed by using the business code.

Logical operator	Pushdown configuration	Example of SQL statement	Expected result
AND only	Columns containing values for geographical locations exist	select * from table where val_geo = '{"centerPoint":"3,0", "distanceInMeter": 100000}' and val_long1 = 37691900 and val_long2 > 2134234;	The SQL statement is executed. The configurations of push.down.range.long determine whether the Spark compute layer filters the results returned based on the condition val_long2 > 2134234.
OR	• push.down.range.long=true • push.down.range.string=true	select * from table where val_long1 > 1000 or val_long1 is null or name like 'table%' and pk in {12341,213432};	The SQL statement is executed. All predicates are pushed down.
OR	Columns containing values for geographical locations exist	select * from table where val_geo = '{"centerPoint":"3,0", "distanceInMeter": 100000}' or val_long1 = 37691900;	The SQL statement is executed. All predicates are pushed down.
OR	• push.down.range.long=false • push.down.range.string=true • The SQL statement contains the rangeLong field for filtering.	select * from table where val_long1 > 1 or name like 'table%';	The custom configurations for predicate pushdown become invalid. All predicates are pushed down.
OR	• push.down.range.long=true • push.down.range.string=false • The SQL statement contains the rangeString field for filtering.	select * from table where val_string1 > 'string1' or name like 'table%';	The custom configurations for predicate pushdown become invalid. All predicates are pushed down.

2.5.2. Implementation of stream computing

This topic describes how Spark Structured Streaming works in the micro-batch mode. This topic also describes how Tablestore interacts with Spark Structured Streaming.

Background information

Spark Data Source API V1 is used in the example to describe how Spark Structured Streaming works in the micro-batch mode.

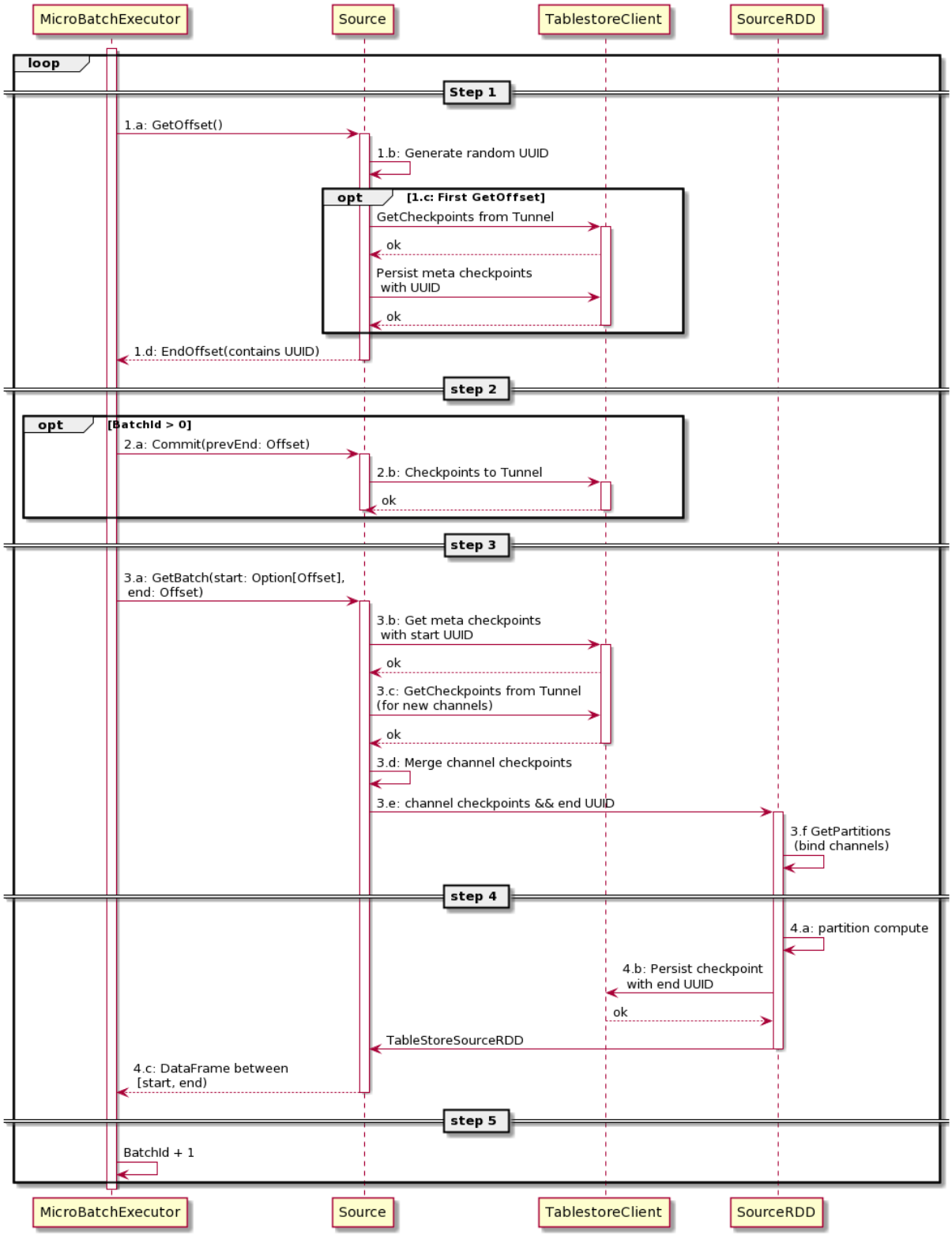
1. Call the GetOffset method to obtain the maximum offset (EndOffset) that can be read for the current batch.
2. Call the GetBatch method to obtain the data between the start offset (the EndOffset value of the last batch) and the maximum offset of the current batch. Then, convert the data.
3. Execute the custom computational logic of Spark.

In this case, the streaming operation of the upstream database must provide a flexible and precise Seek feature. Then, the streaming operation can obtain the start or end cursor of each partition in real time to estimate the offset for the micro-batch mode.

If the upstream database is a distributed NoSQL database, it is difficult to connect the database to the micro-batch operation of Spark Structured Streaming. This is because the database provides a Change Data Capture (CDC)-based streaming operation, which uses the continuous streaming mode and does not provide the Seek feature. If data is obtained in the GetOffset stage in advance, the desired EndOffset value can be obtained. However, additional Resilient Distributed Dataset (RDD)-based computing must be performed in advance and the obtained EndOffset value must be persisted to the cache. This degrades the performance of Source.

Interaction between Tablestore and Spark Structured Streaming

The following Unified Modeling Language (UML) sequence diagram shows how Tablestore interacts with Spark Structured Streaming.



In the figure, MicroBatchExecutor serves as the micro-batch framework of Spark. Source serves as the abstract operation class of Spark Structured Streaming. SourceRDD serves as the RDD abstract class of Spark. TablestoreClient serves as the client of Tablestore. Solid lines indicate operations. Dashed lines indicate success responses.

The overall process is implemented by the GetOffset, Commit, and GetBatch steps by using loops. Each loop corresponds to a batch operation of Spark Streaming.

Detailed procedure:

1. Call the GetOffset method to obtain the maximum offset (EndOffset) that can be read for the current batch.
   i. Call the GetOffset method of Source by using MicroBatchExecutor to obtain the maximum offset (EndOffset) that can be read for the current batch.
   ii. Generate a random UUID string in Source. Each UUID corresponds to an offset.
   iii. (Optional) Obtain the tunnel information from Tablestore.

   > **Note**
   >
   > This step is required only when the GetOffset method of Source is called for the first time. The system skips this step in the subsequent operations.

      a. Obtain the consumption checkpoints of all current channels from the Tablestore Tunnel Service server.
      b. Persist the checkpoints to the Meta table and establish the mappings between the UUID and checkpoints.

   iv. Encapsulate the UUID into an offset (you can convert between the offset and UUID). Then, the offset is used as the EndOffset value of the current batch and returned to MicroBatchExecutor.
2. (Optional) Persist the checkpoints to the Tunnel Service server.

   > **Note**
   >
   > This step is only performed when the ID of the current batch is greater than 0. In other cases, the system skips this step.

   i. Call the Commit logic of Source by using MicroBatchExecutor.
   ii. Persist the checkpoints that correspond to the EndOffset value of the last batch (start offset of the current batch) to the Tunnel Service server.

   > **Note**
   >
   > In normal cases, the Commit logic is not required to process additional content. The checkpoints are persisted to the Tunnel Service server for the following purposes:
   >
   > - Display the consumption progress in real time.
   > - Ensure that data is sequentially distributed based on the parent-child relationship. Child channels can only be loaded after the client returns the checkpoints when parent channels finished data consumption to the server.

3. Call the GetBatch method to obtain the data between the start offset (the EndOffset value of the last batch) and the maximum offset of the current batch. Then, convert the data.

i. Call the GetBatch method of Source by using MicroBatchExecutor based on the start offset of the current batch and the EndOffset value returned by using the GetOffset method. The GetBatch method is called to obtain the data of the current batch, which is used for the computational logic.

ii. Obtain the real-time checkpoints of the channels in tunnels from the Meta table based on the UUID to which the start offset corresponds.

iii. Obtain the checkpoints of channels from the Tunnel Service server on a regular basis.

New partitions such as subpartitions may be generated due to changes to partitions of a table. Therefore, the checkpoints of channels must be obtained regularly.

iv. Obtain the checkpoints of all channels by merging the real-time checkpoints obtained from the Meta table and the checkpoints that are regularly obtained from the Tunnel Service server.

v. Create SourceRDD based on information such as the latest checkpoints and the UUID to which the EndOffset value corresponds.

RDD is the fundamental data abstraction of Spark. An RDD is an immutable and partitionable collection of objects that can be used for parallel computing.

vi. Bind the channels of tunnels to RDD partitions in SourceRDD. This way, each channel converts and processes data in parallel on the Spark executor node.

4. Execute the custom computational logic of Spark.

i. Execute the computational logic on each RDD partition. Data is read from the channels and the checkpoints in the memory are updated for each channel.

ii. After each RDD partition completes the task for the current batch, such as reading the specified number of entries or detecting that no new data is found, persist the latest checkpoints to the row with the UUID that corresponds to the EndOffset value in the Meta table. After a batch is completed, the checkpoints that correspond to the start offset of the next batch can be found in the Meta table.

iii. After the computational logic is executed for all RDD partitions, return the data within the offset range of the current batch to MicroBatchExecutor.

5. Increase the batch ID after the computational logic is executed for all RDD partitions in the current batch. The preceding steps are performed from Step 1 in a continuous loop.

3.Hive/HadoopMR

3.1. Preparations

This topic describes how to make environment preparations to access a Tablestore table by using Hive and HadoopMR.

Use Hive and HadoopMR to access a Tablestore table

You can use Hive and HadoopMR to access and analyze data in Tablestore directly by using the dependency package released by the official teams of Tablestore and E-MapReduce.

Install JDK V7 or later

1. Download the installation package of JDK V7 or later. Install JDK V7 or later.
   - Linux or macOS: Use the package manager built in the system.
   - Windows: For more information about the download path, visit Java SE Development Kit 8 Downloads.
2. Follow the example to check the installation.

```
$ java -version
java version "1.8.0_77"
Java(TM) SE Runtime Environment (build 1.8.0_77-b03)
Java HotSpot(TM) 64-Bit Server VM (build 25.77-b03, mixed mode)
```

Install Hadoop and start the Hadoop environment

1. Download the Hadoop installation package whose version is later than 2.6.0. For more information, visit Index of /apache/hadoop/common.
2. Decompress the package and install Hadoop based on the cluster conditions.
3. Follow the example to start the Hadoop environment.

```
$ bin/start-all.sh
# Check whether the service is started.
$ jps
24017 NameNode
24835 Jps
24131 DataNode
24438 ResourceManager
5114 HMaster
24287 SecondaryNameNode
24527 NodeManager
```

4. Add the path of Hadoop in */etc/profile*. Run the **source /etc/profile** command to make the configurations take effect.

```
export HADOOP_HOME=/data/hadoop/hadoop-2.6.0
export PATH=$PATH:$HADOOP_HOME/bin
```

Download the Hive installation package and install Hive

1. Download the Hive installation package of the bin.tar.gz type. For more information about the download path, visit DOWNLOADS.
2. Follow the example to decompress the installation package.

```
$ mkdir /home/admin/hive-2.1.0
$ tar -zxvf apache-hive-2.1.0-bin.tar.gz -C /home/admin/
$ mv /home/admin/apache-hive-2.1.0-bin /home/admin/hive-2.1.0/
```

3. Follow the example to initialize schema.

```
    # Go to the specified directory.
    $ cd /home/admin/hive-2.1.0/
    # Start initialization. For mysql, replace derby with mysql.
    # If an error occurs in the execution process, delete rm -rf metastore_db/ before y
ou execute the code again.
    $ ./bin/schematool -initSchema -dbType derby
```

4. Follow the example to start the Hive environment.

```
$ ./bin/hive
# Check whether the service is started.
hive> show databases;
OK
default
Time taken: 0.207 seconds, Fetched: 1 row(s)
```

Download Tablestore SDK for Java

1. Download related dependent packages of Java SDK whose version is later than 4.1.0 from the Maven library. For more information about the download path, see Tablestore SDK for Java.

   Related dependent packages of Java SDK are released with the latest Java SDK. Download the latest related dependent packages.
2. Follow the example to copy the SDK to the directory of Hive.

```
$ mv tablestore-4.1.0-jar-with-dependencies.jar /home/admin/hive-2.1.0/
```

Download Alibaba Cloud EMR SDK

Download the dependent packages of EMR SDK. For more information about the specific download path, visit aliyun-emapreduce-datasources.

3.2. Tutorial

This topic describes how to use Hive or HadoopMR to access a Tablestore table.

Data preparation

Prepare a data table named pet in Tablestore. The name column is the only primary key column. The following table describes the data in the table.

> **Note** You do not need to write data to empty columns. Tablestore is schema-free. You do not need to write NULL even if no values exist.

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	
Claws	Gwen	cat	m	1994-03-17	
Buffy	Harold	dog	f	1989-05-13	
Fang	Benny	dog	m	1990-08-27	
Bowser	Diane	dog	m	1979-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	
Whistler	Gwen	bird		1997-12-09	
Slim	Benny	snake	m	1996-04-29	
Puffball	Diane	hamster	f	1999-03-30	

Access a Tablestore table by using Hive

1. Run the following commands to add HADOOP_HOME and HADOOP_CLASSPATH to */etc/profile*:

```
export HADOOP_HOME=${Your Hadoop installation directory}
export HADOOP_CLASSPATH=emr-tablestore-1.4.2.jar:tablestore-4.3.1-jar-with-dependencies
.jar:joda-time-2.9.4.jar
```

2. Run the **bin/hive** command to go to Hive and create an external table. Sample SQL statement:

```
CREATE EXTERNAL TABLE pet
  (name STRING, owner STRING, species STRING, sex STRING, birth STRING, death STRING)
  STORED BY 'com.aliyun.openservices.tablestore.hive.TableStoreStorageHandler'
  WITH SERDEPROPERTIES(
    "tablestore.columns.mapping"="name,owner,species,sex,birth,death")
  TBLPROPERTIES (
    "tablestore.endpoint"="YourEndpoint",
    "tablestore.access_key_id"="YourAccessKeyId",
    "tablestore.access_key_secret"="YourAccessKeySecret",
    "tablestore.table.name"="pet");
```

The following table describes the parameters that you can configure to create an external table.

Parameter	Description

Parameter	Description
WITH SERDEPROPERTIES	Field mapping configurations, including the configuration of tablestore.columns.mapping. By default, the names of fields in the external table are the same as the names of primary key or attribute columns in the Tablestore table. If the names of fields in the external table are different from the names of columns in the Tablestore table, you must specify tablestore.columns.mapping. The value of this parameter is a comma-separated string. No spaces are allowed at both ends of the comma (,). Each item specifies the name of a column in the Tablestore table. The order of the column names is the same as the order of the field names in the external table. **Note** The name of a column in Tablestore can contain whitespace characters. Therefore, whitespace characters are considered part of the name of a column in a Tablestore table.
TBLPROPERTIES	Property configurations of the external table, which include: ◦ tablestore.endpoint: required. This item specifies the endpoint that is used to access Tablestore. You can view the endpoint information about an instance in the Tablestore console. For more information about endpoints, see Endpoint. ◦ tablestore.instance: optional. This item specifies the name of the Tablestore instance. If you do not configure this item, the instance name is the first field of the value of tablestore.endpoint. For more information about instances, see Instance. ◦ tablestore.access_key_id: required. This item specifies the AccessKey ID of your Alibaba Cloud account or a RAM user. For more information, see Obtain an AccessKey pair. If you want to use temporary access credentials that are obtained from Security Token Service (STS) to access resources, set tablestore.access_key_id to the AccessKey ID in the temporary access credentials. ◦ tablestore.access_key_secret: required. This item specifies the AccessKey secret of your Alibaba Cloud account or a RAM user. For more information, see Obtain an AccessKey pair. If you want to use temporary access credentials that are obtained from STS to access resources, set tablestore.access_key_secret to the AccessKey secret in the temporary access credentials. ◦ tablestore.sts_token: optional. This item specifies the security token in the temporary access credentials. If you want to use temporary access credentials that are obtained from STS to access resources, you must configure this item. For more information, see Grant permissions to a temporary user. ◦ tablestore.table.name: required. This item specifies the name of the Tablestore table.

3. Query data in the external table.
   - Execute the **SELECT * FROM pet;** statement to query all rows in the table.

     Sample output:

     ```
     Bowser  Diane   dog     m       1979-08-31      1995-07-29
     Buffy   Harold  dog     f       1989-05-13      NULL
     Chirpy  Gwen    bird    f       1998-09-11      NULL
     Claws   Gwen    cat     m       1994-03-17      NULL
     Fang    Benny   dog     m       1990-08-27      NULL
     Fluffy  Harold  cat     f       1993-02-04      NULL
     Puffball        Diane   hamster f       1999-03-30      NULL
     Slim    Benny   snake   m       1996-04-29      NULL
     Whistler        Gwen    bird    NULL    1997-12-09      NULL
     Time taken: 5.045 seconds, Fetched 9 row(s)
     ```

   - Execute the **SELECT * FROM pet WHERE birth > "1995-01-01";** statement to query the rows in which the value of the birth column is greater than 1995-01-01.

     Sample output:

     ```
     Chirpy  Gwen    bird    f       1998-09-11      NULL
     Puffball        Diane   hamster f       1999-03-30      NULL
     Slim    Benny   snake   m       1996-04-29      NULL
     Whistler        Gwen    bird    NULL    1997-12-09      NULL
     Time taken: 1.41 seconds, Fetched 4 row(s)
     ```

   Sample output:

   ```
   Bowser  Diane   dog     m       1979-08-31      1995-07-29
   Buffy   Harold  dog     f       1989-05-13      NULL
   Chirpy  Gwen    bird    f       1998-09-11      NULL
   Claws   Gwen    cat     m       1994-03-17      NULL
   Fang    Benny   dog     m       1990-08-27      NULL
   Fluffy  Harold  cat     f       1993-02-04      NULL
   Puffball        Diane   hamster f       1999-03-30      NULL
   Slim    Benny   snake   m       1996-04-29      NULL
   Whistler        Gwen    bird    NULL    1997-12-09      NULL
   Time taken: 5.045 seconds, Fetched 9 row(s)
   hive> SELECT * FROM pet WHERE birth > "1995-01-01";
   Chirpy  Gwen    bird    f       1998-09-11      NULL
   Puffball        Diane   hamster f       1999-03-30      NULL
   Slim    Benny   snake   m       1996-04-29      NULL
   Whistler        Gwen    bird    NULL    1997-12-09      NULL
   Time taken: 1.41 seconds, Fetched 4 row(s)
   ```

Access a Tablestore table by using HadoopMR

The following sample code provides an example on how to use HadoopMR to collect statistics about the number of rows in the pet data table.

- Construct Mappers and Reducers

```
public class RowCounter {
public static class RowCounterMapper
extends Mapper<PrimaryKeyWritable, RowWritable, Text, LongWritable> {
    private final static Text agg = new Text("TOTAL");
    private final static LongWritable one = new LongWritable(1);
    @Override
    public void map(
        PrimaryKeyWritable key, RowWritable value, Context context)
        throws IOException, InterruptedException {
        context.write(agg, one);
    }
}
public static class IntSumReducer
extends Reducer<Text,LongWritable,Text,LongWritable> {
    @Override
    public void reduce(
        Text key, Iterable<LongWritable> values, Context context)
        throws IOException, InterruptedException {
        long sum = 0;
        for (LongWritable val : values) {
            sum += val.get();
        }
        context.write(key, new LongWritable(sum));
    }
}
}
```

The map() function of mapper is called each time the data source reads a row of data from Tablestore. The PrimaryKeyWritable parameter specifies the primary key of the row. The RowWritable parameter specifies the content of the row. You can call PrimaryKeyWritable.getPrimaryKey() to obtain the primary key object that is defined in Tablestore SDK for Java and RowWritable.getRow() to obtain the row object that is defined in the SDK.

- Specify Tablestore as the data source of mapper

```
private static RangeRowQueryCriteria fetchCriteria() {
    RangeRowQueryCriteria res = new RangeRowQueryCriteria("YourTableName");
    res.setMaxVersions(1);
    List<PrimaryKeyColumn> lower = new ArrayList<PrimaryKeyColumn>();
    List<PrimaryKeyColumn> upper = new ArrayList<PrimaryKeyColumn>();
    lower.add(new PrimaryKeyColumn("YourPkeyName", PrimaryKeyValue.INF_MIN));
    upper.add(new PrimaryKeyColumn("YourPkeyName", PrimaryKeyValue.INF_MAX));
    res.setInclusiveStartPrimaryKey(new PrimaryKey(lower));
    res.setExclusiveEndPrimaryKey(new PrimaryKey(upper));
    return res;
}
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "row count");
    job.addFileToClassPath(new Path("hadoop-connector.jar"));
    job.setJarByClass(RowCounter.class);
    job.setMapperClass(RowCounterMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(LongWritable.class);
    job.setInputFormatClass(TableStoreInputFormat.class);
    TableStoreInputFormat.setEndpoint(job, "https://YourInstance.Region.ots.aliyuncs.com/
");
    TableStoreInputFormat.setCredential(job, "YourAccessKeyId", "YourAccessKeySecret");
    TableStoreInputFormat.addCriteria(job, fetchCriteria());
    FileOutputFormat.setOutputPath(job, new Path("output"));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

In the preceding example, job.setInputFormatClass(TableStoreInputFormat.class) is used to specify Tablestore as the data source. You must also perform the following operations:

- Deploy hadoop-connector.jar to the cluster and add the path of hadoop-connector.jar to classpath. The path is the local path of hadoop-connector.jar. The path is specified in addFileToClassPath(). In the sample code, hadoop-connector.jar is in the current path.
- Specify the endpoint and AccessKey pair that are used to access Tablestore. Use TableStoreInputFormat.setEndpoint() to specify the endpoint and TableStoreInputFormat.setCredential() to specify the AccessKey pair.

- Specify a table to record the statistics about the number of rows in the Tablestore table.

> **Note**
>
> - A RangeRowQueryCriteria object that is defined in Tablestore SDK for Java is added to the data source each time addCriteria() is called. You can call addCriteria() multiple times. The limits on the RangeRowQueryCriteria object are the same as the limits on the RangeRowQueryCriteria object that is used by the GetRange operation of Tablestore SDK for Java.
> - You can use setFilter() and addColumnsToGet() of RangeRowQueryCriteria to filter out unnecessary rows and columns on the Tablestore server. This reduces the volume of accessed data and costs, and improves performance.
> - You can add multiple RangeRowQueryCriteria objects to multiple tables to perform a union query that combines the results of two or more independent tables.
> - You can add multiple RangeRowQueryCriteria objects to a single table to evenly split the range. TableStore-Hadoop Connector can split the range that is specified by users based on specific policies.

Run the program

1. Specify HADOOP_CLASSPATH.

```
HADOOP_CLASSPATH=hadoop-connector.jar bin/hadoop jar row-counter.jar
```

2. Run the **find output -type f** command to query all files in the output directory.

   Sample output:

```
output/_SUCCESS
output/part-r-00000
output/._SUCCESS.crc
output/.part-r-00000.crc
```

3. Run the **cat out/part-r-00000** command to collect statistics about the number of rows in the output/.part-r-00000 file.

```
TOTAL   9
```

Data type conversion

Tablestore supports data types that are partially identical to the data types that are supported by Hive or Spark.

The following table describes the conversion of the data types that are supported by Tablestore (rows) into the data types that are supported by Hive or Spark (columns).

Data type	TINYINT	SMALLINT	INT	BIGINT	FLOAT	DOUBLE	BOOLEAN	STRING	BINARY

Data type	TINYINT	SMALLINT	INT	BIGINT	FLOAT	DOUBLE	BOOLEAN	STRING	BINARY
INTEGER	Supported (with loss of precision)	Supported (with loss of precision)	Supported (with loss of precision)	Supported	Supported (with loss of precision)	Supported (with loss of precision)	Not supported	Not supported	Not supported
DOUBLE	Supported (with loss of precision)	Supported (with loss of precision)	Supported (with loss of precision)	Supported (with loss of precision)	Supported (with loss of precision)	Supported	Not supported	Not supported	Not supported
BOOLEAN	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Supported	Not supported	Not supported
STRING	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Supported	Not supported
BINARY	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Supported

4.Function Compute

4.1. Use Function Compute

This topic describes how to use Function Compute to perform real-time computing on the incremental data in Tablestore.

Context

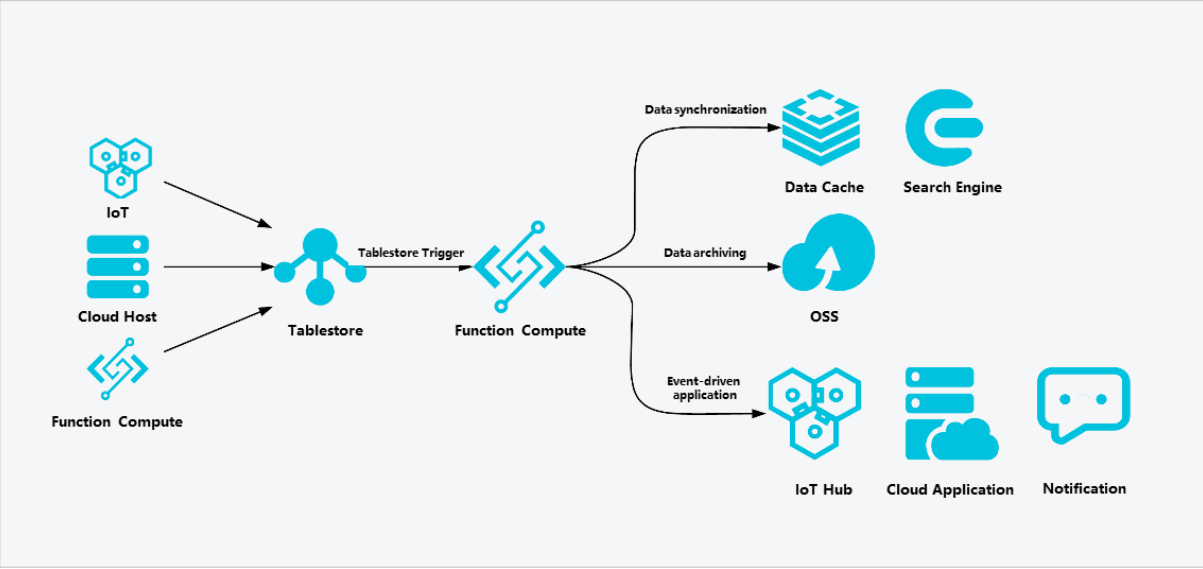
Alibaba Cloud Function Compute is an event-driven computing service that allows you to focus on writing and uploading code without the need to manage servers. Function Compute prepares computing resources and runs your code in an elastic manner. You are charged only for the resources that are consumed when the code is run. For more information, see What is Function Compute?.

A Tablestore Stream is a tunnel that is used to obtain incremental data from Tablestore tables. After you create Tablestore triggers, a Tablestore Stream and a function in Function Compute can be automatically connected. This allows the custom program logic in the function to automatically process modifications to data in Tablestore tables.

Scenarios

The following figure shows the tasks that you can perform by using Function Compute.

- Data synchronization: You can use Function Compute to synchronize real-time data that is stored in Tablestore to data cache, search engines, or other database instances.
- Data archiving: You can use Function Compute to archive incremental data that is stored in Tablestore to OSS for cold backup.
- Event-driven application: You can create triggers to trigger functions to call API operations that are provided by IoT Hub and cloud applications. You can also create triggers to send notifications.



Prerequisites

- A Tablestore instance and a data table are created. For more information, see Create an instance and Create a data table.
- Function Compute is activated. For more information, see Activate Function Compute.

Usage notes

Make sure that the Tablestore data table resides in the same region as the Function Compute service.

Step 1: Enable the Stream feature for the data table

Before you create a trigger, you must enable the Stream feature for the data table in the Tablestore console to allow the function to process incremental data that is written to the table.

1. Log on to the Tablestore console.
2. On the **Overview** page, click the name of the instance that you want to manage or click Manage Instance in the **Actions** column of the instance that you want to manage.
3. In the **Tables** section of the **Instance details** tab, click the name of the required data table and click the **Tunnels** tab. Alternatively, you can click the ⋮ icon and then click **Tunnels**.
4. On the **Tunnels** tab, click **Enabled** in the Stream Information section.
5. In the **Enable Stream** dialog box, configure the Log Expiration Time parameter and click **Enabled**.

   The value of the Log Expiration Time parameter must be a non-zero integer and cannot be changed after it is specified. Unit: hours. Maximum value: 168.

   > **Note** Specify a value for the Log Expiration Time parameter based on your business requirements.

Step 2: Configure a Tablestore trigger

You can create a Tablestore trigger in the Function Compute console to process the real-time data stream in a Tablestore data table.

1. Create a Function Compute service.
   i. Log on to the Function Compute console.
   ii. In the left-side navigation pane, click **Services & Functions**.
   iii. In the top navigation bar, select a region.
   iv. On the **Services** page, click **Create Service**.
   v. In the **Create Service** panel, configure the Name and Description parameters and configure the Logging and Tracing Analysis parameters based on your business requirements.

      For more information about the parameters, see Manage services.
   vi. Click **OK**.

      After you create the service, you can view the service and service configurations on the **Services** page.
2. Create a Function Compute function.

   > **Note** You can create a function from scratch, by using a container image, or by using a template. The following procedure describes how to create a function **from scratch**. For information about how to create a function by using other methods, see Use a container image to create a function and Use function templates to create functions.

   i. On the **Services** page, click the name of the service that you want to manage.

ii. In the left-side navigation pane, click **Functions** and then click **Create Function**.

iii. On the **Create Function** page, select **Start from Scratch**.

iv. In the **Basic Settings** section, configure the parameters. The following table describes the parameters.

Parameter	Required	Description	Example
Function Name	No	Enter a name for the function. The name can be up to 64 characters in length and can contain digits, letters, underscores (_), and hyphens (-). The function name is case-sensitive and must start with a letter. **Note** If you leave this parameter empty, Function Compute automatically creates a name for your function.	Function
Runtime Environments	Yes	Select a language, for example, Python, Java, PHP, or Node.js. For information about the runtime environments that are supported by Function Compute, see Manage functions.	Python 3.6
Request Type	Yes	If you want to use a Tablestore trigger, select **Event Requests**.	Event Requests
Instance Category	Yes	Select an instance category for the function. Valid values: ■ **Elastic Instance** ■ **Performance Instance** For more information, see Instance types and usage modes. For information about the billing of each instance category, see Billing.	Elastic Instance
Memory Capacity	Yes	Specify the size of the memory that is required to execute the function by using one of the following methods: ■ **Select a value**: Select a value from the drop-down list. ■ **Input a value**: Click **Enter Memory Size** and enter a value for the memory size. Valid values: ■ **Elastic Instance**: [128, 3072]. Unit: MB. ■ **Performance Instance**: [4, 32]. Unit: GB. **Note** The value must be a multiple of 64 MB.	512 MB

After you create the function, you can view the function on the **Functions** page.

v. In the **Configure Trigger** section, configure the parameters. The following table describes the parameters.

Parameter	Description	Example
Trigger Type	Select **Tablestore**.	
Name	Enter a name for the trigger.	Tablestore-trigger
Instance	Select a Tablestore instance from the drop-down list.	distribute-test
Table	Select a data table from the drop-down list.	source_data
Role Name	Select **AliyunTableStoreStreamNotificationRole**. **Note** After you configure the preceding parameters, click **OK**. The first time you create a trigger of this type, click **Authorize Now** in the message that appears, create the role, and assign permissions to the role as prompted.	AliyunTableStoreStreamNotificationRole

vi. Click **Create**.

The trigger that you created is displayed on the **Triggers** tab.

Note You can also view and create Tablestore triggers on the **Trigger** tab of the table in the Tablestore console.

Step 3: Test the function

Function Compute allows you to debug functions online. You can create an event to trigger a function and test whether the function logic is implemented as expected.

Tablestore events that trigger Function Compute are in the CBOR format, which is a JSON-like binary format. You can debug a function online by performing the following steps:

1. Write code.
   i. On the **Functions** page, click the name of the function that you want to manage.

ii. On the function details page, click the **Code** tab to write code in the code editor.

> **Note** In this example, the function code is written in Python.

In the code, `records = json.loads(event)` is used to process custom trigger events for the test.

```
import logging
import cbor
import json
 def get_attribute_value(record, column):
     attrs = record[u'Columns']
     for x in attrs:
         if x[u'ColumnName'] == column:
             return x['Value']
 def get_pk_value(record, column):
     attrs = record[u'PrimaryKey']
     for x in attrs:
         if x['ColumnName'] == column:
             return x['Value']
 def handler(event, context):
     logger = logging.getLogger()
     logger.info("Begin to handle event")
     #records = cbor.loads(event)
     records = json.loads(event)
     for record in records['Records']:
         logger.info("Handle record: %s", record)
         pk_0 = get_pk_value(record, "pk_0")
         attr_0 = get_attribute_value(record, "attr_0")
     return 'OK'
```

2. Test the function.

i. On the **Code** tab, click the icon next to **Test Function** and select **Configure Test Parameters** from the drop-down list.

ii. In the **Configure Test Parameters** panel, click the **Create New Test Event** or **Modify Existing Test Event** tab, select **Tablestore** for **Event Template**, and then enter the event name and event content. Click **OK**.

For more information about the format of event content, see Appendix: Data processing.

iii. Click **Test Function** to check whether the function performs as expected.

After you test the function by using `records=json.loads(event)`, you can change this line of code to `records = cbor.loads(event)` and click **Deployment Code**. This way, the specific function logic is triggered when data is written to the Tablestore data table.

Appendix: Data processing

When you configure test parameters, you must specify the event content based on a specific data format.

- Data format

A Tablestore trigger encodes incremental data in the CBOR format to create a Function Compute event. The following code provides an example of the format of incremental data:

```
{
    "Version": "string",
    "Records": [
        {
            "Type": "string",
            "Info": {
                "Timestamp": int64
            },
            "PrimaryKey": [
                {
                    "ColumnName": "string",
                    "Value": formated_value
                }
            ],
            "Columns": [
                {
                    "Type": "string",
                    "ColumnName": "string",
                    "Value": formated_value,
                    "Timestamp": int64
                }
            ]
        }
    ]
}
```

- Parameters

  The following table describes the parameters that are included in the preceding format.

Parameter	Description
Version	The version of the payload, which is Sync-v1. Data type: string.
Records	The array that stores the row of incremental data in the data table. This parameter includes the following fields: ○ Type: the type of the operation that is performed on the row. Valid values: PutRow, UpdateRow, and DeleteRow. Data type: string. ○ Info: includes the Timestamp field, which specifies the time when the row was last modified. The value of Timestamp must be in UTC. Data type: int64.
PrimaryKey	The array that stores the primary key column. This parameter includes the following fields: ○ ColumnName: the name of the primary key column. Data type: string. ○ Value: the content of the primary key column. Data type: formated_value. Valid values: integer, string, and blob.

Parameter	Description
Columns	The array that stores the attribute columns. This parameter includes the following fields: ◦ Type: the type of the operation that is performed on the attribute column. Valid values: Put, DeleteOneVersion, and DeleteAllVersions. Data type: string. ◦ ColumnName: the name of the attribute column. Data type: string. ◦ Value: the content of the attribute column. Data type: formated_value. Valid values: integer, boolean, double, string, and blob. ◦ Timestamp: the time when the attribute column was last modified. The value of Timestamp must be in UTC. Data type: int64.

- Data example

```
{
    "Version": "Sync-v1",
    "Records": [
        {
            "Type": "PutRow",
            "Info": {
                "Timestamp": 1506416585740836
            },
            "PrimaryKey": [
                {
                    "ColumnName": "pk_0",
                    "Value": 1506416585881590900
                },
                {
                    "ColumnName": "pk_1",
                    "Value": "2017-09-26 17:03:05.8815909 +0800 CST"
                },
                {
                    "ColumnName": "pk_2",
                    "Value": 1506416585741000
                }
            ],
            "Columns": [
                {
                    "Type": "Put",
                    "ColumnName": "attr_0",
                    "Value": "hello_table_store",
                    "Timestamp": 1506416585741
                },
                {
                    "Type": "Put",
                    "ColumnName": "attr_1",
                    "Value": 1506416585881590900,
                    "Timestamp": 1506416585741
                }
            ]
        }
    ]
}
```

Appendix: Create a trigger by using an existing Function Compute service and function

You can use an existing Function Compute function and service to create a Tablestore trigger in the Tablestore console.

1. Log on to the Tablestore console.
2. On the **Overview** page, click the name of the instance that you want to manage or click Manage Instance in the **Actions** column of the instance that you want to manage.
3. In the **Tables** section of the **Instance Details** tab, click the name of the required data table.
4. On the **Trigger** tab, click **Use Existing Function Compute**.
5. In the **Create Trigger** dialog box, select a Function Compute service for FC Service and a Function

Compute function for FC Function, enter a trigger name, and then click **OK**.

4.2. Use Function Compute to cleanse data

Tablestore provides highly concurrent write performance and low storage cost and is suitable for storing IoT data, logs, and monitoring data. When you write data to a Tablestore data table, you can cleanse the data by using Function Compute and write the cleansed data to another data table in Tablestore. You can access raw data or cleansed data in Tablestore in real time.

Sample scenarios

You want to write log data that includes three fields to Tablestore. To efficiently query the logs, you must write the logs in which the value of the level field is greater than 1 to another data table named result. The following table describes the fields that are included in the logs.

Field	Type	Description
id	Integer	The ID of the log.
level	Integer	The level of the log. A larger value indicates a higher level.
message	String	The content of the log.

Step 1: Enable the Stream feature for the data table

Before you create a trigger, you must enable the Stream feature for the data table in the Tablestore console to allow the function to process incremental data that is written to the table.

1. Log on to the Tablestore console.
2. On the **Overview** page, click the name of the instance that you want to manage or click Manage Instance in the **Actions** column of the instance that you want to manage.
3. In the **Tables** section of the **Instance details** tab, click the name of the required data table and click the **Tunnels** tab. Alternatively, you can click the ⋮ icon and then click **Tunnels**.
4. On the **Tunnels** tab, click **Enabled** in the Stream Information section.
5. In the **Enable Stream** dialog box, configure the Log Expiration Time parameter and click **Enabled**.

   The value of the Log Expiration Time parameter must be a non-zero integer and cannot be changed after it is specified. Unit: hours. Maximum value: 168.

   > **Note** Specify a value for the Log Expiration Time parameter based on your business requirements.

Step 2: Configure a Tablestore trigger

You can create a Tablestore trigger in the Function Compute console to process the real-time data stream in a Tablestore data table.

1. Create a Function Compute service.
   i. Log on to the Function Compute console.
   ii. In the left-side navigation pane, click **Services & Functions**.
   iii. In the top navigation bar, select a region.
   iv. On the **Services** page, click **Create Service**.
   v. In the **Create Service** panel, configure the Name and Description parameters and configure the Logging and Tracing Analysis parameters based on your business requirements.

      For more information about the parameters, see Manage services.
   vi. Click **OK**.

      After you create the service, you can view the service and service configurations on the **Services** page.
2. Create a Function Compute function.

   > **Note** You can create a function from scratch, by using a container image, or by using a template. The following procedure describes how to create a function **from scratch**. For information about how to create a function by using other methods, see Use a container image to create a function and Use function templates to create functions.

   i. On the **Services** page, click the name of the service that you want to manage.
   ii. In the left-side navigation pane, click **Functions** and then click **Create Function**.
   iii. On the **Create Function** page, select **Start from Scratch**.
   iv. In the **Basic Settings** section, configure the parameters. The following table describes the parameters.

Parameter	Required	Description	Example
Function Name	No	Enter a name for the function. The name can be up to 64 characters in length and can contain digits, letters, underscores (_), and hyphens (-). The function name is case-sensitive and must start with a letter. **Note** If you leave this parameter empty, Function Compute automatically creates a name for your function.	Function
Runtime Environments	Yes	Select a language, for example, Python, Java, PHP, or Node.js. For information about the runtime environments that are supported by Function Compute, see Manage functions.	Python 3.6
Request Type	Yes	If you want to use a Tablestore trigger, select **Event Requests**.	Event Requests

Parameter	Required	Description	Example
Instance Category	Yes	Select an instance category for the function. Valid values: ■ **Elastic Instance** ■ **Performance Instance** For more information, see Instance types and usage modes. For information about the billing of each instance category, see Billing.	Elastic Instance
Memory Capacity	Yes	Specify the size of the memory that is required to execute the function by using one of the following methods: ■ **Select a value**: Select a value from the drop-down list. ■ **Input a value**: Click **Enter Memory Size** and enter a value for the memory size. Valid values: ■ **Elastic Instance**: [128, 3072]. Unit: MB. ■ **Performance Instance**: [4, 32]. Unit: GB. **Note** The value must be a multiple of 64 MB.	512 MB

After you create the function, you can view the function on the **Functions** page.

v. In the **Configure Trigger** section, configure the parameters. The following table describes the parameters.

Parameter	Description	Example
Trigger Type	Select **Tablestore**.	
Name	Enter a name for the trigger.	Tablestore-trigger
Instance	Select a Tablestore instance from the drop-down list.	distribute-test
Table	Select a data table from the drop-down list.	source_data
Role Name	Select **AliyunTableStoreStreamNotificationRole**. **Note** After you configure the preceding parameters, click **OK**. The first time you create a trigger of this type, click **Authorize Now** in the message that appears, create the role, and assign permissions to the role as prompted.	AliyunTableStoreStreamNotificationRole

vi. Click **Create**.

The trigger that you created is displayed on the **Triggers** tab.

Note You can also view and create Tablestore triggers on the **Trigger** tab of the table in the Tablestore console.

Step 3: Verify data cleansing

After you create a trigger, you can write data to Tablestore and query the data to verify whether the data is cleansed as expected.

1. Write code.
   i. On the **Functions** page, click the name of the required function.
   ii. On the function details page, click the **Code** tab to write code in the code editor.

   In this example, the function code is written in Python. Set the following parameters to actual values: INSTANCE_NAME, REGION, and ENDPOINT.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import cbor
import json
import tablestore as ots
INSTANCE_NAME = 'distribute-test'
REGION = 'cn-shanghai'
ENDPOINT = 'http://%s.%s.vpc.tablestore.aliyuncs.com'%(INSTANCE_NAME, REGION)
RESULT_TABLENAME = 'result'
def _utf8(input):
    return str(bytearray(input, "utf-8"))
def get_attrbute_value(record, column):
    attrs = record[u'Columns']
    for x in attrs:
        if x[u'ColumnName'] == column:
            return x['Value']
def get_pk_value(record, column):
    attrs = record[u'PrimaryKey']
    for x in attrs:
        if x['ColumnName'] == column:
            return x['Value']
# The obtained credentials can be used to access Tablestore because the AliyunOTSFu
llAccess policy is attached to the role.
def get_ots_client(context):
    creds = context.credentials
    client = ots.OTSClient(ENDPOINT, creds.accessKeyId, creds.accessKeySecret, INST
ANCE_NAME, sts_token = creds.securityToken)
    return client
def save_to_ots(client, record):
    id = int(get_pk_value(record, 'id'))
    level = int(get_attrbute_value(record, 'level'))
    msg = get_attrbute_value(record, 'message')
    pk = [(_utf8('id'), id),]
    attr = [(_utf8('level'), level), (_utf8('message'), _utf8(msg)),]
    row = ots.Row(pk, attr)
    client.put_row(RESULT_TABLENAME, row)
def handler(event, context):
    records = cbor.loads(event)
    #records = json.loads(event)
    client = get_ots_client(context)
    for record in records['Records']:
        level = int(get_attrbute_value(record, 'level'))
        if level > 1:
            save_to_ots(client, record)
        else:
            print "Level <= 1, ignore."
```

2. Write data to the data table named source_data. Enter the values of the id, level, and message fields and query the cleansed data in the table named result.
   - When you write a log in which the value of the level field is greater than 1 to the source_data table, the log is synchronized to the result table.
   - When you write a log in which the value of the level field is less than or equal to 1 to the source_data table, the log is not synchronized to the result table.

5.DataV

5.1. Preparations

Before you use DataV to display Tablestore data, you must activate Tablestore and create instances and data tables. You must also activate DataV and create projects.

Background information

DataV can transform statistic data into a variety of dynamic visualization charts. After you add Tablestore data sources to DataV, DataV can generate a data dashboard based on the table data to display the data in real time.

Activate Tablestore

When you use Tablestore for the first time, take the following steps to activate Tablestore and create instances and data tables:

1. Activate Tablestore.
2. Create instances.
3. Create tables.
4. Read and write data in the console.

Activate DataV

When you use DataV for the first time, take the following steps to activate DataV and create projects:

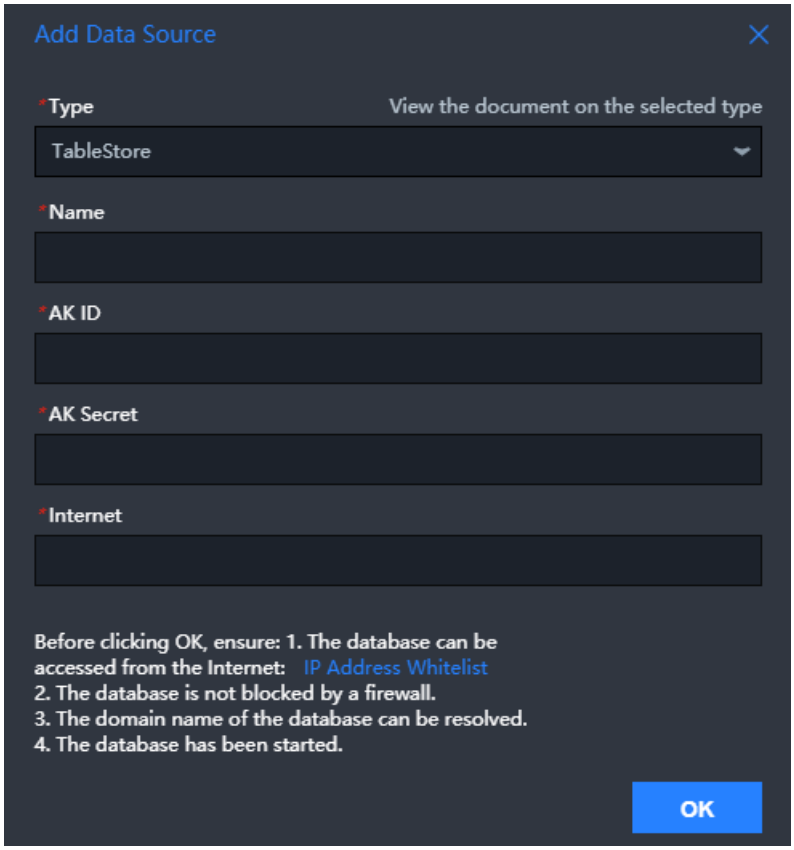
1. Activate DataV.
2. Create a project from a template or on a blank canvas. For more information, see Create a project (template) or Create a project (blank canvas).

5.2. Tutorial

After you add a Tablestore data source in the DataV console, you can use DataV to display Tablestore data.

Add a Tablestore data source

1. Log on to the DataV console.
2. On the **Data Sources** tab, click **Data Sources** in the left-side navigation pane.
3. On the **Data Sources** page, click **Add Source**.



4. In the **Add Data Source** dialog box, select **Tablestore** from the Type drop-down list. Then, configure other parameters for a Tablestore data source. The following table describes the parameters you can configure for a Tablestore data source.

Parameter	Description
Name	The display name of the data source.
AK ID	The AccessKey ID of the account that has permissions to access the Tablestore instance whose data you want to use as a data source.
AK Secret	The AccessKey secret of the account that has permissions to access the Tablestore instance whose data you want to use as a data source.
Internet	The endpoint of the Tablestore instance. Set this parameter based on the Tablestore instance you want to access.

5. Click **OK**.

   The added data source is displayed in the data source list.

Use DataV to display Tablestore data

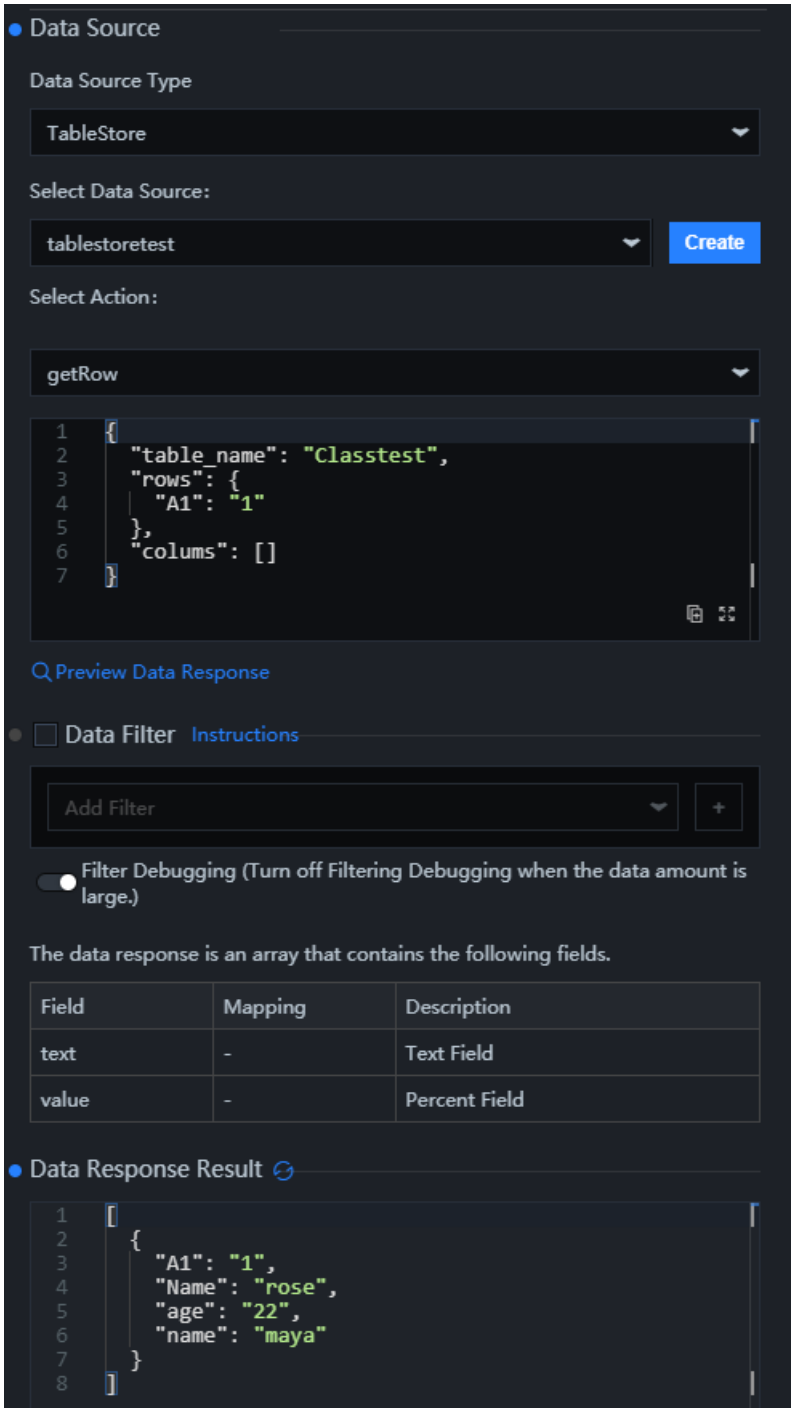
1. Log on to the DataV console.
2. On the **Projects** tab, move the pointer over the project that you want to edit and click **Edit**.
3. On the Canvas Editor page, click a widget.

> **Note** If no widgets have been added to the canvas, add a widget. For more information, see Add a widget.

4. Configure the Tablestore data source.

   i. In the right-side component configurations panel of the canvas, click the [−] icon.

   ii. On the **Data** tab, click **Set**.



   iii. In the **Data Source** panel, set Data Source Type to **TableStore** and select an existing data source.

iv. Select an operation to perform on the data source and enter a query statement.

Tablestore supports getRow and getRange. getRow corresponds to GetRow in Tablestore. getRange corresponds to GetRange in Tablestore.

- If you select getRow, a row whose primary key is specified is read. The following code and table describe the format you can use and the parameters you can configure for a query statement.

```
{
"table_name": "test",
"rows": {
"id": 2
},
"columns": [
"id",
"test"
]
}
```

Parameter	Description
table_name	The name of the table in Tablestore.
rows	The primary keys of the rows. **Notice** The number and data types of primary key columns specified for each row must be the same as those of primary key columns in the table.
columns	The names of the columns to read. You can specify the names of primary key columns and the names of attribute columns. If you do not specify this parameter, all data in each row is returned.

- If you select getRange, all data whose primary keys are within a specified range is read. The following code and table describe the format you can use and the parameters you can configure for a query statement.

```
{
"table_name": "test",
"direction": "FORWARD",
"columns": [
"id",
"test"
],
"range": {
"limit": 4,
"start": {
"id": "InfMin"
},
"end": {
"id": 3
}
}
}
```

Parameter	Description
table_name	The name of the table in Tablestore.
direction	The direction in which to read data. ■ If this parameter is set to FORWARD, the value of the start primary key must be smaller than that of the end primary key. The returned rows are sorted in ascending order based on their primary key values. ■ If the value is BACKWARD, the value of the start primary key must be greater than that of the end primary key, and the returned rows are sorted in descending order based on their primary key values. Example: A table contains two primary keys A and B where the value of A is smaller than that of B. If you set direction to FORWARD, the rows whose primary key values are equal to or larger than the value of A and smaller than the value of B are returned in ascending order from A to B. If you set direction to BACKWARD, the rows whose primary key values are equal to or larger than the value of B and smaller than the value of A are returned in descending order from B to A.
columns	The names of the columns to read. You can specify the names of primary key columns and the names of attribute columns. If you do not specify this parameter, all data in each row is returned. If a row is within the specified range of primary key values to read but does not contain the specified columns to return, the row is not included in the response.

Parameter	Description
limit	The maximum number of rows to return. The value of this parameter must be greater than 0. An operation stops when the maximum number of rows are returned in a forward or backward direction, even if a part of rows within the specified range are not returned.
start end	The start and end primary keys of the range to read. The start and end primary keys must be valid primary keys or virtual points consisting of the InfMin and InfMax type data. The number of columns for each virtual point must be the same as that of each primary key. InfMin indicates an infinitely small value. All values of other types are greater than InfMin. InfMax indicates an infinitely great value. All values of other types are smaller than InfMax. **Notice** The number and data types of primary key columns specified for each row must be the same as those of primary key columns in the table. ■ start indicates the start primary key. If the row that contains the start primary key exists, the response includes this row. ■ end indicates the end primary key. No matter whether the row that contains the end primary key exists, the response does not include this row.

v. Click **Data Response Result** next to the icon to obtain a response.

Note After you obtain the response, you can click **Preview Data Response** to view the response.

5. Preview and publish the project

i. In the upper-right corner of Canvas Editor, click the icon to preview the project.

ii. In the upper-right corner of Canvas Editor, click the icon.

iii. In the **Publish** dialog box, click **Publish Project**.

iv. Click the icon on the right side of the URL in the **Project UPL** section.

v. Copy and paste the URL to the browser address bar to view a published project.