ALIBABA CLOUD

Alibaba Cloud

E-MapReduce
Developer Guide

Document Version: 20210909

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.Preparations

1.1. Development preparations

This topic describes the preparations required for E-MapReduce (EMR) development.

You must make the following preparations:

- Activate the Alibaba Cloud service and create an AccessKey pair. Activate Alibaba Cloud Object Storage Service (OSS). You can visit the OSS product page to activate the service.
- Have a basic understanding of Spark, Hadoop, Hive, and Pig.

  The development practices of Spark, Hadoop, Hive, and Pig are not described in EMR documentation.
- Have a basic understanding of the development components of EMR.

1.2. Configure the OSS URI to use E-MapReduce

This topic describes how to configure the OSS URI to use E-MapReduce.

OSS URI

When using E-MapReduce, you can use two types of OSS URIs:

- native URI: *oss://[accessKeyId:accessKeySecret@]bucket[.endpoint]/object/path*

  This URI is used for specifying input and output data sources for a job, which is similar to hdfs://. When you operate OSS data, you can configure the accessKey ID, accessKey Secret, and the endpoint. Alternatively, you can specify the accessKey ID, accessKey Secret, and the endpoint in the URI.
- ref URI: ossref://bucket/object/path

  It is only valid in the configuration of an E-MapReduce job and is used to specify the resources needed for running the job.

We call prefixes, such as oss and ossref, as schemes. Pay special attention to the scheme difference in URI.

> **Notice**
>
> Currently, operations only support OSS for standard storage types.
>
> - E-MapReduce uses the multipart mode to upload large files to OSS. Note that if your job is interrupted, some of the result data remains in OSS. You must remove it manually. The procedure here is the same with the use of HDFS. One difference, however, is that E-MapReduce uses the multipart mode to upload large files. The file fragments are uploaded to the OSS fragment management. Therefore, you must delete the remaining job files in OSS file management and clean the file fragments in OSS fragment management. Otherwise, you are charged for data storage.
> - Except for the preceding manual cleanup, you can also configure the lifecycle of the fragments, so that the expired fragments are automatically cleaned. For more information, see Lifecycle management of OSS files.

1.3. Use the sample project

This sample project is a complete, compilable, and executable project. It includes the sample code of MapReduce, Pig, Hive, and Spark.

Sample project

The sample project includes the following jobs:

- MapReduce

  WordCount: counts how often a word occurs in a file.

- Hive

  sample.hive: queries data from tables.

- Pig

  sample.pig: processes Object Storage Service (OSS) objects.

- Spark
  - SparkPi: calculates Pi.
  - SparkWordCount: counts how often a word occurs in a file.
  - LinearRegression: trains a linear regression model and extracts model summary statistics.
  - OSSSample: uses Spark with OSS.
  - MaxComputeSample: uses Spark with MaxCompute.
  - MNSSample: uses Spark with Message Service (MNS).
  - LoghubSample: uses Spark with LogHub.

Dependencies

- Test data in the data directory
  - The_Sorrows_of_Young_Werther.txt: the input data file for MapReduce WordCount or SparkWordCount.
  - patterns.txt: the word patterns to be ignored in the MapReduce WordCount job.
  - u.data: the test table for sample.hive.

- abalone: the test data file for linear regression.

- JAR package in the lib directory

  tutorial.jar: the JAR package required for sample.pig.

Preparation

This project provides some test data. You can directly upload the test data to OSS for use. You can also prepare test data in services such as MaxCompute, MNS, Message Queue (MQ), and Log Service.

- For more information about how to use Log Service, see 5-minute quick start.
- For more information about how to create MaxCompute projects and tables, see Create a project and Create and view a table.
- For more information about how to use MQ, see Quick Start of MQ.
- For more information about how to use MNS, see the **"Overview"** section in **Quick-start** of MNS.

Concepts

- OSS URI: specifies the input or output data file for a job. It is similar to URLs like *hdfs://*. The OSS URI is in the *oss://accessKeyId:accessKeySecret@bucket.endpoint/a/b/c.txt* format.
- AccessKey ID and AccessKey secret: the AccessKey for you to access Alibaba Cloud API. Click Security Management to obtain the AccessKey.

Run jobs in your E-MapReduce cluster

- Spark
  - SparkWordCount:

    ```
    spark-submit --class SparkWordCount examples-1.0-SNAPSHOT-shaded.jar <inputPath>
        <outputPath> <numPartition>
    ```

    The following table describes parameters for submitting the SparkWordCount job.

Parameter	Description
inputPath	The path of the input data file.
outputPath	The path of the output data file.
numPartition	The number of Resilient Distributed Dataset (RDD) partitions of the input data file.

  - SparkPi:

    ```
    spark-submit --class SparkPi examples-1.0-SNAPSHOT-shaded.jar
    ```

- OSSSample:

```
spark-submit --class OSSSample examples-1.0-SNAPSHOT-shaded.jar <inputPath>
    <numPartition>
```

The following table describes parameters for submitting the OSSSample job.

Parameter	Description
inputPath	The path of the input data file.
numPartition	The number of RDD partitions of the input data file.

- ONSSample:

```
spark-submit --class ONSSample examples-1.0-SNAPSHOT-shaded.jar <accessKeyId>
    <accessKeySecret> <consumerId> <topic> <subExpression> <parallelism>
```

The following table describes parameters for submitting the ONSSample job.

Parameter	Description
accessKeyId	Your AccessKey ID.
accessKeySecret	Your AccessKey secret.
consumerId	The IDs of consumers. For more information, see Terms.
topic	The topic of message queues. Each message queue has a topic.
subExpression	The tag for filtering messages. For more information, see Message filtering.
parallelism	The number of consumers that consume messages in the queue.

- MaxComputeSample:

```
spark-submit --class MaxComputeSample examples-1.0-SNAPSHOT-shaded.jar <accessKeyId>
    <accessKeySecret> <envType> <project> <numPartitions>
```

The following table describes parameters for submitting the MaxComputeSample job.

Parameter	Description
accessKeyId	Your AccessKey ID.
accessKeySecret	Your AccessKey secret.
envType	The environment type. A value of 0 indicates the public network, and 1 indicates the internal network. If the sample is run on a local server, set this parameter to 0. If the program is run in E-MapReduce, set this parameter to 1.
project	The name of the project. For more information, see Terms.
numPartition	The number of RDD partitions of the input data file.

- MNSSample:

```
spark-submit --class MNSSample examples-1.0-SNAPSHOT-shaded.jar <queueName>
    <accessKeyId> <accessKeySecret> <endpoint>
```

The following table describes parameters for submitting the MNSSample job.

Parameter	Description
queueName	The name of the queue.
accessKeyId	Your AccessKey ID.
accessKeySecret	Your AccessKey secret.
endpoint	The endpoint used to access the queue.

- LoghubSample:

```
spark-submit --class LoghubSample examples-1.0-SNAPSHOT-shaded.jar <sls project> <sls
    logstore> <loghub group name> <sls endpoint> <access key id> <access key secret> <batch
    interval seconds>
```

The following table describes parameters for submitting the LoghubSample job.

Parameter	Description
sls project:	The name of the project.
sls logstore	The name of the Logstore.
loghub group name	The name of the group that consumes Logstore data in the job. You can specify a name as needed. When the values of the sls project and sls store parameters are the same, jobs in the same group collaboratively consume data in the Logstore, and jobs in different groups separately consume data in the Logstore.
sls endpoint	The endpoint used to access Log Service. For more information, see Service endpoint.
accessKeyId	Your AccessKey ID.
accessKeySecret	Your AccessKey secret.

- LinearRegression:

```
spark-submit --class LinearRegression examples-1.0-SNAPSHOT-shaded.jar <inputPath>
    <numPartitions>
```

The following table describes parameters for submitting the LinearRegression job.

Parameter	Description
inputPath	The path of the input data file.
numPartition	The number of RDD partitions of the input data file.

- MapReduce

WordCount:

```
hadoop jar examples-1.0-SNAPSHOT-shaded.jar WordCount
    -Dwordcount.case.sensitive=true <inputPath> <outputPath> -skip <patternPath>
```

The following table describes parameters for submitting the MapReduce WordCount job.

Parameter	Description
inputPath	The path of the input data file.

Parameter	Description
outputPath	The path of the output data file.
patternPath	The path of the file that contains the word patterns to be ignored. You can use the *patterns.txt* file in the data directory.

- Hive

  ```
  hive -f sample.hive -hiveconf inputPath=<inputPath>
  ```

  The following table describes parameters for submitting the sample.hive job.

  inputPath: the path of the input data file.

- Pig

  ```
  pig -x mapreduce -f sample.pig -param tutorial=<tutorialJarPath> -param
      input=<inputPath> -param result=<resultPath>
  ```

  The following table describes parameters for submitting the sample.pig job.

Parameter	Description
tutorialJarPath	The JAR package. You can use the tutorial.jar package in the lib directory.
inputPath	The path of the input data file.
resultPath	The path of the output data file.

Notice

- To run jobs in E-MapReduce, you can upload the test data and the JAR package to OSS. Make sure that you set the storage path by following the OSS URI format.
- Alternatively, you can store the test data and the JAR package in the E-MapReduce cluster.

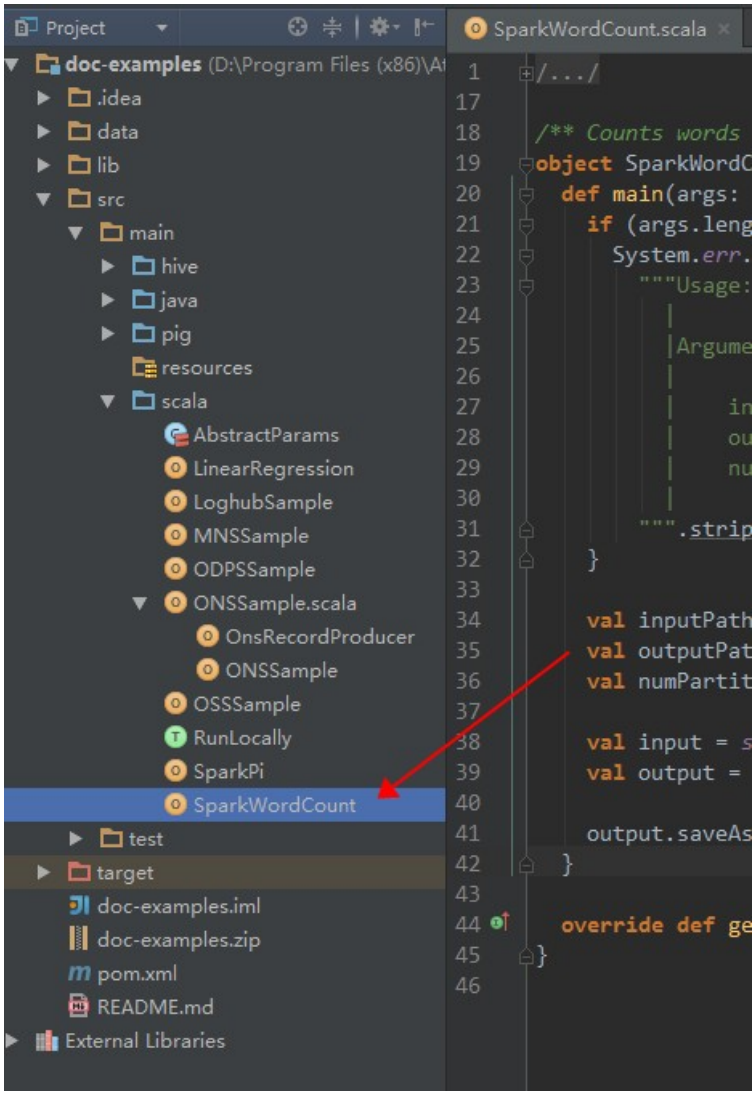
Run jobs on a local server

This section describes how to run Spark jobs on a local server to access data stored in Alibaba Cloud services, such as OSS. If you want to debug and run the jobs on a local server, we recommend that you use some development tools, such as IntelliJ IDEA or Eclipse, especially when you use Windows. Otherwise, you need to configure Hadoop and Spark runtime environments on Windows servers.

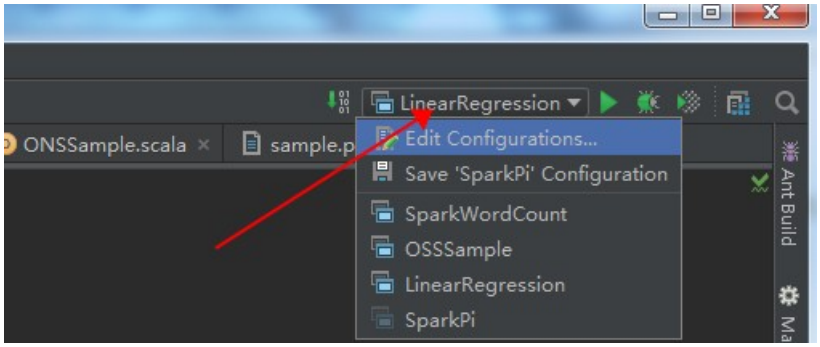
- IntelliJ IDEA
  - Preparation

    Install IntelliJ IDEA, Maven, Maven plugin for IntelliJ IDEA, Scala, and Scala plugin for IntelliJ IDEA.
  - Procedure

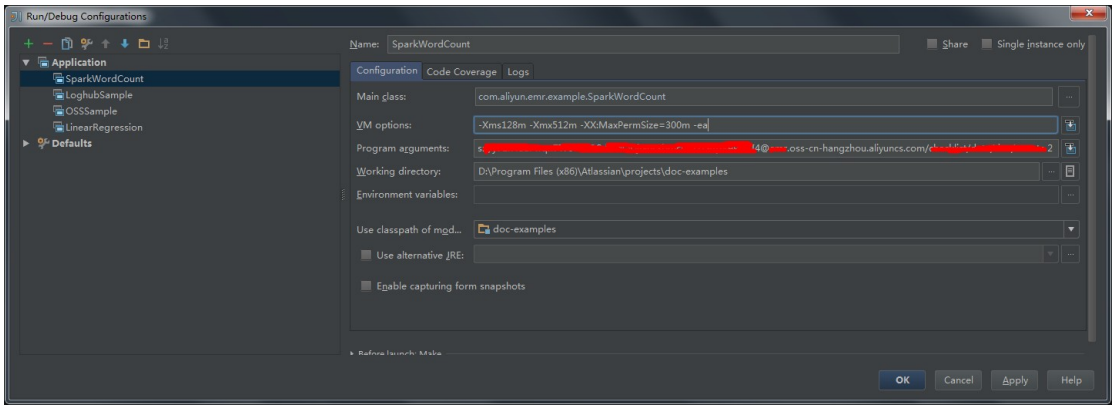
a. In IntelliJ IDEA, find and double-click SparkWordCount.scala in the left-side project list to open it.



b. Go to the Run/Debug Configurations page for SparkWordCount.scala.

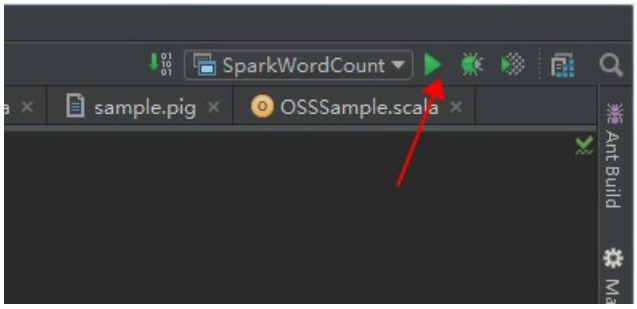


c. Click SparkWordCount in the application list and set required parameters on the right.

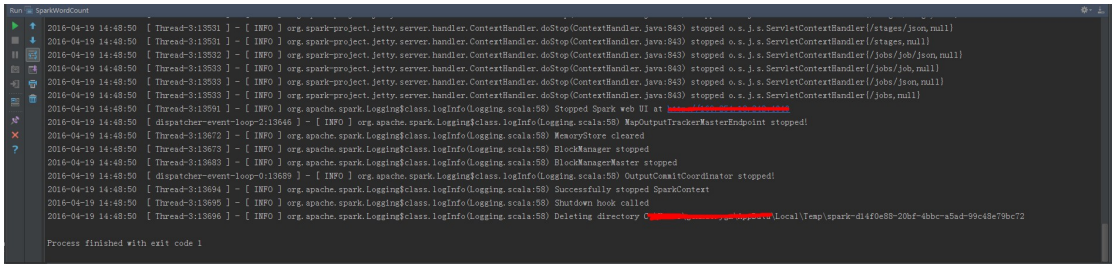


d. Click **OK**.

e. Click the **Run** icon to run SparkWordCount.



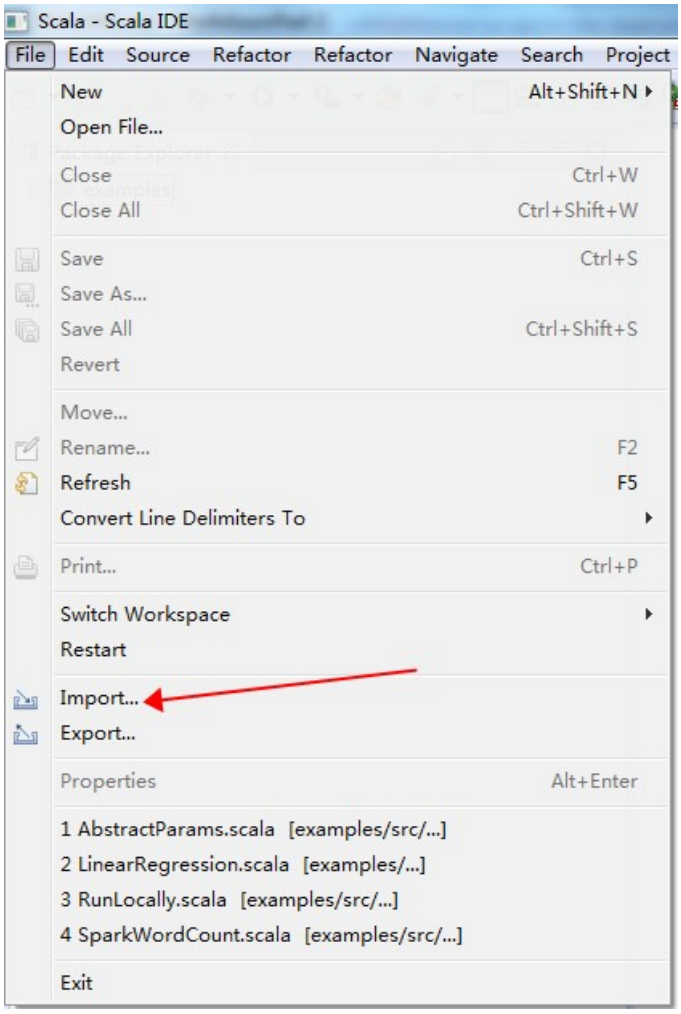
f. View operational logs.

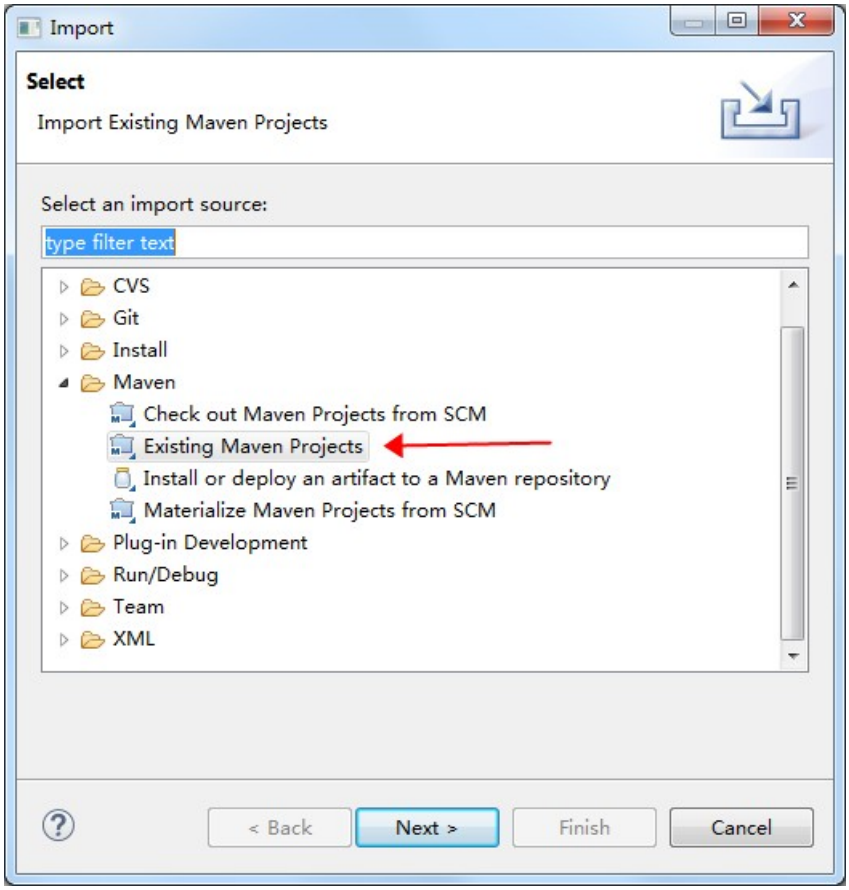


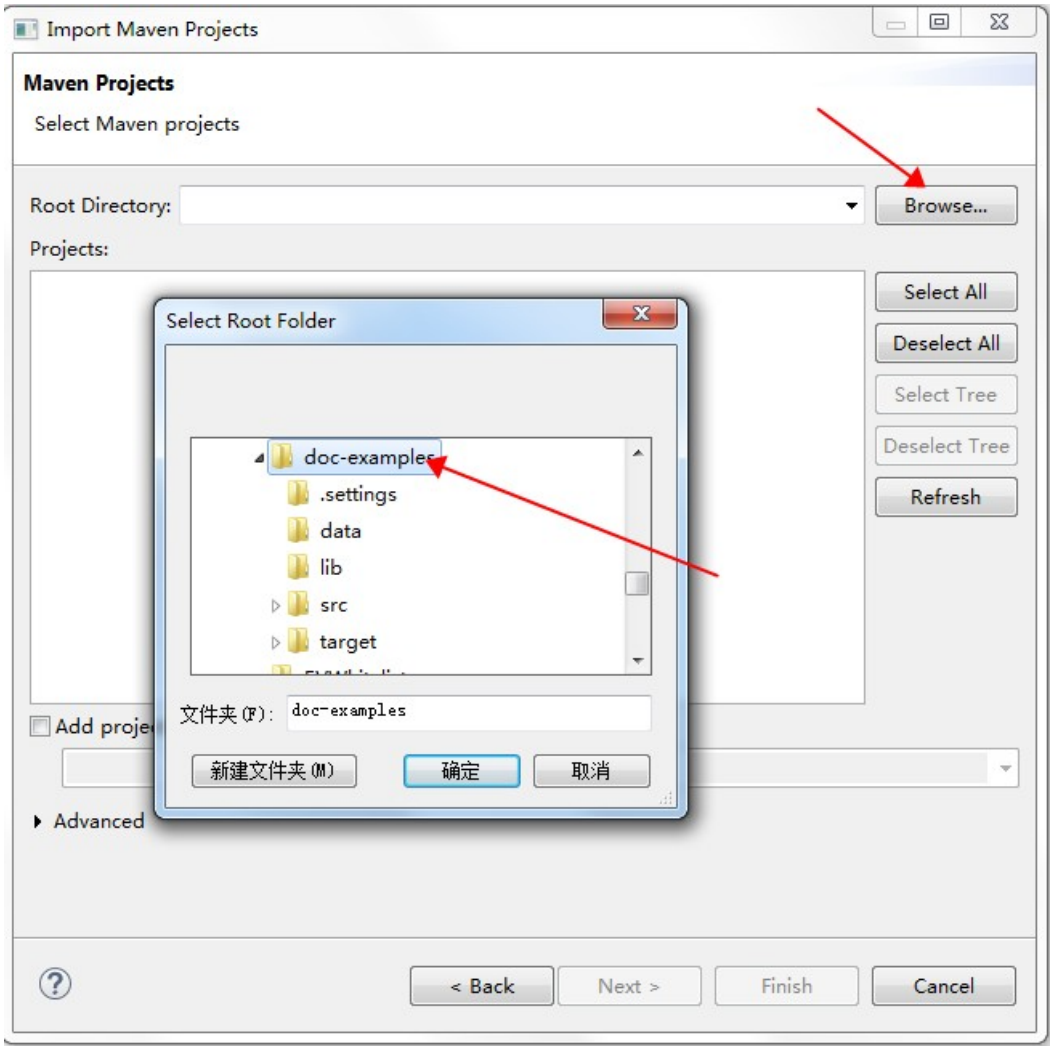
- Scala IDE for Eclipse
  - Preparation

    Install the Scala IDE for Eclipse, Maven, and Maven plugin for Eclipse.
  - Procedure

    a. Import a Maven project as shown in the following figures.


Scala - Scala IDE
File Edit Source Refactor Refactor Navigate Search Project
New Alt+Shift+N
Open File...
Close Ctrl+W
Close All Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Switch Workspace
Restart
Import...
Export...
Properties Alt+Enter
1 AbstractParams.scala [examples/src/...]
2 LinearRegression.scala [examples/...]
3 RunLocally.scala [examples/src/...]
4 SparkWordCount.scala [examples/src/...]
Exit

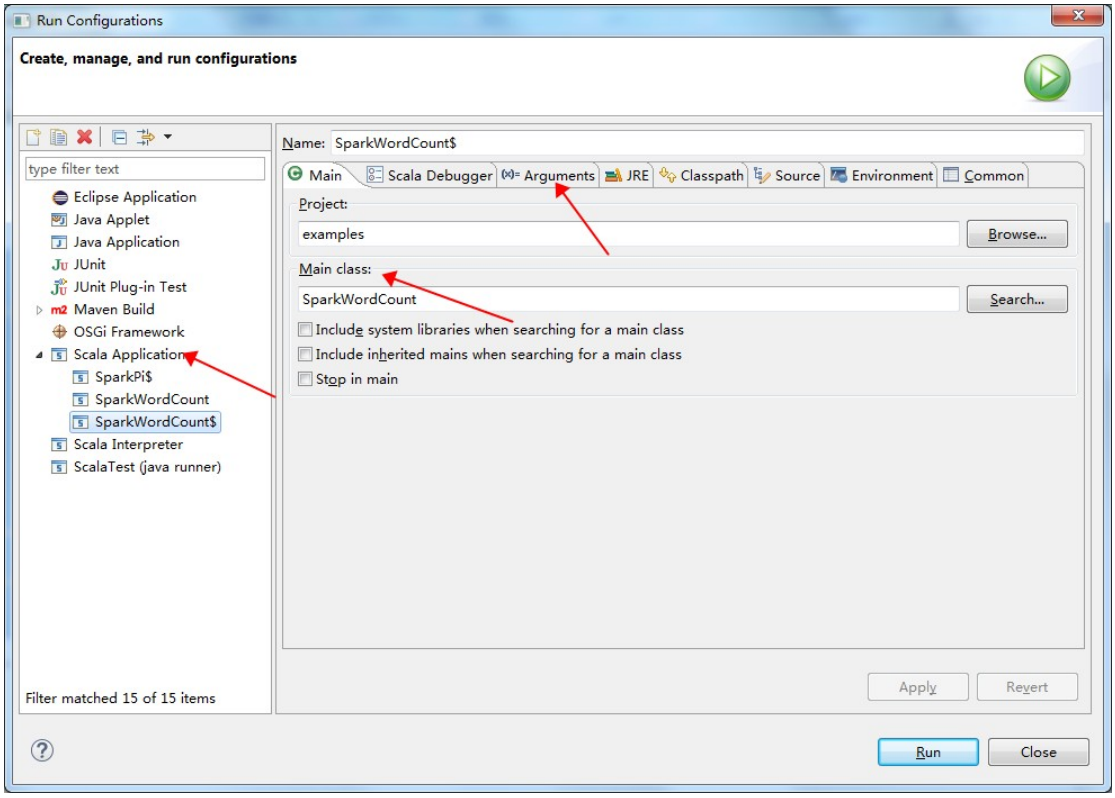

Import
Select
Import Existing Maven Projects
Select an import source:
type filter text
CVS
Git
Install
Maven
Check out Maven Projects from SCM
Existing Maven Projects
Install or deploy an artifact to a Maven repository
Materialize Maven Projects from SCM
Plug-in Development
Run/Debug
Team
XML
< Back
Next >
Finish
Cancel



b. Build the Maven project. You can press Alt + Shift + X and then M to build the Maven project. Alternatively, you can right-click the project name and choose **Run As > Maven build**.

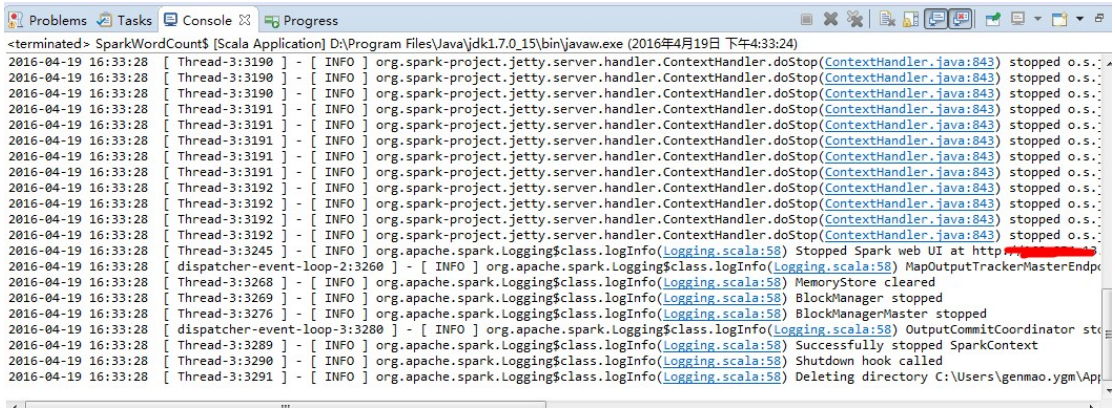
c. After the project is built, right-click the project and select **Run Configuration** to go to the configuration page.

d. On the configuration page, choose **Scala Application** > SparkWordCount$ and set the main class and required arguments on the right, as shown in the following figure.



e. Click **Run**.

f. View the output logs on the Console tab, as shown in the following figure.



1.4. Install Python

Python 3.6.4 is used in E-MapReduce (EMR).

Install Python 3.6.4

- By default, Python 3.6.4 is installed in EMR V2.10.0 and later 2.X.X versions, and V3.10.0 and later 3.X.X versions.

  The installation directory is */usr/bin/python3.6*.

- In EMR V2.X.X earlier than V2.10.0 and V3.X.X earlier than 3.10.0, you must perform the following steps to download and install Python 3.6.4:
  i. Download the software package Python-3.6.4.tgz.
  ii. Use the file transfer tool SSH Secure File Transfer Client to upload the JAR package to the */usr/local* directory of the master node of your cluster.
  iii. Decompress the package and install the software.
    a. Log on to the master node of your cluster. For more information, see Log on to a cluster.
    b. Create an installation directory for Python 3.6.4.

    ```
    mkdir -p /usr/local/python3
    ```

    c. Decompress the downloaded package.

    ```
    tar zxvf Python-3.6.4.tgz
    ```

    d. Access the folder where the installation file resides and specify the installation path.

    ```
    cd Python-3.6.4
    ./configure --prefix=/usr/local/python3
    ```

    e. Compile and install the software.

    ```
    make && make install
    ```

    f. Create a symbolic link to Python 3.6.4.

    ```
    ln -s /usr/local/python3/bin/python3 /usr/bin/python3
    ```

  iv. Check whether Python 3.6.4 is installed.

  ```
  python3 -V
  ```

  If the following information is returned, Python 3.6.4 is installed:

  ```
  Python 3.6.4
  ```

  v. Check whether pip3 is installed.

  ```
  pip3 -V
  ```

  If the following information is returned, pip3 is installed:

  ```
  pip 9.0.1 from /usr/local/Python-3.6.4/lib/python3.6/site-packages (python 3.6)
  ```

2.Spark

2.1. Preparations

This topic describes how to prepare for Spark development.

Install EMR SDK

You can use one of the following methods to install EMR SDK:

- Use the JAR package in Eclipse. Follow these steps:
  i. Download the dependencies required by EMR from the Maven repository.
  ii. Copy the required JAR package to your project directory. Select an SDK version based on the Spark version for the cluster in which your jobs run. Version 2.10 is recommended for Spark 1.X, and version 2.11 is recommended for Spark 2.X.
  iii. Right-click the project name in Eclipse and select **Properties**. In the dialog box that appears, click **Java Build Path** and then click **Add JARs**.
  iv. Select the JAR package you downloaded and click OK.

  After you complete the preceding steps, you can perform read/write operations on Object Storage Service (OSS), Log Service, Message Service (MNS), Message Queue (MQ), Tablestore, and MaxCompute data.
- Create a Maven project and add the following dependencies:

```
<dependency>
  <groupId>com.aliyun.emr</groupId>
  <artifactId>emr-tablestore</artifactId>
  <version>1.9.0</version>
</dependency>
<dependency>
  <groupId>com.aliyun.emr</groupId>
  <artifactId>emr-mns_2.11</artifactId>
  <version>1.9.0</version>
</dependency>
<dependency>
  <groupId>com.aliyun.emr</groupId>
  <artifactId>emr-logservice_2.11</artifactId>
  <version>1.9.0</version>
</dependency>
<dependency>
  <groupId>com.aliyun.emr</groupId>
  <artifactId>emr-maxcompute_2.11</artifactId>
  <version>1.9.0</version>
</dependency>
<dependency>
  <groupId>com.aliyun.emr</groupId>
  <artifactId>emr-ons_2.11</artifactId>
  <version>1.9.0</version>
</dependency>
<dependency>
  <groupId>com.aliyun.emr</groupId>
  <artifactId>emr-datahub_2.11</artifactId>
  <version>1.9.0</version>
</dependency>
```

Debug Spark code locally

When you debug the Spark code that is used to read or write OSS data, you must configure a SparkConf object. Set `spark.hadoop.mapreduce.job.run-local` to `true` and retain the default values of other parameters. Example:

```
val conf = new SparkConf().setAppName(getAppName).setMaster("local[4]")
  conf.set("spark.hadoop.fs.oss.impl", "com.aliyun.emr.fs.oss.JindoOssFileSystem")
  conf.set("spark.hadoop.mapreduce.job.run-local", "true")
  val sc = new SparkContext(conf)
  val data = sc.textFile("oss://...")
  println(s"count: ${data.count()}")
```

Usage notes

- Configure third-party dependencies

  To allow EMR to access Alibaba Cloud data sources such as OSS and MaxCompute, you need to configure the required third-party dependencies for your Spark job.

  For more information about how to add or remove third-party dependencies, see the pom file.

- Specify an OSS output directory

Add the fs.oss.buffer.dirs parameter to the local Hadoop configuration file. Set this parameter to a local directory. If you do not specify this parameter, a null pointer exception occurs when you run a Spark job to write OSS data.

- Clean up junk data

  When a Spark job fails, the system does not automatically delete the data generated by the job. You need to check the OSS output directory and clean up the generated files if any. Then, log on to the OSS console, click the target bucket, and clean up the parts if any on the Parts tab.

- Use PySpark

  For information about how to create a PySpark job, see aliyun-emapreduce-sdk.

2.2. Parameter description

The description of the parameters in Spark code

The following parameters can be configured in Spark code:

Property	Default	Description
spark.hadoop.fs.oss.accessKeyId	None	(Optional) The AccessKey ID required for accessing OSS.
spark.hadoop.fs.oss.accessKeySecret	None	(Optional) The AccessKey Secret required for accessing OSS.
spark.hadoop.fs.oss.securityToken	None	(Optional) The STS token required for accessing OSS.
spark.hadoop.fs.oss.endpoint	None	(Optional) The endpoint used for accessing OSS.
spark.hadoop.fs.oss.multipart.thread.number	5	The number of threads (concurrency) used by OSS for the upload part - copy operation.
spark.hadoop.fs.oss.copy.simple.max.byte	134217728	The file size limit for copying files between buckets in OSS using common APIs.
spark.hadoop.fs.oss.multipart.split.max.byte	67108864	The file multipart limit for copying files between buckets in OSS using common APIs.
spark.hadoop.fs.oss.multipart.split.number	5	The number of multipart files for copying files between buckets in OSS using common APIs. By default, the number is the same as the number of threads (concurrency) used.
spark.hadoop.fs.oss.impl	com.aliyun.fs.oss.nat.NativeOssFileSystem	The implementation class for the native file system of OSS.

Property	Default	Description
spark.hadoop.fs.oss.buffer.dirs	/mnt/disk1,/mnt/disk2,...	The OSS local temporary directory. It uses a data disk in the cluster by default.
spark.hadoop.fs.oss.buffer.dirs.exists	false	Whether the OSS temporary directory exists.
spark.hadoop.fs.oss.client.connection.timeout	50000	Timeout period for OSS client connections (unit: milliseconds).
spark.hadoop.fs.oss.client.socket.timeout	50000	Timeout period for OSS Client sockets (unit: milliseconds).
spark.hadoop.fs.oss.client.connection.ttl	-1	The value of time-to-live for clients
spark.hadoop.fs.oss.connection.max	1024	The maximum connections allowed.
spark.hadoop.job.runlocal	false	If the data source is OSS and you need to run and debug Spark code locally, set this parameter to true. Otherwise, set this parameter to false.
spark.logservice.fetch.interval.millis	200	The interval for receivers to retrieve data from LogHub.
spark.logservice.fetch.inOrder	true	Whether to consume the Shard data in order after the data has been parted.
spark.logservice.heartbeat.interval.millis	30000	The heartbeat interval for the data consumption processes (unit: milliseconds).
spark.mns.batchMsg.size	16	The number of MNS messages to fetch in bulk, with a maximum of 16.
spark.mns.pollingWait.seconds	30	The polling wait time if the MNS queue is empty.
spark.hadoop.io.compression.codec.snappy.native	false	Whether the Snappy files are in the standard Snappy format. By default, Hadoop recognizes Snappy files edited in Hadoop.

Smart Shuffle optimized configurations

Smart Shuffle is a shuffle implementation provided by Spark SQL in EMR version 3.16.0. It processes queries that have a large amount of shuffle data to improve the execution efficiency of Spark SQL. After Smart Shuffle is started, shuffle output data is not stored on ephemeral disks. Instead, data is transmitted to a remote node through a network in advance. Data in the same partition is transmitted to the same node and stored in the same file. Currently, Smart Shuffle has the following limitations:

- Smart Shuffle cannot guarantee the atomictiy of task execution. When the execution of a task fails, you need to restart all the Spark jobs.
- Smart Shuffle currently does not provide the External Smart Shuffle implementation, nor support Dynamic Resource Allocation.
- Speculative tasks are not supported.
- Smart Shuffle is incompatible with Adaptive Execution.

Smart Shuffle related configurations are as follows. You can enable Smart Shuffle by using Spark configuration files or the spark-submit parameters.

Parameter	Default	Description
spark.shuffle.manager	sort	A shuffle method that allows you to enable the Smart Shuffle feature with Spark Session by configurating spark.shuffle.manager=org.apache.spark.shuffle.sort.SmartShuffleManager or spark.shuffle.manager=smart.
spark.shuffle.smart.spill.memorySizeForceSpillThreshold	128m	When Smart Shuffle is enabled, the memory used for each shuffle task has a threshold. When the threshold is reached, shuffle data is sent to the corresponding remote node through the network according to partitions.
spark.shuffle.smart.transfer.blockSize	1m	The size of the cache used for network transfers when Smart Shuffle is enabled.

2.3. Use Spark to access OSS

This topic describes how to use Spark to read data from Object Storage Service (OSS).

Background information

E-MapReduce (EMR) provides the following methods for you to access OSS:

- You can access OSS by using MetaService.
- You can access OSS without using an AccessKey pair.
- You can access OSS by explicitly writing an AccessKey pair and an endpoint.

> **Note** You must use an internal endpoint of OSS. For more information about endpoints, see OSS endpoints.

Example of using Spark to access OSS

The following example shows how to use Spark to read data from OSS and write the processed data back to OSS without using an AccessKey pair:

```
val conf = new SparkConf().setAppName("Test OSS")
val sc = new SparkContext(conf)
val pathIn = "oss://bucket/path/to/read"
val inputData = sc.textFile(pathIn)
val cnt = inputData.count
println(s"count: $cnt")
val outputPath = "oss://bucket/path/to/write"
val outpuData = inputData.map(e => s"$e has been processed.")
outpuData.saveAsTextFile(outputPath)
```

For the complete sample code, visit GitHub.

Example of using PySpark to access OSS

The following example shows how to use PySpark to read data from OSS and write the processed data back to OSS without using an AccessKey pair:

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("Python Spark SQL OSS example").getOrCreate()
pathIn = "oss://bucket/path/to/read"
df = spark.read.text(pathIn)
cnt = df.count()
print(cnt)
outputPath = "oss://bucket/path/to/write"
df.write.format("parquet").mode('overwrite').save(outputPath)
```

2.4. Use Spark to access MaxCompute

This topic describes how to use Spark to read data from and write data to MaxCompute.

Procedure

1. Initialize an OdpsOps object.

   In Spark, the OdpsOps class is used to perform operations related to data in MaxCompute.

```
import com.aliyun.odps.TableSchema
  import com.aliyun.odps.data.Record
  import org.apache.spark.aliyun.odps.OdpsOps
  import org.apache.spark.{SparkContext, SparkConf}
  object Sample {
   def main(args: Array[String]): Unit = {
    // == Step-1 ==
    val accessKeyId = "<accessKeyId>"
    val accessKeySecret = "<accessKeySecret>"
    // Use internal URLs as examples.
    val urls = Seq("http://odps-ext.aliyun-inc.com/api", "http://dt-ext.odps.aliyun-inc.com")
    val conf = new SparkConf().setAppName("Test Odps")
    val sc = new SparkContext(conf)
    val odpsOps = OdpsOps(sc, accessKeyId, accessKeySecret, urls(0), urls(1))
    // Code for invocation:
    // == Step-2 ==
    ...
    // == Step-3 ==
    ...
   }
   // == Step-2 ==
   // Method definition 1
   // == Step-3 ==
   // Method definition 2
  }
```

2. Load table data from MaxCompute to Spark.

   Use the readTable method of the OdpsOps object to load table data from MaxCompute to Spark.

   ```
   // == Step-2 ==
       val project = <odps-project>
       val table = <odps-table>
       val numPartitions = 2
       val inputData = odpsOps.readTable(project, table, read, numPartitions)
       inputData.top(10).foreach(println)
       // == Step-3 ==
       ...
   ```

   In the preceding code, you must define a read function to parse and preprocess table data in MaxCompute. Code to define the read function:

   ```
   def read(record: Record, schema: TableSchema): String = {
         record.getString(0)
       }
   ```

3. Save the result data in Spark to a MaxCompute table.

   Use the saveToTable method of the OdpsOps object to save the result data in Spark to a MaxCompute table.

   ```
   val resultData = inputData.map(e => s"$e has been processed.")
       odpsOps.saveToTable(project, table, dataRDD, write)
   ```

In the preceding code, you must define a write function to preprocess the data. Code to define the write function:

```
def write(s: String, emptyReord: Record, schema: TableSchema): Unit = {
      val r = emptyReord
      r.set(0, s)
    }
```

4. Pay attention to the format of parameters for a partitioned table.

   When you use Spark to read data from or write data to a partitioned table in MaxCompute, you must specify a partition in the Partition key column name=Partition name format. If multiple partitions are involved, separate them with commas (,).

   - Example 1: To read data from the partition in which pt is 1, use pt='1'.
   - Example 2: To read data from the partition in which pt is 1 and the partition in which ps is 2, use pt='1', ps='2'.

Appendix

For the complete sample code, visit GitHub.

2.5. Use Spark to access Message Queue for Apache RocketMQ

This topic describes how to use Spark Streaming to consume data in Message Queue for Apache RocketMQ and calculate the number of words in each batch.

Use Spark to access Message Queue for Apache RocketMQ

Sample code:

```
val Array(cId, topic, subExpression, parallelism, interval) = args
  val accessKeyId = "<accessKeyId>"
  val accessKeySecret = "<accessKeySecret>"
  val numStreams = parallelism.toInt
  val batchInterval = Milliseconds(interval.toInt)
  val conf = new SparkConf().setAppName("Test ONS Streaming")
  val ssc = new StreamingContext(conf, batchInterval)
  def func: Message => Array[Byte] = msg => msg.getBody
  val onsStreams = (0 until numStreams).map { i =>
   println(s"starting stream $i")
   OnsUtils.createStream(
    ssc,
    cId,
    topic,
    subExpression,
    accessKeyId,
    accessKeySecret,
    StorageLevel.MEMORY_AND_DISK_2,
    func)
  }
  val unionStreams = ssc.union(onsStreams)
  unionStreams.foreachRDD(rdd => {
   rdd.map(bytes => new String(bytes)).flatMap(line => line.split(" "))
    .map(word => (word, 1))
    .reduceByKey(_ + _).collect().foreach(e => println(s"word: ${e._1}, cnt: ${e._2}"))
  })
  ssc.start()
  ssc.awaitTermination()
```

Appendix

For the complete sample code, visit GitHub.

2.6. Use Spark to access Tablestore

This topic describes how to use Spark to consume data in Tablestore.

Use Spark to access Tablestore

Create a table named pet in the Tablestore console and set the name field as the primary key.

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

name	owner	species	sex	birth	death
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	-

The following example shows how to consume data in the pet table:

```
private static RangeRowQueryCriteria fetchCriteria() {
  RangeRowQueryCriteria res = new RangeRowQueryCriteria("pet");
  res.setMaxVersions(1);
  List<PrimaryKeyColumn> lower = new ArrayList<PrimaryKeyColumn>();
  List<PrimaryKeyColumn> upper = new ArrayList<PrimaryKeyColumn>();
  lower.add(new PrimaryKeyColumn("name", PrimaryKeyValue.INF_MIN));
  upper.add(new PrimaryKeyColumn("name", PrimaryKeyValue.INF_MAX));
  res.setInclusiveStartPrimaryKey(new PrimaryKey(lower));
  res.setExclusiveEndPrimaryKey(new PrimaryKey(upper));
  return res;
}
public static void main(String[] args) {
  SparkConf sparkConf = new SparkConf().setAppName("RowCounter");
  JavaSparkContext sc = new JavaSparkContext(sparkConf);
  Configuration hadoopConf = new Configuration();
  JavaSparkContext sc = null;
  try {
    sc = new JavaSparkContext(sparkConf);
    Configuration hadoopConf = new Configuration();
    TableStore.setCredential(
        hadoopConf,
        new Credential(accessKeyId, accessKeySecret, securityToken));
    Endpoint ep = new Endpoint(endpoint, instance);
    TableStore.setEndpoint(hadoopConf, ep);
    TableStoreInputFormat.addCriteria(hadoopConf, fetchCriteria());
    JavaPairRDD<PrimaryKeyWritable, RowWritable> rdd = sc.newAPIHadoopRDD(
        hadoopConf, TableStoreInputFormat.class,
        PrimaryKeyWritable.class, RowWritable.class);
    System.out.println(
      new Formatter().format("TOTAL: %d", rdd.count()).toString());
  } finally {
    if (sc != null) {
      sc.close();
    }
  }
}
```

Appendix

For the complete sample code, visit GitHub.

2.7. Use Spark to consume Log Service data

2.7.1. Consume data in Log Service

This topic describes how to use Spark Streaming to consume log data in Log Service and calculate the number of log entries.

Use Spark to access Log Service

- Method 1: Receiver-based DStream

```
val logServiceProject = args(0)   // The name of a project in Log Service.
  val logStoreName = args(1)    // The name of a Logstore in Log Service.
  val loghubConsumerGroupName = args(2)  // Jobs that have the same consumer group name jointly con
sume the data in the Logstore.
  val loghubEndpoint = args(3)  // The endpoint of Log Service.
  val accessKeyId = "<accessKeyId>"    // The AccessKey ID that is used to access Log Service.
  val accessKeySecret = "<accessKeySecret>" // The AccessKey secret that is used to access Log Service.
  val numReceivers = args(4).toInt  // The number of receivers to be started to read data from the Logstor
e.
  val batchInterval = Milliseconds(args(5).toInt * 1000) // The data processing interval of Spark Streaming.
  val conf = new SparkConf().setAppName("Test Loghub Streaming")
  val ssc = new StreamingContext(conf, batchInterval)
  val loghubStream = LoghubUtils.createStream(
   ssc,
   logServiceProject,
   logStoreName,
   loghubConsumerGroupName,
   loghubEndpoint,
   numReceivers,
   accessKeyId,
   accessKeySecret,
   StorageLevel.MEMORY_AND_DISK)
  loghubStream.foreachRDD(rdd => println(rdd.count()))
  ssc.start()
  ssc.awaitTermination()
```

- Method 2: Direct API-based DStream

```
val logServiceProject = args(0)
  val logStoreName = args(1)
  val loghubConsumerGroupName = args(2)
  val loghubEndpoint = args(3)
  val accessKeyId = args(4)
  val accessKeySecret = args(5)
  val batchInterval = Milliseconds(args(6).toInt * 1000)
  val zkConnect = args(7)
  val checkpointPath = args(8)
  def functionToCreateContext(): StreamingContext = {
   val conf = new SparkConf().setAppName("Test Direct Loghub Streaming")
   val ssc = new StreamingContext(conf, batchInterval)
   val zkParas = Map("zookeeper.connect" -> zkConnect, "enable.auto.commit" -> "false")
   val loghubStream = LoghubUtils.createDirectStream(
    ssc,
    logServiceProject,
    logStoreName,
    loghubConsumerGroupName,
    accessKeyId,
    accessKeySecret,
    loghubEndpoint,
    zkParas,
    LogHubCursorPosition.END_CURSOR)
   ssc.checkpoint(checkpointPath)
   val stream = loghubStream.checkpoint(batchInterval)
   stream.foreachRDD(rdd => {
    println(rdd.count())
    loghubStream.asInstanceOf[CanCommitOffsets].commitAsync()
   })
   ssc
  }
  val ssc = StreamingContext.getOrCreate(checkpointPath, functionToCreateContext _)
  ssc.start()
  ssc.awaitTermination()
```

In E-MapReduce (EMR) SDK V1.4.0 and later, Spark Streaming can use Direct API-based DStreams to process data. If this method is used, data in LogHub is not repeatedly stored as write ahead logs (WALs). This method allows you to write data at least once without the need to enable the WAL feature of Spark Streaming. This method is still in the experimental stage. When you use this method, pay attention to the following points:

- When you perform a DStream action, you must make a commit.
- In a Spark Streaming job, you can perform only one action on a Logstore.
- This method requires the support of the ZooKeeper service.

Use MetaService for access

In the preceding sample code, an AccessKey pair is explicitly passed to the interface. In EMR SDK V1.3.2 and later, Spark Streaming can use MetaService to process data in Log Service. An AccessKey pair is not required. For more information, see the description of the LoghubUtils class in EMR SDK.

```
LoghubUtils.createStream(ssc, logServiceProject, logStoreName, loghubConsumerGroupName, storageLevel)
LoghubUtils.createStream(ssc, logServiceProject, logStoreName, loghubConsumerGroupName, numReceivers, storageLevel)
LoghubUtils.createStream(ssc, logServiceProject, logStoreName, loghubConsumerGroupName, storageLevel, cursorPosition, mLoghubCursorStartTime, forceSpecial)
LoghubUtils.createStream(ssc, logServiceProject, logStoreName, loghubConsumerGroupName, numReceivers, storageLevel, cursorPosition, mLoghubCursorStartTime, forceSpecial)
```

Note

- EMR SDK supports three consumption modes for Log Service, which are BEGIN_CURSOR, END_CURSOR, and SPECIAL_TIMER_CURSOR. By default, END_CURSOR is used.
  - BEGIN_CURSOR: consumes data from the log header. If a checkpoint record exists, the consumption starts from the checkpoint.
  - END_CURSOR: consumes data from the end of the log. If a checkpoint record exists, the consumption starts from the checkpoint.
  - SPECIAL_TIMER_CURSOR: consumes data from a specified point in time. Unit: seconds. If a checkpoint record exists, the consumption starts from the checkpoint.

  All of the consumption modes are affected by checkpoint records. If a checkpoint record exists, the consumption always starts from the checkpoint. The SPECIAL_TIMER_CURSOR mode allows you to forcibly start data consumption from a specified point in time. To use this mode, you must configure the following parameters as required in the createStream method of the LoghubUtils class:
  - cursorPosition: Set this parameter to LogHubCursorPosition.SPECIAL_TIMER_CURSOR.
  - orceSpecial: Set this parameter to true.
- All the nodes of an EMR cluster, except for the master node, cannot be connected to the Internet. Therefore, you must configure an internal endpoint of Log Service. Otherwise, you cannot request data from Log Service.

Appendix

For the complete sample code, visit GitHub.

2.7.2. Consume data in real time

This topic describes how to call Spark DataFrame API operations to develop a streaming job to consume Log Service data.

Sample code

```
## StructuredLoghubWordCount.Scala
object StructuredLoghubSample {
 def main(args: Array[String]) {
  if (args.length < 7) {
   System.err.println("Usage: StructuredLoghubSample <logService-project> " +
    "<logService-store> <access-key-id> <access-key-secret> <endpoint> " +
    "<starting-offsets> <max-offsets-per-trigger> [<checkpoint-location>]")
   System.exit(1)
  }
  val Array(project, logStore, accessKeyId, accessKeySecret, endpoint, startingOffsets, maxOffsetsPerTrigge
r, outputPath, _*) = args
  val checkpointLocation =
   if (args.length > 8) args(8) else "/tmp/temporary-" + UUID.randomUUID.toString
  val spark = SparkSession
   .builder
   .appName("StructuredLoghubSample")
   .master("local[5]")
   .getOrCreate()
  import spark.implicits. _
  // Create a dataset to represent the stream of input lines from LogHub.
  val lines = spark
   .readStream
   .format("loghub")
   .option("sls.project", project)
   .option("sls.store", logStore)
   .option("access.key.id", accessKeyId)
   .option("access.key.secret", accessKeySecret)
   .option("endpoint", endpoint)
   .option("startingoffsets", startingOffsets)
   .option("zookeeper.connect.address", "localhost:2181")
   .option("maxOffsetsPerTrigger", maxOffsetsPerTrigger)
   .load()
   .selectExpr("CAST(content AS STRING)")
   .as[String]
  val query = lines.writeStream
   .format("parquet")
   .option("checkpointLocation", checkpointLocation)
   .option("path", outputPath)
   .outputMode("append")
   .trigger(Trigger.ProcessingTime(30000))
   .start()
  query.awaitTermination()
 }
}
```

Note For more information about the Maven project object model (POM) file, see aliyun-emapreduce-demo.

Compilation and running

```
## Compile and run a command.
mvn clean package -DskipTests
## After the compiled command is run, the JAR file of the job is stored in the target/shaded/ directory.
## Submit and run the job.
spark-submit --master yarn-cluster --executor-cores 2 --executor-memory 1g --driver-memory 1g
--num-executors 2--class x.x.x.StructuredLoghubSample xxx.jar <logService-project>
<logService-store> <access-key-id> <access-key-secret> <endpoint> <starting-offsets>
<max-offsets-per-trigger> <zookeeper-connect-address> <output-path> <checkpoint-location>
```

Notice

- You need to specify the classpath and package path based on the actual situation in the format of x.x.x.StructuredLoghubSample and xxx.jar.
- You need to adjust the job resources based on the actual data size and cluster scale. If the cluster is too small, you may fail to run the job.

Notes

Some Spark versions are incompatible with E-MapReduce (EMR). Some EMR versions are incompatible with the emr-logservice SDK. The following table lists compatible Spark, EMR, and emr-logservice SDK versions.

emr-logservice SDK version	Spark version	EMR version
1.6.0	2.3.1	EMR 3.18.x or earlier
1.7.0	2.4.3	EMR 3.19.x or later

2.7.3. Consume data offline

This topic describes how to call Spark RDD API operations to develop an offline job to consume Log Service data.

Sample code

```
## TestBatchLoghub.Scala
object TestBatchLoghub {
 def main(args: Array[String]): Unit = {
  if (args.length < 6) {
   System.err.println(
    """Usage: TestBatchLoghub <sls project> <sls logstore> <sls endpoint>
     | <access key id> <access key secret> <output path> <start time> <end time=now>
    """.stripMargin)
   System.exit(1)
  }
  val loghubProject = args(0)
  val logStore = args(1)
  val endpoint = args(2)
  val accessKeyId = args(3)
  val accessKeySecret = args(4)
  val outputPath = args(5)
  val startTime = args(6).toLong
  val sc = new SparkContext(new SparkConf().setAppName("test batch loghub"))
  var rdd:JavaRDD[String] = null
  if (args.length > 7) {
   rdd = LoghubUtils.createRDD(sc, loghubProject, logStore, accessKeyId, accessKeySecret, endpoint, startT
ime, args(7).toLong)
  } else {
   rdd = LoghubUtils.createRDD(sc, loghubProject, logStore, accessKeyId, accessKeySecret, endpoint, startT
ime)
  }
  rdd.saveAsTextFile(outputPath)
 }
}
```

Note For more information about the Maven project object model (POM) file, see aliyun-emapreduce-demo.

Compilation and running

```
## Compile and run a command.
mvn clean package -DskipTests
## After the compiled command is run, the JAR file of the job is stored in the target/shaded/ directory.
## Submit and run the job.
spark-submit --master yarn-cluster --executor-cores 2 --executor-memory 1g --driver-memory 1g
--num-executors 2--class x.x.x.TestBatchLoghub xxx.jar <sls project> <sls logstore>
<sls endpoint> <access key id> <access key secret> <output path> <start time> [<end time=now>]
```

Notice

- You need to specify the classpath and package path based on the actual situation in the format of x.x.x.TestBatchLoghub and xxx.jar.
- You need to adjust the job resources based on the actual data size and cluster scale. If the cluster is too small, you may fail to run the job.

2.8. Use Spark to access MNS

This topic describes how to use Spark Streaming to consume data in Message Service (MNS) and calculate the number of words in each batch.

Use Spark to access MNS

Sample code:

```
val conf = new SparkConf().setAppName("Test MNS Streaming")
  val batchInterval = Seconds(10)
  val ssc = new StreamingContext(conf, batchInterval)
  val queuename = "queuename"
  val accessKeyId = "<accessKeyId>"
  val accessKeySecret = "<accessKeySecret>"
  val endpoint = "http://xxx.yyy.zzzz/abc"
  val mnsStream = MnsUtils.createPullingStreamAsRawBytes(ssc, queuename, accessKeyId, accessKeySecre
t, endpoint,
    StorageLevel.MEMORY_ONLY)
  mnsStream.foreachRDD( rdd => {
   rdd.map(bytes => new String(bytes)).flatMap(line => line.split(" "))
    .map(word => (word, 1))
    .reduceByKey(_ + _).collect().foreach(e => println(s"word: ${e._1}, cnt: ${e._2}"))
  })
  ssc.start()
  ssc.awaitTermination()
```

Use MetaService for access

In the preceding sample code, an AccessKey pair is explicitly passed to MNS. In E-MapReduce (EMR) SDK 1.3.2 and later, Spark Streaming can use MetaService to process MNS data. An AccessKey pair is not required. For more information, see the description of the MnsUtils class in EMR SDK:

```
MnsUtils.createPullingStreamAsBytes(ssc, queueName, endpoint, storageLevel)
MnsUtils.createPullingStreamAsRawBytes(ssc, queueName, endpoint, storageLevel)
```

Appendix

For the complete sample code, visit GitHub.

2.9. Use Spark to write data to HBase

This topic describes how Spark writes data to HBase.

> **Notice** computing clusters must be in the same security group as HBase clusters. Otherwise, the network cannot be connected. When you create a cluster in E-MapReduce, make sure that you select the security group where the HBase cluster is located.

Allow Spark to access HBase

Use the following code:

```
object ConnectionUtil extends Serializable {
  private val conf = HBaseConfiguration.create()
  conf.set(HConstants.ZOOKEEPER_QUORUM,"ecs1,ecs1,ecs3")
  conf.set(HConstants.ZOOKEEPER_ZNODE_PARENT, "/hbase")
  private val connection = ConnectionFactory.createConnection(conf)
  def getDefaultConn: Connection = connection
}
//Create data streaming unionStreams
unionStreams.foreachRDD(rdd => {
 rdd.map(bytes => new String(bytes))
  .flatMap(line => line.split(" "))
  .map(word => (word, 1))
  .reduceByKey(_ + _)
  .mapPartitions {words => {
   val conn = ConnectionUtil.getDefaultConn
   val tableName = TableName.valueOf(tname)
   val t = conn.getTable(tableName)
   try {
    words.sliding(100, 100).foreach(slice => {
     val puts = slice.map(word => {
      println(s"word: $word")
      val put = new Put(Bytes.toBytes(word. _1 + System.currentTimeMillis()))
      put.addColumn(COLUMN_FAMILY_BYTES, COLUMN_QUALIFIER_BYTES,
       System.currentTimeMillis(), Bytes.toBytes(word._2))
      put
     }).toList
     t.put(puts)
    })
   } finally {
    t.close()
   }
   Iterator.empty
  }}.count()
})
ssc.start()
ssc.awaitTermination()
```

Appendix

For the complete sample code, see Allow Spark to access HBase.

2.10. Use Spark to access Kafka

This topic describes how to run a Spark Streaming job in an E-MapReduce (EMR) Hadoop cluster to process data in a Kafka cluster.

Background information

EMR Hadoop and Kafka clusters run based on open source software. Therefore, you can use the relevant official documentation for reference during data development.

- Spark official documentation: streaming-kafka-integration and structured-streaming-kafka-integration

- EMR demo: GitHub

Methods to access Kafka clusters for which Kerberos authentication is enabled

EMR allows you to create Kafka clusters for which Kerberos authentication is enabled. You can run a job in a Hadoop cluster to access a Kafka cluster for which Kerberos authentication is enabled. The access method depends on whether Kerberos authentication is enabled for the Hadoop cluster:

- Hadoop cluster for which Kerberos authentication is disabled: Provide the *kafka_client_jaas.conf* and *krb5.conf* files that are used for the Kerberos authentication of the Kafka cluster.
- Hadoop cluster for which Kerberos authentication is enabled: Provide the *kafka_client_jaas.conf* and *krb5.conf* files that are used for the Kerberos authentication of the Hadoop cluster. The Kafka cluster can be authenticated based on the cross-domain trust feature of Kerberos authentication. For more information, see Cross-region access.

Both methods require you to provide the *kafka_client_jaas.conf* file to support Kerberos authentication when you run a job.

Content of the *kafka_client_jaas.conf* file:

```
KafkaClient {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  storeKey=true
  serviceName="kafka"
  keyTab="/path/to/kafka.keytab"
  principal="kafka/emr-header-1.cluster-12345@EMR.12345.COM";
};
```

- For information about how to obtain the *keytab* file, see Configure MIT Kerberos authentication.
- You can obtain the *krb5.conf* file from the */etc/* directory of the Kafka cluster.

Use Spark Streaming to access a Kafka cluster for which Kerberos authentication is enabled

Add the long domain name and IP address of each node of the Kafka cluster to the */etc/hosts* file for each node of the Hadoop cluster. The long domain name and IP address of a node can be obtained in the */etc/hosts* file for the node. A long domain name is in the format of `emr-xxx-x.cluster-xxx`.

When you run a Spark Streaming job to access a Kafka cluster for which Kerberos authentication is enabled, you can specify the *kafka_client_jaas.conf* and *kafka.keytab* files in the spark-submit command line.

```
spark-submit --conf "spark.driver.extraJavaOptions=-Djava.security.auth.login.config={{PWD}}/kafka_client_jaas.conf -Djava.security.krb5.conf={{PWD}}/krb5.conf" --conf "spark.executor.extraJavaOptions=-Djava.security.auth.login.config={{PWD}}//kafka_client_jaas.conf -Djava.security.krb5.conf={{PWD}}/krb5.conf" --files /local/path/to/kafka_client_jaas.conf,/local/path/to/kafka.keytab,/local/path/to/krb5.conf --class xx.xx.xx.KafkaSample --num-executors 2 --executor-cores 2 --executor-memory 1g --master yarn-cluster xxx.jar arg1 arg2 arg3
```

In the *kafka_client_jaas.conf* file, the path of the keytab file must be a relative path. Make sure the keyTab parameters are configured in the following format:

```
KafkaClient {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  storeKey=true
  serviceName="kafka"
  keyTab="kafka.keytab"
  principal="kafka/emr-header-1.cluster-12345@EMR.12345.COM";
};
```

Appendix

For the sample code, visit GitHub.

2.11. Use Spark to access MySQL

This topic describes how to run a Spark job in an E-MapReduce (EMR) Hadoop cluster to calculate the number of words and write the result to MySQL. You can run a Spark Streaming job to write data to MySQL in a similar way.

Use Spark to access MySQL

Sample code:

```
val input = getSparkContext.textFile(inputPath, numPartitions)
  input.flatMap(_.split(" ")).map(x => (x, 1)).reduceByKey(_ + _)
   .mapPartitions(e => {
    var conn: Connection = null
    var ps: PreparedStatement = null
    val sql = s"insert into $tbName(word, count) values (?, ?)"
    try {
     conn = DriverManager.getConnection(s"jdbc:mysql://$dbUrl:$dbPort/$dbName", dbUser, dbPwd)
     ps = conn.prepareStatement(sql)
     e.foreach(pair => {
      ps.setString(1, pair._1)
      ps.setLong(2, pair._2)
      ps.executeUpdate()
     })
     ps.close()
     conn.close()
    } catch {
     case e: Exception => e.printStackTrace()
    } finally {
     if (ps != null) {
      ps.close()
     }
     if (conn != null) {
      conn.close()
     }
    }
   Iterator.empty
  }).count()
```

Appendix

For the complete sample code, visit GitHub.

2.12. Configure spark-submit parameters

This topic describes how to configure spark-submit parameters in E-MapReduce.

Cluster configuration

- Software configuration

  E-MapReduce V1.1.0

  - Hadoop V2.6.0
  - Spark V1.6.0

- Hardware configuration
  - Master node
    - 8-core, 16 GB memory, and 500 GB storage space (ultra disk)
    - 1 set
  - Worker node
    - 8-core, 16 GB memory, and 500 GB storage space (ultra disk)
    - 10 sets
  - Total: 8-core 16 GB (Worker) × 10 + 8-core 16 GB (Master)

    > **Notice** Only CPU and memory resources are calculated when a job is submitted. Therefore, the disk size is not included in total resource calculation.

  - Total resources available for YARN: 12-core 12.8 GB (worker) × 10

    > **Notice** By default, cores available for YARN = number of cores × 1.5, and memory available for YARN = node memory × 0.8.

Submit a job

After you create a cluster, you can submit jobs. First, you need to create a job in E-MapReduce. The following figure shows the job parameters.

The job in the preceding figure uses the official Spark example package. Therefore, you do not need to upload your own JAR package.

The parameters are listed as follows:

```
--class org.apache.spark.examples.SparkPi --master yarn --deploy-mode client --driver-memory 4g --num-executors 2 --executor-memory 2g --executor-cores 2 /opt/apps/spark-1.6.0-bin-hadoop2.6/lib/spark-examples*.jar 10
```

The parameters are described as follows:

Parameter	Example	Description
class	org.apache.spark.examples.SparkPi	The main class of the job.
master	yarn	E-MapReduce uses the YARN mode. Set the value to yarn.
	yarn-client	Equivalent to setting the master parameter to yarn and the deploy-mode parameter to client. If you have set this parameter, then you do not need to set the deploy-mode parameter.
	yarn-cluster	Equivalent to setting the master parameter to yarn and the deploy-mode parameter to cluster. If you have set this parameter, then you do not need to set the deploy-mode parameter.

Parameter	Example	Description
deploy-mode	client	The client mode indicates that the ApplicationMaster (AM) of the job runs on the master node. If you set this parameter, you must also set the master parameter to yarn.
	cluster	The cluster mode indicates that the AM runs randomly on one of the worker nodes. If you set this parameter, you must also set the master parameter to yarn.
driver-memory	4g	The memory to be allocated to the driver. The allocated memory must not be greater than total memory size per node.
num-executors	2	The number of executors to be created.
executor-memory	2g	The maximum amount of memory to be allocated to each executor. The allocated memory cannot be greater than the maximum available memory per node.
executor-cores	2	The number of threads used by each executor, which equals the maximum number of tasks that can be executed concurrently by each executor.

Resource calculation

The resources consumed by jobs running in different modes and settings are shown in the following table:

- Resource calculation for the yarn-client mode

Node	Resource type	Resource amount (the result is calculated based on the preceding examples)
Master	Core	1 core
	Memory	driver-memory = 4 GB
	Core	num-executors × executor-cores = 4 cores

Worker Node	Resource type	Resource amount (the result is calculated based on the preceding examples)
	Memory	num-executors × executor-memory = 4 GB

- The main program of the job (the driver program) runs on the master node. As specified by the --driver-memory parameter, 4 GB memory is allocated to the main program based on the job settings. The main program may not use all of the allocated memory.
- As specified by the --num-executors parameter, two executors are initiated on work nodes. Each executor is allocated with 2 GB memory (specified by the --executor-memory parameter) and supports a maximum of 2 concurrent tasks (specified by the --executor-cores parameter).

- Resource calculation for the yarn-cluster mode

Node	Resource type	Resource amount (the result is calculated based on the preceding examples)
Master	N/A	A tiny client program, which is responsible for synchronizing job information and consumes only a small amount of resources.
Worker	Core	num-executors × executor-cores + spark.driver.cores = 5 cores
	Memory	num-executors × executor-memory + driver-memory = 8 GB

> **Note** The default value of spark.driver.cores is 1. You can set it to a value greater than 1.

Resource usage optimization

- yarn-client mode

  If you have a large job running in the yarn-client mode and want to use more resources of the cluster, see the following configurations:

```
--master yarn-client --driver-memory 5g --num-executors 20 --executor-memory 4g --executor-cores 4
```

> **Notice**
>
> - Spark will allocate 375 MB or 7% (whichever is higher) memory in addition to the memory value that you have set.
> - When allocating memory to containers, YARN rounds up to the nearest integer gigabyte. The memory value here must be a multiple of 1 GB.

  Based on the preceding resource formula:

- The resource amount for the master is as follows:
  - Cores: 1
  - Memory: 6 GB (5 GB + 375 MB, which is rounded up to 6 GB)
- The resource amount for the workers is as follows:
  - Core: 20 × 4 = 80 cores
  - Memory: 20 × 5 GB (4 GB + 375 MB, which is rounded up to 5 GB) = 100 GB

According to the resource calculation results, the amount of resources allocated to the job has not exceeded the total amount of the resources of the cluster. By following this rule, you can use other resource allocation configurations, such as:

```
--master yarn-client --driver-memory 5g --num-executors 40 --executor-memory 1g --executor-cores 2
```

```
--master yarn-client --driver-memory 5g --num-executors 15 --executor-memory 4g --executor-cores 4
```

```
--master yarn-client --driver-memory 5g --num-executors 10 --executor-memory 9g --executor-cores 6
```

Theoretically, you only need to make sure that the total amount of resources calculated by using the preceding formula does not exceed the total amount of the resources of the cluster. However, in production scenarios, the operating system, HDFS file systems, and E-MapReduce services may also need to use core and memory resources. If no core and memory resources are available for them, then the job performance declines or the job fails.

Typically, the executor-cores parameter is set to the same value as the number of cluster cores. If the value is too large, the CPU switches frequently without benefiting the performance as expected.

- yarn-cluster mode

  In the yarn-cluster mode, the driver program runs on worker nodes. Resources in the resource pool of the worker nodes are used. If you want to use more resources of the cluster, use the following configuration:

  ```
  --master yarn-cluster --driver-memory 5g --num-executors 15 --executor-memory 4g --executor-cores 4
  ```

Recommended configuration

- If you set the memory to a very large value, you should pay close attention to the overhead caused by garbage collection. Typically, we recommend that you assign memory less than or equal to 64 GB to an executor.
- If you are executing an HDFS read/write job, we recommend that you set the number of concurrent jobs for each executor to a value smaller than or equal to 5 for reading and writing data.
- If you are executing an OSS read/write job, we recommend that you distribute executors to different ECS instances so that the bandwidth of every ECS instance can be used. For example, if you have 10 ECS instances, you can set num-executors to 10, and set the appropriate memory and number of concurrent jobs.
- If the code that you use in the job is not thread-safe, you need to monitor whether the concurrency causes job errors when you set the executor-cores parameter. If yes, we recommend that you set executor-cores to 1.

3.Spark Streaming SQL

3.1. Common keywords

This topic introduces common keywords used in Spark Streaming SQL and describes how to use these keywords.

Background information

EMR V3.21.0 and later provide a preview release of Spark Streaming SQL for you to develop streaming analytics jobs.

Spark Streaming SQL is developed based on Spark Structured Streaming. All syntax functions and limits are compliant with Spark Structured Streaming.

Common keywords

Keyword type	Keyword
DDL	CREATE TABLE, CREATE TABLE AS SELECT, CREATE SCAN, CREATE STREAM
DML	INSERT INTO, MERGE INTO
SELECT clause	SELECT FROM, WHERE, GROUP BY, JOIN, UNION ALL

Use keywords as field names

If you want to use a keyword as a field name, enclose the keyword in backticks (`) in the format of `` `value` ``.

3.2. Streaming query

3.2.1. Job template (EMR V3.23.0 and later)

This topic describes how to use Spark SQL to develop a streaming query job in EMR V3.23.0 or later.

Job template

```
-- dbName: the database name
CREATE DATABASE IF NOT EXISTS ${dbName};
USE ${dbName};
-- Create a Log Service table.
-- slsTableName: the name of the Log Service table.
-- logProjectName: the name of the Log Service project.
-- logStoreName: the name of the Logstore in Log Service.
-- accessKeyId: your AccessKey ID.
-- accessKeySecret: your AccessKey secret.
-- endpoint: the endpoint of the Logstore in Log Service.
-- When you explicitly define a Logstore field, you must specify a value of the STRING data type.
-- Reserve the following system fields: `__logProject__` (STRING), `__logStore__` (STRING), `__shard__` (INT), `__time__` (TIMESTAMP), `__topic__` (STRING), and `__source__` (STRING).
CREATE TABLE IF NOT EXISTS ${slsTableName} (col1 dataType[, col2 dataType])
USING loghub
OPTIONS (
sls.project = '${logProjectName}',
sls.store = '${logStoreName}',
access.key.id = '${accessKeyId}',
access.key.secret = '${accessKeySecret}',
endpoint = '${endpoint}');
-- Create an HDFS table and define the column fields in the table.
-- hdfsTableName: the name of the HDFS table.
-- location: the data storage path. You can store data in HDFS or Object Storage Service (OSS).
-- Supported data formats: delta, csv, json, orc, and parquet. Default value: delta.
CREATE TABLE IF NOT EXISTS ${hdfsTableName} (col1 dataType[, col2 dataType])
USING delta
LOCATION '${location}';
-- Configure the method for reading tables. Valid values: STREAM and BATCH. Default value: BATCH.
CREATE SCAN tmp_read_sls_table
ON ${slsTableName}
USING STREAM;
-- Create a streaming query job.
CREATE STREAM ${queryName}
OPTIONS(
outputMode='Append',
triggerType='ProcessingTime',
triggerInterval='30000',
checkpointLocation='${checkpointLocation}')
INSERT INTO ${hdfsTableName}
SELECT col1, col2
FROM tmp_read_sls_table
WHERE ${condition};
```

> **Note** For more information about Endpoint, see Regions and endpoints.

3.2.2. Job template

This topic describes how to use Spark SQL to develop a streaming job.

> **Note** We recommend that you do not use this template in E-MapReduce V3.23.0 or later.

Query statement block

Some job parameters such as `streaming.query.name` cannot be expressed in SQL statements. Therefore, you must first use the SET statement to specify these parameters.

The following code shows a valid query statement block:

```
SET streaming.query.name=${queryName};
queryStatement
```

Job template

```
-- Create a database.
-- dbName: the database name.
CREATE DATABASE IF NOT EXISTS ${dbName};
USE ${dbName};
-- Create a Log Service table.
-- slsTableName: the name of the Log Service table.
-- logProjectName: the name of the Log Service project.
-- logStoreName: the name of the Logstore in Log Service.
-- accessKeyId: your AccessKey ID.
-- accessKeySecret: your AccessKey secret.
-- endpoint: the endpoint of the Logstore in Log Service.
-- When you explicitly define a Logstore field, you must specify a value of the STRING data type.
-- Reserve the following system fields: `__logProject__` (STRING), `__logStore__` (STRING), `__shard__` (I
NT), `__time__` (TIMESTAMP), `__topic__` (STRING), and `__source__` (STRING).
CREATE TABLE IF NOT EXISTS ${slsTableName} (col1 dataType[, col2 dataType])
USING loghub
OPTIONS (
sls.project = '${logProjectName}',
sls.store = '${logStoreName}',
access.key.id = '${accessKeyId}',
access.key.secret = '${accessKeySecret}',
endpoint = '${endpoint}');
-- Create an HDFS table and define the column fields in the table.
-- hdfsTableName: the name of the HDFS table.
-- location: the data storage path. You can store data in HDFS or Object Storage Service (OSS).
-- Supported data formats: delta, csv, json, orc, and parquet. Default value: delta.
CREATE TABLE IF NOT EXISTS ${hdfsTableName} (col1 dataType[, col2 dataType])
USING delta
LOCATION '${location}';
-- Define parameters for running a streaming query:
-- streaming.query.name: the name of the streaming query job.
-- spark.sql.streaming.checkpointLocation.${queryName}: the directory where the checkpoint file of the stre
aming query job is stored.
SET streaming.query.name=${queryName};
SET spark.sql.streaming.query.options.${queryName}.checkpointLocation=${checkpointLocation};
-- Optional parameters:
-- outputMode: the output mode of the query result. Default value: append.
-- trigger: the execution mode of the query. Default value: ProcessingTime. You can set this parameter only t
o ProcessingTime.
-- trigger.intervalMs: the interval between queries. Unit: milliseconds. Default value: 0.
-- SET spark.sql.streaming.query.outputMode.${queryName}=${outputMode};
SET spark.sql.streaming.query.trigger.${queryName}=ProcessingTime;
SET spark.sql.streaming.query.trigger.intervalMs.${queryName}=30;
INSERT INTO ${hdfsTableName}
SELECT col1, col2
FROM ${slsTableName}
WHERE ${condition};
```

> **Note** For more information about Endpoint, see Endpoints.

Parameters

The following table describes the key parameters.

Parameter	Description	Default value
streaming.query.name	The name of the streaming query job.	This parameter must be explicitly configured.
spark.sql.streaming.query.options.${queryName}.checkpointLocation	The directory where the checkpoint file of the streaming query job is stored.	This parameter must be explicitly configured.
spark.sql.streaming.query.outputMode.${queryName}	The output mode of the query result.	append
spark.sql.streaming.query.trigger.${queryName}	The execution mode of the query. You can set this parameter only to ProcessingTime.	ProcessingTime
spark.sql.streaming.query.trigger.intervalMs.${queryName}	The interval between queries. Unit: milliseconds.	0

3.2.3. Streaming query configuration

This topic describes basic concepts and related parameters of streaming query configuration.

Query configuration

Before using Spark SQL for a streaming query, you need to know the following two concepts:

- Data source configuration: the definition of a table.
- Query instance configuration: the parameter settings for running each streaming query.

The definition of a table only contains data source configurations, such as the connection address of the Kafka data source and the topic name. You can perform multiple non-business queries simultaneously in a table. Therefore, the definition of a table must not contain the configurations for running specific query instances.

Each query instance must be configured separately. To reduce unnecessary modifications to the query SQL statements, you can set a query name for each query instance. By using the query name, you can set parameters for running each specific query instance. The parameters of query instances are configured using the `SET` syntax. For more information, see Configuration parameters.

Conventions regarding query configuration: The query name for each query instance is available in the nearest SQL SET statement. Here are two examples:

- Case 1

```
SET streaming.query.name=one_test_job
-- query 1
INSERT INO tb_test_1 SELECT ...
-- query 2
INSERT INO tb_test_2 SELECT ...
-- The names of queries 1 and 2 are both one_test_job. However, this case is invalid because the name of each query instance must be unique.
```

- Case 2

```
SET streaming.query.name=one_test_job_1
SET streaming.query.name=one_test_job_2
-- query 1
CREATE TABLE tb_test_1 AS SELECT ...
-- The name of query 1 is one_test_job_2.
```

The statements for query instances include:

- `INSERT INTO ...`
- `CREATE TABLE ... AS SELECT ...`

Configuration parameters

Parameter	Corresponding DataFrame API	SQL statement format	Description	Required
queryName	writeStream.queryName(...)	SET streaming.query.name=$queryName	The name of each streaming query. The parameters of different query instances are distinguished by the query name.	Yes
option	writeStream.option(...)	SET spark.sql.streaming.query.options.$queryName.$optionName=$optionValue	checkpointLocation: the directory of the checkpoint.	Yes
			The custom option.	No
outputMode	writeStream.outputMode(...)	SET spark.sql.streaming.query.outputMode.$queryName=$outputMode	The output mode of the query result. Default value: *append*.	No
trigger	writeStream.trigger(...)	SET spark.sql.streaming.query.trigger.$queryName=$triggerType	The trigger controlling the moment where the query is executed. Default value: *ProcessingTime*. **Note** Currently, this parameter can only be set to *ProcessingTime*.	No
		SET spark.sql.streaming.query.trigger.intervalMs.$queryName=$intervalMs	The interval between query batches. Unit: milliseconds. Default value: 0.	No

3.3. DDL overview

3.3.1. DDL overview

This topic describes the data definition language (DDL) syntax in Spark SQL.

Syntax

```
CREATE TABLE tbName[(columnName dataType [,columnName dataType]*)]
USING providerName
OPTIONS(propertyName=propertyValue[,propertyName=propertyValue]*);
```

```
-- CTAS
CREATE TABLE tbName[(columnName dataType [,columnName dataType]*)]
USING providerName
OPTIONS(propertyName=propertyValue[,propertyName=propertyValue]*)
AS
queryStatement;
```

CREATE TABLE AS SELECT (CTAS) combines the statement to create a table and the statement to write query results to the table. The CTAS statement creates a table and generates a StreamQuery instance that queries data and writes the query results to the table.

Description

In the syntax, the field information of the table is optional. You must define the schema of the table based on the specific data source implementation.

- If you do not specify a schema in the CREATE TABLE statement, the system automatically identifies the schema of the data source.
- If you specify a schema in the CREATE TABLE statement, make sure that the schema is a subset of the schema of the data source and the data types are consistent.

Examples:

```
CREATE TABLE kafka_table
USING kafka
OPTIONS (
kafka.bootstrap.servers = "${BOOTSTRAP_SERVERS}",
subscribe = "${TOPIC_NAME}",
output.mode = "${OUTPUT_MODE}",
kafka.schema.registry.url = "${SCHEMA_REGISTRY_URL}",
kafka.schema.record.name = "${SCHEMA_RECORD_NAME}",
kafka.schema.record.namespace = "${SCHEMA_RECORD_NAMESPACE}");
```

For the Kafka data source, if the schema of a table is not specified, the system automatically retrieves the schema definition of the corresponding topic from Kafka Schema Registry.

```
CREATE TABLE kafka_table(col1 string, col2 double)
USING kafka
OPTIONS (
kafka.bootstrap.servers = "${BOOTSTRAP_SERVERS}",
subscribe = "${TOPIC_NAME}",
output.mode = "${OUTPUT_MODE}",
kafka.schema.registry.url = "${SCHEMA_REGISTRY_URL}",
kafka.schema.record.name = "${SCHEMA_RECORD_NAME}",
kafka.schema.record.namespace = "${SCHEMA_RECORD_NAMESPACE}");
```

For the Kafka data source, if the schema of a table is specified, the system automatically retrieves the schema definition of the corresponding topic from Kafka Schema Registry. Then, the system checks whether the table schema information, including the field names and types, is consistent with the topic schema information. We recommend that you define the same schema, including the names and data types of fields, in SQL declarations as the schema of the external data store.

3.3.2. CREATE SCAN

This topic describes the CREATE SCAN statement of Spark SQL. This statement is supported in EMR V3.23.0 and later.

Background information

The CREATE SCAN statement supports the data model of Spark Data Source API V2. You can use the CREATE SCAN statement to implement batch and streaming queries in Spark SQL in a centralized manner. For example, you can define a Kafka data source table and define a CREATE SCAN statement for batch read and another for streaming read.

When you define a table, specify only the basic information of the data source. You do not need to specify the method to read the parameters of the table. Take note of the following limits on the CREATE SCAN statement:

- You can use a view defined by the CREATE SCAN statement only as a data source table, not a data output table.
- You can use the CREATE SCAN statement to process raw tables, but raw tables support only batch read.

Syntax

```
CREATE SCAN tbName_alias
ON tbName
USING queryType
OPTIONS (propertyName=propertyValue[,propertyName=propertyValue]*)
```

You can set queryType to one of the following values:

- BATCH: implements batch read on source table tbName and defines temporary view tbName_alias.
- STREAM: implements streaming read on source table tbName and defines temporary view tbName_alias.

You can define the runtime parameters for reading the data source in the OPTIONS clause. The parameters vary depending on data sources. Most of the parameters are used to define streaming read.

Example

1. Create a Log Service data source table.

```
spark-sql> CREATE TABLE loghub_table_intput_test(content string)
    > USING loghub
    > OPTIONS
    > (...)
```

2. Process the Log Service table data offline by using the BATCH method, and count the number of data records generated till the current time.

```
spark-sql> CREATE SCAN loghub_table_intput_test_batch
    > ON loghub_table_intput_test
    > USING BATCH;
spark-sql> SELECT COUNT(*) FROM loghub_table_intput_test_batch;
```

3. Process the Log Service table data by using the STREAM method.

```
spark-sql> CREATE TABLE loghub_table_output_test(content string)
    > USING loghub
    > OPTIONS
    > (...);
spark-sql> CREATE SCAN loghub_table_intput_test_stream
    > ON loghub_table_intput_test
    > USING STREAM;
```

   The following code shows an invalid SELECT operation on a streaming table:

```
spark-sql> SELECT COUNT(*) FROM loghub_table_test_stream;
```

   An error is reported.

```
Error in query: Queries with streaming sources must be executed with writeStream.start();
```

   The following code shows a valid SELECT operation on a streaming table:

```
spark-sql> INSERT INTO loghub_table_output_test SELECT content FROM loghub_table_intput_test_stream;
```

3.3.3. STREAM statement

This topic describes the STREAM syntax in Spark SQL. The STREAM syntax is available in E-MapReduce V3.23.0 and later versions.

Why is the STREAM syntax required?

Before running a streaming query, you need to set parameters required for writeStream, such as checkpointLocation and outputMode. Currently, you can use the SET syntax to set these parameters for a query specified by the queryName parameter. However, this method has certain limitations. For example, you must set the parameters before running the query. To improve ease of use, E-MapReduce provides the STREAM syntax for you to set the parameters required for writeStream.

Note: E-MapReduce supports both the SET and STREAM syntaxes to set the parameters required for writeStream.

Syntax

```
CREATE STREAM queryName
OPTIONS (propertyName=propertyValue[,propertyName=propertyValue]*)
INSERT INTO tbName
queryStatement;
```

The following table lists the parameters required for writeStream.

Parameter	Description	Default value
checkpointLocation	The directory of the checkpoint for the streaming query job.	None
outputMode	The output mode of the query result.	Append
triggerType	The execution mode of the streaming query.	ProcessingTime
triggerIntervalMs	The interval between streaming queries. Unit: milliseconds.	0

Example

```
CREATE STREAM job1
OPTIONS(
checkpointLocation='/tmp/spark',
outputMode='Append',
triggerType='ProcessingTime'
triggerIntervalMs='3000')
INSERT INTO LargeOrders
SELECT * FROM Orders WHERE units > 1000;
```

3.4. DML overview

3.4.1. MERGE INTO

This topic describes how to use the MERGE INTO statement in Spark Streaming SQL.

Syntax

```
mergeInto
  : MERGE INTO target=tableIdentifier tableAlias
    USING (source=tableIdentifier (timeTravel)? | '(' subquery = query ')') tableAlias
    mergeCondition?
    matchedClauses*
    notMatchedClause?
mergeCondition
  : ON condition=booleanExpression
matchedClauses
  : deleteClause
  | updateClause
notMatchedClause
  : insertClause
deleteClause
  : WHEN MATCHED (AND deleteCond=booleanExpression)? THEN deleteAction
  | WHEN deleteCond=booleanExpression THEN deleteAction
updateClause
  : WHEN MATCHED (AND updateCond=booleanExpression)? THEN updateAction
  | WHEN updateCond=booleanExpression THEN updateAction
insertClause
  : WHEN NOT MATCHED (AND insertCond=booleanExpression)? THEN insertAction
  | WHEN insertCond=booleanExpression THEN insertAction
```

Example

- Define a source table and a target table.

```
-- source table
source_table
  : id int,      --primary key
  | name string,
  | opType string  --data operation type
-- target table
target_table
  : id int,      --primary key
  | name string
```

- Merge data into a Delta table.

```
MERGE INTO target_table t
USING source_table s
ON s.id = t.id
WHEN MATCHED AND s.opType = 'delete' THEN DELETE
WHEN MATCHED AND s.opType = 'update' THEN UPDATE SET id = s.id, name = s.name
WHEN NOT MATCHED AND s.opType = 'insert' THEN INSERT (key, value) VALUES (key, value)
```

- Merge data into a Kudu table.

```
MERGE INTO target_table t
USING source_table s
WHEN s.opType = 'delete' THEN DELETE
WHEN s.opType = 'update' THEN UPDATE SET *
WHEN s.opType = 'insert' THEN INSERT *
```

Description

Take note of the following rules for the MERGE INTO statement:

- You can use a maximum of three clauses.
- When you use the WHEN NOT MATCHED clause, you can specify a maximum of two WHEN clauses, and you must place the WHEN NOT MATCHED clause at the end of the statement.

Take note of the following rules for clauses in the MERGE INTO statement:

- WHEN MATCHED:
  - You can use only one UPDATE or DELETE operation.
  - You can specify an optional condition in each WHEN MATCHED clause. If you use two WHEN MATCHED clauses, the first one must contain a condition.
  - WHEN MATCHED clauses are executed in sequence. A data entry is updated or deleted based on only one WHEN MATCHED clause.
  - You can use `UPDATE SET *` to indicate `UPDATE SET column1 = source.column1 [, column2 = source.column2 ...]`. If the number of fields in the source table is different from that in the target table, a parsing exception is thrown.
- WHEN NOT MATCHED:
  - You can use only INSERT operations and can specify an optional condition.
  - You can use `INSERT *` to indicate `INSERT (column1 [, column2 ...]) VALUES (source.value1 [, source.value2 ...])`. If the number of fields in the source table is different from that in the target table, a parsing exception is thrown.

If the target storage system supports the UPSERT operation, you can use a variant of MERGE INTO to update or delete data. Take note of the following points on the variants of MERGE INTO:

- You can use a maximum of three WHEN clauses.
- `mergeCondition`: You do not need to configure a merge condition.
- You can omit the `MATCHED` and `NOT MATCHED` keywords.

3.4.2. DML overview

This topic describes how to use the INSERT INTO statement in Spark Streaming SQL.

Syntax

```
INSERT INTO tbName[(columnName[,columnName]*)]
queryStatement;
```

Example

```
INSERT INTO LargeOrders
SELECT * FROM Orders WHERE units > 1000;
```

Note

- Spark Streaming SQL does not allow you to use a separate SELECT statement for query. A SELECT statement must be used together with a CTAS statement or be contained in an INSERT INTO

statement.

- For a single job, an SQL file can contain multiple Data Manipulation Language (DML) operations, and multiple data sources and sinks.

3.5. Query overview

3.5.1. SELECT

The SELECT statement is used to read data from tables.

Syntax

```
SELECT [ DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression;
```

Test data

a (VARCHAR)	b (INT)	c (DATE)
a1	211	1990-02-20
b1	120	2018-05-12
c1	89	2010-06-14
a1	46	2016-04-05

Example 1

- Statement

```
SELECT * FROM table_name;
```

- Result

a (VARCHAR)	b (INT)	c (DATE)
a1	211	1990-02-20
b1	120	2018-05-12
c1	89	2010-06-14
a1	46	2016-04-05

Example 2

- Statement

```
SELECT a, c AS d FROM table_name;
```

- Result

a (VARCHAR)	d (DATE)
a1	1990-02-20
b1	2018-05-12
c1	2010-06-14
a1	2016-04-05

Example 3

- Statement

```
SELECT DISTINCT a FROM table_name;
```

- Result

a (VARCHAR)
a1
b1
c1

Subquery

In most scenarios, the SELECT statement reads data from several tables, for example, `SELECT column_1, column_2 ··· FROM table_name` .

> **Note** If the query object is another SELECT operation, you must specify an alias for the subquery.

- Example

```
INSERT INTO result_table
SELECT * FROM
        (SELECT  t.a,
             sum(t.b) AS sum_b
         FROM   t1 t
         GROUP BY t.a
        ) t1
WHERE t1.sum_b > 100;
```

- Result

a (VARCHAR)	b (INT)
a1	211
b1	120

a (VARCHAR)	b (INT)
a1	257

3.5.2. WHERE statement

A WHERE statement can be used to filter data generated from a SELECT statement.

Syntax

```
SELECT [ ALL | DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression
[ WHERE booleanExpression ];
```

3.5.3. GROUP BY statement

A GROUP BY statement groups a result set by one or more columns.

Syntax

```
SELECT [ DISTINCT ]
{ * | projectItem [, projectItem ]* }
FROM tableExpression
[ GROUP BY { groupItem [, groupItem ]* } ];
```

Example

```
SELECT Customer, SUM(OrderPrice) FROM xxx
GROUP BY Customer;
```

3.5.4. JOIN statement

Spark SQL supports joining batch processing data and streaming data, joining batch processing data, and joining streaming data. The JOIN statement has the same semantics as a traditional JOIN statement for batch processing.

Syntax

```
tableReference [, tableReference ]* | tableexpression
[ joinType ] JOIN tableexpression [ joinCondition ];
```

Constraint

When joining streaming data, note that some join types are not supported. For more information, see Spark official documentation. The following table lists some join types.

Left table	Right table	Join type	Supported
Stream	Static	Inner	Supported, not stateful.
		Left Outer	Supported, not stateful.
		Right Outer	Not supported.
		Full Outer	Not supported.
Static	Stream	Inner	Supported, not stateful.
		Left Outer	Not supported.
		Right Outer	Supported, not stateful.
		Full Outer	Not supported.
Stream	Stream	Inner	Supported. You can optionally specify a watermark on both sides and time constraints for state cleanup.
		Left Outer	Conditionally supported. You must specify a watermark on right and time constraints for correct results. You can optionally specify a watermark on left for all state cleanup.
		Right Outer	Conditionally supported. You must specify a watermark on left and time constraints for correct results. You can optionally specify a watermark on right for all state cleanup.
		Full Outer	Not supported.

3.5.5. UNION ALL statement

A UNION ALL statement is used to combine two data streams. The fields of the two data streams must be the same in terms of the field type and sequence.

Syntax

```
select_statement
UNION ALL
select_statement;
```

Example

```
SELECT
  a,
  sum(b),
FROM
  (SELECT * from tb_1
  UNION ALL
  SELECT * from tb_2
  )t
 GROUP BY a;
```

3.5.6. WATERMARK

The WATERMARK clause is used to process out-of-order data in streaming data queries. This topic describes the WATERMARK syntax and provides an example.

Syntax

```
SELECT watermark(projectItem, durationSpec) as watermarkItem, projectItem [, projectItem ]*
FROM tableExpression
```

You can use WATERMARK to solve data delay issues during streaming data processing. When Spark aggregates or joins data, it maintains intermediate state data. The watermark feature allows Spark to discard delayed data. When a data delay occurs, Spark also discards the intermediate state data that has expired from the memory. This feature ensures the stable running of streaming data queries.

For more information about WATERMARK, visit:

- Aggregate+Watermark
- Join+Watermark
- Policy for handling multiple watermarks

Example

The stream_source table must contain a field of the TIMESTAMP type to indicate event time. In the following examples, the maximum data delay is 10 seconds. <input class="dnt" value="Do Not Translate" readonly="readonly" tabindex="-1">

- Aggregate + Watermark

```
SELECT watermark(ts, interval 10 seconds), count(*) as col1, col2
FROM stream_source
GROUP BY ts, col2
```

- Join + Watermark

```
SELECT
  watermark(stream_source_1.ts, interval 20 second) as ts1,
  watermark(stream_source_2.ts, interval 10 second) as ts2,
  stream_source_1.col1 as col1,
  stream_source_2.col2 as col2
FROM
stream_source_1
INNER JOIN
stream_source_2
ON stream_source_1.col1=stream_source_2.col1
AND ts1 >= ts2
AND ts1 <= ts2 + interval 10 seconds
```

3.6. Window functions

3.6.1. Overview

This topic describes the window functions, window types, and time attribute that Spark Streaming SQL supports.

Window function

Window functions support aggregation over a specific window. For example, to calculate the number of users who visited a certain webpage in the past minute, you can define a window to collect data in the past minute, and calculate data in this window. Spark Streaming SQL supports the following two types of windows:

- Tumbling window
- Sliding window

Time attribute

Spark SQL supports aggregation of data in a window by using the event time attribute.

Generally, the event time attribute indicates the time when a data record was created. It is provided in the schema.

Note

For queries with time windows, a window column is auto-generated, which contains the window start and end time specified by window.start and window.end.

3.6.2. Tumbling window

This topic describes how to use a tumbling window function in Spark Streaming SQL.

What is a tumbling window?

When tumbling windows are used, each element is assigned to a window with the specified size. Tumbling windows are a series of fixed-sized, non-overlapping, and contiguous time intervals. For example, if a 5-minute tumbling window is defined, elements are assigned to windows based on time periods: [0:00, 0:05), [0:05, 0:10), [0:10, 0:15), and so on.

Syntax of a tumbling window function

```
GROUP BY TUMBLING ( colName, windowDuration )
```

Example

```
SELECT avg(inv_quantity_on_hand) qoh
FROM kafka_inventory
GROUP BY TUMBLING (inv_data_time, interval 1 minute)
```

3.6.3. Sliding window

This topic describes how to use the sliding window function in Spark Streaming SQL.

What is a sliding window?

A sliding window is also called a hop window. Unlike tumbling windows, sliding windows can overlap each other. A sliding window has two parameters: windowDuration and slideDuration. The slideDuration parameter indicates the step size of each slide. The windowDuration parameter indicates the window size.

- If the value of the slideDuration parameter is less than that of the windowDuration parameter, the windows overlap each other, and each element is assigned to multiple windows.
- If the value of the slideDuration parameter equals that of the windowDuration parameter, the windows are tumbling windows.

Syntax

```
GROUP BY HOPPING (colName, windowDuration, slideDuration)
```

Example

```
SELECT avg(inv_quantity_on_hand) qoh
FROM kafka_inventory
GROUP BY HOPPING (inv_data_time, interval 1 minute, interval 30 second)
```

3.7. Data sources

3.7.1. Overview

You can use Spark SQL to develop streaming analytics jobs in EMR V3.21.0 and later. This topic describes the data sources supported by Spark SQL and methods to process data in the data sources.

Data sources

Data source	Batch read	Batch write	Streaming read	Streaming write
Kafka	Supported	Not supported	Supported	Supported
Loghub	Supported	Supported	Supported	Supported
Tablestore	Supported	Supported	Supported	Supported
DataHub	Not supported	Not supported	Supported	Supported
HBase	Supported	Supported	Not supported	Supported
JDBC	Supported	Supported	Not supported	Supported
Druid	Not supported	Not supported	Not supported	Supported
Redis	Not supported	Not supported	Not supported	Supported
Kudu	Supported	Supported	Not supported	Supported
DTS	Supported	Not supported	Supported	Not supported

Methods to process data in the data sources

You can use one of the following methods to process data in the data sources:

- Command line
  i. Download the data source JAR package that has been precompiled.

     The JAR package contains the implementation packages and related dependency packages of the Loghub, Tablestore, HBase, JDBC, and Redis data sources. The packages for the Kafka and Druid data sources are not contained in this JAR package and will be supplemented later. For more information, see Release notes.

  ii. Use the **streaming-sql** command line for interactive development.

```
[hadoop@emr-header-1 ~]# streaming-sql --master yarn-client --jars emr-datasources_shaded_2.11-${version}.jar --driver-class-path emr-datasources_shaded_2.11-${version}.jar
```

- Workflow

  For more information, see Configure a Streaming SQL job.

3.7.2. Kafka data source

This topic describes how to use Spark SQL to perform data analysis and interactive development on the Kafka data source.

CREATE TABLE syntax

```
CREATE TABLE tbName[(columnName dataType [,columnName dataType]*)]
USING kafka
OPTIONS(propertyName=propertyValue[,propertyName=propertyValue]*);
```

Parameters

Parameter	Description	Required
subscribe	The name of the associated Kafka topic.	Yes
kafka.bootstrap.servers	The connection address of the Kafka cluster.	Yes

References

For more information about the Kafka data source, see Structured Streaming + Kafka Integration Guide.

3.7.3. LogHub data source

This topic describes how to use Spark Streaming SQL to perform data analysis and interactive development on the LogHub data source.

CREATE TABLE syntax

```
CREATE TABLE tbName(columnName dataType [,columnName dataType]*)
USING loghub
OPTIONS(propertyName=propertyValue[,propertyName=propertyValue]*);
```

Table schema

When you create a LogHub data table, you must explicitly define the fields in the data table. Custom field names in the table schema must be the same as key names in Log Service.

```
spark-sql> CREATE TABLE loghub_table_test(content string)
     > USING loghub
     > OPTIONS(
     > endpoint="sls.aliyuncs.com",
     > access.key.id="yHiu*******BG2s",
     > access.key.secret="ABctuw0M***************iKKljZy",
     > sls.project="test",
     > sls.store="myInstance");
spark-sql> DESC loghub_table_test;
content  string  NULL
Time taken: 0.436 seconds, Fetched 1 row(s)
```

CREATE TABLE syntax, with no schema specified:

```
spark-sql> CREATE TABLE loghub_table_test
      > USING loghub
      > OPTIONS
      > (...)
spark-sql> DESC loghub_table_test;
__logProject__ string  NULL
__logStore__ string NULL
__shard__ string NULL
__time__ string NULL
__topic__ string NULL
__source__ string NULL
__value__ string NULL
__sequence_number__ string NULL
Time taken: 0.436 seconds, Fetched 1 row(s)
```

Parameters

Parameter	Description	Required
endpoint	The endpoint of the Log Service API.	Yes
access.key.id	The AccessKey ID.	Yes
access.key.secret	The AccessKey secret.	Yes
sls.store	The name of the Log Service project.	Yes
sls.project	The name of the Logstore.	Yes

The LogHub schema is optional for EMR SDK 2.0.0 or later. The following table describes relevant parameters.

Parameter	Type	Description
__logProject__	STRING	The name of the Logstore.
__logStore__	STRING	The name of the Log Service project.
__shard__	STRING	The shard of the Logstore.
__time__	STRING	The time when the log entry was created.
__topic__	STRING	The topic of the log.
__source__	STRING	The source IP address of Log Service.

Parameter	Type	Description
__value__	STRING	The content of Log Service, in the JSON format.
__sequence_number__	STRING	The sequence number of the record. This field can be specified only when appendSequenceNumber is set to true. Default value: NULL.

3.7.4. HBase data source

This topic describes how to use Spark Streaming SQL to perform data analysis and interactive development on the HBase data source.

CREATE TABLE syntax

```
CREATE TABLE tbName
USING hbase
OPTIONS(propertyName=propertyValue[,propertyName=propertyValue]*);
```

Table schema

When you create an HBase data table, you do not need to explicitly define the fields in the data table. Example:

```
spark-sql> CREATE DATABASE IF NOT EXISTS default;
spark-sql> USE default;
spark-sql> DROP TABLE IF EXISTS hbase_table_test;
spark-sql> CREATE TABLE hbase_table_test
     > USING hbase
     > OPTIONS(
     > catalog='{"table":{"namespace":"default","name":"test"},"rowkey":"key", "columns":{"key":{"cf":"ro
wkey", "col":"key", "type":"string"},"data":{"cf":"info", "col":"data", "type":"string"}}}',
     > hbaseConfiguration='{"hbase.zookeeper.quorum":"a.b.c.d:2181"}');
spark-sql> DESC hbase_table_test;
key string  NULL
data  string  NULL
Time taken: 0.436 seconds, Fetched 2 row(s)
```

Parameters

Parameter	Description	Required
catalog	The description of fields in an HBase data table, in the JSON format.	Yes

Parameter	Description	Required
hbaseConfiguration	The HBase configuration, in the JSON format. For example, set the HBase endpoint to `{"hbase.zookeeper.quorum":"a.b.c.d:2181"}`.	Yes

The following example shows the catalog configuration for the schema of an HBase data table named table1:

```
{
 "table":{"namespace":"default", "name":"table1"},
 "rowkey":"key",
 "columns":{
  "col0":{"cf":"rowkey", "col":"key", "type":"string"},
  "col1":{"cf":"cf1", "col":"col1", "type":"boolean"},
  "col2":{"cf":"cf2", "col":"col2", "type":"double"},
  "col3":{"cf":"cf3", "col":"col3", "type":"float"},
  "col4":{"cf":"cf4", "col":"col4", "type":"int"},
  "col5":{"cf":"cf5", "col":"col5", "type":"bigint"},
  "col6":{"cf":"cf6", "col":"col6", "type":"smallint"},
  "col7":{"cf":"cf7", "col":"col7", "type":"string"},
  "col8":{"cf":"cf8", "col":"col8", "type":"tinyint"}
 }
}
```

3.7.5. JDBC data source

This topic describes how to use Spark Streaming SQL to perform data analysis and interactive development on the JDBC data source.

CREATE TABLE syntax

```
CREATE TABLE tbName
USING jdbc2
OPTIONS(propertyName=propertyValue[,propertyName=propertyValue]*);
```

Table schema

When you create a JDBC data table, you do not need to explicitly define the fields in the data table. Example:

```
spark-sql> CREATE DATABASE IF NOT EXISTS default;
spark-sql> USE default;
spark-sql> DROP TABLE IF EXISTS rds_table_test;
spark-sql> CREATE TABLE rds_table_test
    > USING jdbc2
    > OPTIONS (
    > url="jdbc:mysql://rm-bp11*********i7w9.mysql.rds.aliyuncs.com:3306/default? useSSL=true",
    > driver="com.mysql.jdbc.Driver",
    > dbtable="test",
    > user="root",
    > password="thisisapassword",
    > batchsize="100",
    > isolationLevel="NONE");
spark-sql> DESC rds_table_test;
id  int NULL
name  string  NULL
Time taken: 0.413 seconds, Fetched 2 row(s)
```

Parameters

Parameter	Description	Required
url	The URL of the database.	Yes
driver	The JDBC driver connected to the database. Example: `com.mysql.jdbc.Driver"eper.quorum":"a.b.c.d:2181"}` .	Yes
dbtable	The name of the data table.	Yes
user	The username of the account that is used to connect to the database.	Yes
password	The password of the account that is used to connect to the database.	Yes
batchsize	The number of data entries updated in each batch. This parameter takes effect only when you write data into a database.	No
isolationLevel	The level of transaction isolation. Default value: READ_UNCOMMITTED.	No

The following table describes the transaction isolation levels and their support to read phenomena.

Transaction isolation level	Dirty read	Non-repeatable read	Phantom read
READ_UNCOMMITTED	Supported	Supported	Supported
READ_COMMITTED	Not supported	Supported	Supported
REPEATABLE_READ	Not supported	Not supported	Supported
SERIALIZABLE	Not supported	Not supported	Not supported
NONE	Not supported	Not supported	Not supported

Write data

If you want to write data into a database, you can run the following command to set an associated SQL statement:

```
spark-sql> SET streaming.query.${queryName}.sql=insert into `test` (`id`,`name`) values(?, ?) ;
spark-sql> SET ...
spark-sql> INSERT INTO rds_table_test SELECT ...
```

3.7.6. Tablestore data source

This topic describes how to use Spark Streaming SQL to perform data analysis and interactive development on the Tablestore data source.

CREATE TABLE syntax

```
CREATE TABLE tbName
USING tablestore
OPTIONS(propertyName=propertyValue[,propertyName=propertyValue]*);
```

Table schema

When you create a Tablestore data table, you do not need to explicitly define the fields in the data table. Example:

```
spark-sql> CREATE DATABASE IF NOT EXISTS default;
spark-sql> USE default;
spark-sql> DROP TABLE IF EXISTS ots_table_test;
spark-sql> CREATE TABLE ots_table_test
    > USING tablestore
    > OPTIONS(
    > endpoint="http://xxx.cn-hangzhou.vpc.ots.aliyuncs.com",
    > access.key.id="yHiu*******BG2s",
    > access.key.secret="ABctuw0M***************iKKljZy",
    > table.name="test",
    > instance.name="myInstance",
    > batch.update.size="100",
    > catalog='{"columns":{"pk":{"col":"pk","type":"string"},"data":{"col":"data","type":"string"}}}');
spark-sql> DESC ots_table_test;
pk string NULL
data string NULL
Time taken: 0.501 seconds, Fetched 2 row(s)
```

Parameters

Parameter	Description
access.key.id	The AccessKey ID.
access.key.secret	The AccessKey secret.
endpoint	The endpoint of the Tablestore API.
table.name	The name of the Tablestore data table.
instance.name	The name of the Tablestore instance.
batch.update.size	The number of data entries updated in the Tablestore data table in each batch. The default value is 0, which indicates that data entries are updated one by one. This parameter takes effect only when you write data into a database.
catalog	The description of fields in the Tablestore data table, in the JSON format.

The following example shows the catalog configuration for the schema of a Tablestore data table named table1:

```
{"columns":{
 "col0":{"cf":"cf0", "col":"col0", "type":"string"},
 "col1":{"cf":"cf1", "col":"col1", "type":"boolean"},
 "col2":{"cf":"cf2", "col":"col2", "type":"double"},
 "col3":{"cf":"cf3", "col":"col3", "type":"float"},
 "col4":{"cf":"cf4", "col":"col4", "type":"int"},
 "col5":{"cf":"cf5", "col":"col5", "type":"bigint"},
 "col6":{"cf":"cf6", "col":"col6", "type":"smallint"},
 "col7":{"cf":"cf7", "col":"col7", "type":"string"},
 "col8":{"cf":"cf8", "col":"col8", "type":"tinyint"}
}
```

3.7.7. DataHub data source

This topic describes how to use Spark Streaming SQL to perform data analysis and interactive development on the DataHub data source.

CREATE TABLE syntax

```
CREATE TABLE tbName
USING datahub
OPTIONS(propertyName=propertyValue[,propertyName=propertyValue]*);
```

Table schema

When you create a DataHub data table, you do not need to explicitly define the fields in the data table. Example:

```
spark-sql> CREATE TABLE datahub_table_test
    > USING datahub
    > OPTIONS
    > ( access.key.id = '<your access key id>',
    >  access.key.secret = '<you access key secret>',
    >  endpoint = '<your end point>',
    >  project = '<your project name>',
    >  topic = '<your topic>'
    > )
spark-sql> DESC datahub_table_test;
id  string  NULL
name string NULL
Time taken: 0.401 seconds, Fetched 2 row(s)
```

Parameters

Parameter	Description	Required
access.key.id	The AccessKey ID.	Yes
access.key.secret	The AccessKey secret.	Yes

Parameter	Description	Required
endpoint	The endpoint of the DataHub API.	Yes
project	The name of the DataHub project.	Yes
topic	The name of the DataHub topic.	Yes
decimal.precision	Specify this parameter if a field of the DECIMAL type is contained in the topic.	No
decimal.scale	Specify this parameter if a field of the DECIMAL type is contained in the topic.	No

3.7.8. Druid data source

This topic describes how to use Spark Streaming SQL to perform data analysis and interactive development on the Druid data source.

CREATE TABLE syntax

```
create table tbName
using druid
options(propertyKey=propertyValue[, propertyKey=propertyValue]*);
```

Table schema

When you create a Druid data table, you do not need to explicitly define the fields in the data table. Example:

```
create table druid_test_table
using druid
options(
curator.connect="${ZooKeeper-host}:${ZooKeeper-port}}",
index.service="druid/overlord",
data.source="test_source",
discovery.path="/druid/discovery",
firehose="druid:firehose:%s",
rollup.aggregators="{\"metricsSpec\":[{\"type\":\"count\",\"name\":\"count\"},
                    {\"type\":\"doubleSum\",\"fieldName\":\"value\",\"name\":\"sum\"},
                    {\"type\":\"doubleMin\",\"fieldName\":\"value\",\"name\":\"min\"},
                    {\"type\":\"doubleMax\",\"fieldName\":\"value\",\"name\":\"max\"}]}",
rollup.dimensions="timestamp,metric,userId",
rollup.query.granularities="minute",
tuning.segment.granularity="FIVE_MINUTE",
tuning.window.period="PT5M",
timestampSpec.column="timestamp",
timestampSpec.format="posix");
```

Parameters

Parameter	Description	Required
curator.connect	The host and port of ZooKeeper, such as emr-header-1:2181.	Yes
curator.max.retries	The maximum number of connection retries upon a ZooKeeper connection failure. Default value: 5.	No
curator.retry.base.sleep	The initial interval at which a connection retry is made upon a ZooKeeper connection failure. Default value: 100. Unit: milliseconds.	No
curator.retry.max.sleep	The maximum interval at which a connection retry is made upon a ZooKeeper connection failure. Default value: 3000. Unit: milliseconds.	No
index.service	The indexing service, such as druid or overlord.	Yes
data.source	The name of the data source from which data is written into Druid.	Yes
discovery.path	The discovery path of Druid. The default value is druid or discovery.	No
firehose	The firehose, such as `druid:firehose:%s`.	Yes

rollup.aggregators	The rollup aggregators of Tranquility, in the JSON format. Example:
	`{\"metricsSpec\":[{\"type\":\"count\",\"name\":\"count\"},`
	`{\"type\":\"doubleSum\",\"fieldName\":\"value\",\"name\":\"sum\"},`
	`{\"type\":\"doubleMin\",\"fieldName\":\"value\",\"name\":\"min\"},`
	`{\"type\":\"doubleMax\",\"fieldName\":\"value\",\"name\":\"max\"}]}`
	where, `metricsSpec` is fixed.
		Yes

rollup.dimensions	The dimension from which data is written into Druid.	Yes
rollup.query.granularities	The rollup granularity, such as minute.	Yes
tuning.window.period	The size of the time window. Default value: PT10M.	No
tuning.segment.granularity	The segment granularity. Default value: DAY.	No

Parameter	Description	Required
tuning.partitions	The number of partitions. Default value: 1.	No
tuning.replications	The number of replicas. Default value: 1.	No
timestampSpec.column	The name of the timestamp column when data is written into Druid. Default value: timestamp.	No
timestampSpec.format	The format of the timestamp column name when data is written into Druid. Default value: iso.	No

3.7.9. Redis data source

This topic describes how to use Spark Streaming SQL to perform data analysis and interactive development on the Redis data source.

CREATE TABLE syntax

```
CREATE TABLE tbName[(columnName dataType [,columnName dataType]*)]
USING redis
OPTIONS(propertyKey=propertyValue[, propertyKey=propertyValue]*);
```

Table schema

When you create a Redis data table, you must explicitly define the fields in the data table. Example:

```
spark-sql> CREATE TABLE redis_test_table(`key0` STRING, `value0` STRING, `key1` STRING, `value1` STRING)
    > USING redis
    > OPTIONS(
    > table="test",
    > redis.save.mode="append",
    > model="hash",
    > filter.keys.by.type="false",
    > key.column="uuid",
    > max.pipeline.size="100",
    > host="localhost",
    > port="6379",
    > dbNum="0");
```

Parameters

Parameter	Description	Required

Parameter	Description	Required
table	The prefix of the keys that are used to write data into Redis. The format of a key is `${table}:${key.column}`, in which ${table} indicates the prefix and `${key.column}` indicates the configuration item.	Yes
redis.save.mode	The processing method if data to be written already exists in Redis. Valid values: append (append data to be written to the existing data), overwrite (overwrite the existing data), errorifexists (throw an exception), and ignore (discard the existing data). Default value: append.	No
model	The storage format of data. Valid values: hash and binary. Default value: hash.	No
filter.keys.by.type	Specifies whether to filter out the data that does not conform to the data storage format. Default value: false.	No
key.column	The column of keys that are used to write data into Redis. If you do not specify this parameter, universally unique identifiers (UUIDs) are used as keys.	No
ttl	The time to live (TTL) of a key. If you do not specify this parameter, data is stored permanently. If you specify this parameter, the value indicates the expiration time. Unit: seconds.	No
max.pipeline.size	The maximum number of bulk data write operations that a pipeline can support. Default value: 100.	No
host	The IP address of the on-premises machine where the Redis instance is deployed. Default value: localhost.	No

Parameter	Description	Required
port	The port of the Redis instance. Default value: 6379.	No
dbNum	The sequence number of the database to which data is stored. Default value: 0.	No

4.Hadoop

4.1. Parameter description

Description of parameters in Hadoop code

The following parameters can be used in Hadoop code:

Property	Default	Description
fs.oss.accessKeyId	None	(Optional) The required AccessKey ID used to access OSS.
fs.oss.accessKeySecret	None	(Optional) The required AccessKey Secret used to access OSS.
fs.oss.securityToken	None	(Optional) The required STS token used to access OSS.
fs.oss.endpoint	None	(Optional) The endpoint used to access OSS.
fs.oss.multipart.thread.number	5	The number of threads (concurrency) used by OSS for upload part copy.
fs.oss.copy.simple.max.byte	134217728	The file limit of the internal copy in OSS using common APIs.
fs.oss.multipart.split.max.byte	67108864	The file multipart limit of copying files between buckets in OSS using common APIs.
fs.oss.multipart.split.number	5	The file multipart limit of copying files between buckets in OSS that is using common APIs. By default, the limit is equal to the number of threads used in the copy.
fs.oss.impl	com.aliyun.fs.oss.nat.NativeOssFileSystem	The implementation class for the native OSS file system.
fs.oss.buffer.dirs	/mnt/disk1,/mnt/disk2,...	OSS local temporary directory. It uses a cluster data disk by default.
fs.oss.buffer.dirs.exists	false	Indicates whether the OSS temporary directory exists.
fs.oss.client.connection.timeout	50000	The connection timeout for the OSS client (ms).

Property	Default	Description
fs.oss.client.socket.timeout	50,000	The socket timeout for the OSS client (ms).
fs.oss.client.connection.ttl	-1	The connection alive time
fs.oss.connection.max	1024	The maximum connections allowed.
io.compression.codec.snappy.native	false	Indicates whether to mark Snappy files as standard. By default, Hadoop recognizes Snappy files modified by Hadoop.

4.2. Develop a MapReduce job

This topic uses an E-MapReduce (EMR) cluster of V3.27.0 as an example to show how to develop a MapReduce job in an EMR cluster.

Process OSS data by using a MapReduce job

To read data from or write data to Object Storage Service (OSS) by using a MapReduce job, you must configure the following parameters:

```
conf.set("fs.oss.accessKeyId", "${accessKeyId}");
conf.set("fs.oss.accessKeySecret", "${accessKeySecret}");
conf.set("fs.oss.endpoint","${endpoint}");
```

Parameter description:

- `${accessKeyId}` : the AccessKey ID of your Alibaba Cloud account.
- `${accessKeySecret}` : the AccessKey secret of your Alibaba Cloud account.
- `${endpoint}` : the endpoint of OSS.

  Set this parameter to the endpoint that corresponds to the region where your EMR cluster resides. You must make sure that the OSS bucket you want to use resides in the same region as the EMR cluster. For more information about regions and endpoints, see Regions and endpoints.

WordCount example

This example demonstrates how to use a MapReduce job to read text from OSS, count the number of occurrences of each word in the text, and then write the results back to the OSS bucket.

1. Log on to your cluster in SSH mode. For more information, see Log on to a cluster.
2. Run the following command to create a directory named *wordcount_classes*:

   ```
   mkdir wordcount_classes
   ```

3. Create a file named *EmrWordCount.java*:

i. Run the following command to create a file named *EmrWordCount.java* and open the file:

```
vim EmrWordCount.java
```

ii. Press the `I` key to switch to the edit mode.

iii. Add the following information to the *EmrWordCount.java* file:

```
package org.apache.hadoop.examples;
 import java.io.IOException;
 import java.util.StringTokenizer;
 import org.apache.hadoop.conf.Configuration;
 import org.apache.hadoop.fs.Path;
 import org.apache.hadoop.io.IntWritable;
 import org.apache.hadoop.io.Text;
 import org.apache.hadoop.mapreduce.Job;
 import org.apache.hadoop.mapreduce.Mapper;
 import org.apache.hadoop.mapreduce.Reducer;
 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
 import org.apache.hadoop.util.GenericOptionsParser;
 public class EmrWordCount {
  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    public void map(Object key, Text value, Context context
                ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }
  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();
    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }
  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length < 2) {
      System.err.println("Usage: wordcount <in> [<in>...] <out>");
      System.exit(2);
    }
```

```
    }
    conf.set("fs.oss.accessKeyId", "${accessKeyId}");
    conf.set("fs.oss.accessKeySecret", "${accessKeySecret}");
    conf.set("fs.oss.endpoint","${endpoint}");
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(EmrWordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    for (int i = 0; i < otherArgs.length - 1; ++i) {
      FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
    }
    FileOutputFormat.setOutputPath(job,
      new Path(otherArgs[otherArgs.length - 1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

iv. Press `Esc` to exit the edit mode. Then, enter `:wq` to save and close the file.

4. Compile and package the created file.

   i. Run the following command to compile the program:

   ```
   javac -classpath <HADOOP_HOME>/share/hadoop/common/hadoop-common-X.X.X.jar:<HADOOP_HOME>/share/hadoop/mapreduce/hadoop-mapreduce-client-core-X.X.X.jar:<HADOOP_HOME>/share/hadoop/common/lib/commons-cli-1.2.jar -d wordcount_classes EmrWordCount.java
   ```

   - `HADOOP_HOME`: the installation directory of Hadoop. In most cases, the */usr/lib/hadoop-current* directory is used.

     You can run the `env |grep hadoop` command to obtain the installation directory.
   - `X.X.X`: the version of the JAR package. It must be the same as the version of Hadoop in the cluster.

     For the *hadoop-common-X.X.X.jar* file, you can view the version in the *<HADOOP_HOME>/share/hadoop/common/* directory. For the *hadoop-mapreduce-client-core-X.X.X.jar* file, you can view the version in the *<HADOOP_HOME>/share/hadoop/mapreduce/* directory.

   ii. Run the following command to package the compiled program into a JAR file:

   ```
   jar cvf wordcount.jar -C wordcount_classes .
   ```

   > **Note** In this example, the JAR file is *wordcount.jar* and is saved in the default directory */root*.

5. Create a job.

   i. Upload the *wordcount.jar* file obtained in Step 4 to OSS. For more information, see 上传文件.

   In this example, the file is uploaded to *oss://<yourBucketName>/jars/wordcount.jar*.

ii. Create a MapReduce job in the EMR console. For more information, see Configure a Hadoop MapReduce job.

Job content:

```
ossref://<yourBucketName>/jars/wordcount.jar org.apache.hadoop.examples.EmrWordCount oss://<yourBucketName>/data/WordCount/Input oss://<yourBucketName>/data/WordCount/Output
```

Replace `<yourBucketName>` in the code with the name of the OSS bucket that you use. *oss://<yourBucketName>/data/WordCount/Input* indicates the input path and *oss://<yourBucketName>/data/WordCount/Output* indicates the output path.

iii. On the Edit Job page, click **Run**.

The MapReduce job starts to run in the cluster.

Wordcount2 example

If your project is large, you can use Maven or a similar tool to manage jobs in the project. This example demonstrates how to use Maven to manage a MapReduce job.

1. Install Maven and Java on your on-premises machine.

   In this example, Maven 3.0 and Java 1.8 are used.

2. Run the following command to generate a project framework.

   In this example, the root directory to develop a project is *D:/workspace*.

   ```
   mvn archetype:generate -DgroupId=com.aliyun.emr.hadoop.examples -DartifactId=wordcountv2 -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
   ```

   After you run the command, an empty sample project is automatically generated in the *D:/workspace/wordcountv2* directory, which is consistent with the specified artifactId. The project contains a file named *pom.xml* and a class named App. The package path of the App class is consistent with the specified groupId.

3. Add Hadoop dependencies.

   Use an integrated development environment (IDE) to open the sample project and edit the *pom.xml* file. If Hadoop 2.8.5 is used, add the following content to the file:

   ```
   <dependency>
       <groupId>org.apache.hadoop</groupId>
       <artifactId>hadoop-mapreduce-client-common</artifactId>
       <version>2.8.5</version>
   </dependency>
   <dependency>
       <groupId>org.apache.hadoop</groupId>
       <artifactId>hadoop-common</artifactId>
       <version>2.8.5</version>
   </dependency>
   ```

4. Write code.

   i. Add a new class named *EMapReduceOSSUtil* at the same directory level as the App class in the *com.aliyun.emr.hadoop.examples* package.

```
package com.aliyun.emr.hadoop.examples;
 import org.apache.hadoop.conf.Configuration;
 public class EMapReduceOSSUtil {
   private static String SCHEMA = "oss://";
   private static String EPSEP = ".";
   private static String HTTP_HEADER = "http://";
   /**
    * complete OSS uri
    * convert uri like: oss://bucket/path to oss://bucket.endpoint/path
    * ossref do not need this
    *
    * @param oriUri original OSS uri
    */
   public static String buildOSSCompleteUri(String oriUri, String endpoint) {
     if (endpoint == null) {
       System.err.println("miss endpoint");
       return oriUri;
     }
     int index = oriUri.indexOf(SCHEMA);
     if (index == -1 || index != 0) {
       return oriUri;
     }
     int bucketIndex = index + SCHEMA.length();
     int pathIndex = oriUri.indexOf("/", bucketIndex);
     String bucket = null;
     if (pathIndex == -1) {
       bucket = oriUri.substring(bucketIndex);
     } else {
       bucket = oriUri.substring(bucketIndex, pathIndex);
     }
     StringBuilder retUri = new StringBuilder();
     retUri.append(SCHEMA)
         .append(bucket)
         .append(EPSEP)
         .append(stripHttp(endpoint));
     if (pathIndex > 0) {
       retUri.append(oriUri.substring(pathIndex));
     }
     return retUri.toString();
   }
   public static String buildOSSCompleteUri(String oriUri, Configuration conf) {
     return buildOSSCompleteUri(oriUri, conf.get("fs.oss.endpoint"));
   }
   private static String stripHttp(String endpoint) {
     if (endpoint.startsWith(HTTP_HEADER)) {
       return endpoint.substring(HTTP_HEADER.length());
     }
     return endpoint;
   }
 }
```

ii. Add a new class named *WordCount2.java* at the same directory level as the App class in the *com.aliyun.emr.hadoop.examples* package.

```
package com.aliyun.emr.hadoop.examples;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.net.URI;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.Counter;
import org.apache.hadoop.util.GenericOptionsParser;
import org.apache.hadoop.util.StringUtils;
public class WordCount2 {
  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{
    static enum CountersEnum { INPUT_WORDS }
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    private boolean caseSensitive;
    private Set<String> patternsToSkip = new HashSet<String>();
    private Configuration conf;
    private BufferedReader fis;
    @Override
    public void setup(Context context) throws IOException,
        InterruptedException {
      conf = context.getConfiguration();
      caseSensitive = conf.getBoolean("wordcount.case.sensitive", true);
      if (conf.getBoolean("wordcount.skip.patterns", true)) {
        URI[] patternsURIs = Job.getInstance(conf).getCacheFiles();
        for (URI patternsURI : patternsURIs) {
          Path patternsPath = new Path(patternsURI.getPath());
          String patternsFileName = patternsPath.getName().toString();
          parseSkipFile(patternsFileName);
        }
      }
    }
    private void parseSkipFile(String fileName) {
      try {
        fis = new BufferedReader(new FileReader(fileName));
        String pattern = null;
        while ((pattern = fis.readLine()) != null) {
          patternsToSkip.add(pattern);
        }
      } catch (IOException ioe) {
```

```
          System.err.println("Caught exception while parsing the cached file '"
              + StringUtils.stringifyException(ioe));
        }
      }
      @Override
      public void map(Object key, Text value, Context context
      ) throws IOException, InterruptedException {
        String line = (caseSensitive) ?
            value.toString() : value.toString().toLowerCase();
        for (String pattern : patternsToSkip) {
          line = line.replaceAll(pattern, "");
        }
        StringTokenizer itr = new StringTokenizer(line);
        while (itr.hasMoreTokens()) {
          word.set(itr.nextToken());
          context.write(word, one);
          Counter counter = context.getCounter(CountersEnum.class.getName(),
              CountersEnum.INPUT_WORDS.toString());
          counter.increment(1);
        }
      }
    }
    public static class IntSumReducer
        extends Reducer<Text,IntWritable,Text,IntWritable> {
      private IntWritable result = new IntWritable();
      public void reduce(Text key, Iterable<IntWritable> values,
                         Context context
      ) throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
          sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
      }
    }
    public static void main(String[] args) throws Exception {
      Configuration conf = new Configuration();
      conf.set("fs.oss.accessKeyId", "${accessKeyId}");
      conf.set("fs.oss.accessKeySecret", "${accessKeySecret}");
      conf.set("fs.oss.endpoint","${endpoint}");
      GenericOptionsParser optionParser = new GenericOptionsParser(conf, args);
      String[] remainingArgs = optionParser.getRemainingArgs();
      if (!(remainingArgs.length != 2 || remainingArgs.length != 4)) {
        System.err.println("Usage: wordcount <in> <out> [-skip skipPatternFile]");
        System.exit(2);
      }
      Job job = Job.getInstance(conf, "word count");
      job.setJarByClass(WordCount2.class);
      job.setMapperClass(TokenizerMapper.class);
      job.setCombinerClass(IntSumReducer.class);
      job.setReducerClass(IntSumReducer.class);
      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(IntWritable.class);
      List<String> otherArgs = new ArrayList<String>();
```

```
        List<String> otherArgs = new ArrayList<String>();
        for (int i=0; i < remainingArgs.length; ++i) {
          if ("-skip".equals(remainingArgs[i])) {
            job.addCacheFile(new Path(EMapReduceOSSUtil.buildOSSCompleteUri(remainingArgs[++i
], conf)).toUri());
            job.getConfiguration().setBoolean("wordcount.skip.patterns", true);
          } else {
            otherArgs.add(remainingArgs[i]);
          }
        }
        FileInputFormat.addInputPath(job, new Path(EMapReduceOSSUtil.buildOSSCompleteUri(oth
erArgs.get(0), conf)));
        FileOutputFormat.setOutputPath(job, new Path(EMapReduceOSSUtil.buildOSSCompleteUri(o
therArgs.get(1), conf)));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
      }
    }
```

5. In the root directory of the project, run the following command to compile and package the files of the two new classes:

   ```
   mvn clean package -DskipTests
   ```

   A JAR file named *wordcountv2-1.0-SNAPSHOT.jar* is generated in the *target* directory of the project.

6. Create a job.

   i. Upload the *wordcountv2-1.0-SNAPSHOT.jar* file obtained in Step 5 to OSS. For more information, see 上传文件.

   In this example, the file is uploaded to *oss://<yourBucketName>/jars/wordcountv2-1.0-SNAPSHOT.jar*.

   ii. Download the following files and upload them to your OSS directory:

   - The_Sorrows_of_Young_Werther.txt
   - patterns.txt

   > **Note** *The_Sorrows_of_Young_Werther.txt* is a text file in which the number of occurrences of each word needs to be counted. *patterns.txt* lists the word patterns to be ignored.

   iii. Create a MapReduce job in the EMR console. For more information, see Configure a Hadoop MapReduce job.

   Job content:

   ```
   ossref://<yourBucketName>/jars/wordcountv2-1.0-SNAPSHOT.jar com.aliyun.emr.hadoop.exampl
   es.WordCount2 -D wordcount.case.sensitive=true oss://<yourBucketName>/jars/The_Sorrows_of
   _Young_Werther.txt oss://<yourBucketName>/jars/output -skip oss://<yourBucketName>/jars/patt
   erns.txt
   ```

   Replace `<yourBucketName>` in the code with the name of the OSS bucket that you use. *oss://<yourBucketName>/jars/output* indicates the output path.

iv. On the Edit Job page, click **Run**.

The MapReduce job starts to run in the cluster.

4.3. Create and run a Hive job

This topic describes how to create and run a Hive job in an E-MapReduce (EMR) cluster.

Use Hive to process OSS data

If you want to use Hive to read data from or write data to an Object Storage Service (OSS) bucket, you must first run the following command to create an external table:

```
CREATE EXTERNAL TABLE eusers (
 userid INT)
LOCATION 'oss://emr/users';
```

If the preceding method is not supported or you want to use the AccessKey pair of another Alibaba Cloud account to access OSS data in other locations, you can run the following command:

```
CREATE EXTERNAL TABLE eusers (
 userid INT)
LOCATION 'oss://${AccessKeyId}:${AccessKeySecret}@${bucket}.${endpoint}/users';
```

Parameters in the command:

- `${accessKeyId}` : the AccessKey ID of an Alibaba Cloud account.
- `${accessKeySecret}` : the AccessKey secret that matches the AccessKey ID.
- `${endpoint}` : the network endpoint that is used to access OSS. It depends on the region where your cluster resides. The OSS bucket must be in the region where your cluster resides.

  For more information, see OSS endpoints.

Examples

The following examples show how to create and run a Hive job:

- Example 1
  i. Write the following script, save it as *hiveSample1.sql*, and then upload it to OSS.

  For more information, see 上传文件.

```
USE DEFAULT;
 set hive.input.format=org.apache.hadoop.hive.ql.io.HiveInputFormat;
 set hive.stats.autogather=false;
 DROP TABLE emrusers;
 CREATE EXTERNAL TABLE emrusers (
  userid INT,
  movieid INT,
  rating INT,
  unixtime STRING )
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY '\t'
 STORED AS TEXTFILE
 LOCATION 'oss://${bucket}/yourpath';
 SELECT COUNT(*) FROM emrusers;
 SELECT * from emrusers limit 100;
 SELECT movieid,count(userid) as usercount from emrusers group by movieid order by usercount de
sc limit 50;
```

ii. Prepare test data.

You can download the following test data and upload it to the destination OSS directory.

Test data: Public test data

iii. Create a job.

Create a Hive job in the EMR console. For more information, see Configure a Hive job.

Content of the job:

```
-f ossref://${bucket}/yourpath/hiveSample1.sql
```

In this example, `${bucket}` indicates your OSS bucket, and `yourpath` indicates a path in the bucket. Replace yourpath with the path where the Hive script is stored.

iv. Run the job.

Click **Run** to run the job. You can associate the job with an existing cluster. You can also enable the system to automatically create a cluster and associate the job with the cluster.

- Example 2

Use scan in HiBench as an example.

i. Write the following script, save it as scan.hive, and then upload it to OSS:

```
USE DEFAULT;
 set hive.input.format=org.apache.hadoop.hive.ql.io.HiveInputFormat;
 set mapreduce.job.maps=12;
 set mapreduce.job.reduces=6;
 set hive.stats.autogather=false;
 DROP TABLE uservisits;
 CREATE EXTERNAL TABLE uservisits (sourceIP STRING,destURL STRING,visitDate STRING,adRevenue
DOUBLE,userAgent STRING,countryCode STRING,languageCode STRING,searchWord STRING,duratio
n INT ) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS SEQUENCEFILE LOCATION 'oss:
//${bucket}/sample-data/hive/Scan/Input/uservisits';
```

For example, you can upload the file to *oss://emr/jars/*.

ii. Prepare test data.

You can download the following test data and upload it to the destination OSS directory.

Test data: uservisits.

iii. Create a Hive job in the EMR console. For more information, see Configure a Hive job.

iv. Run the job.

Click **Run** to run the job. You can associate the job with an existing cluster. You can also enable the system to automatically create a cluster and associate the job with the cluster.

4.4. Create and run a Pig job

This topic describes how to create and run a Pig job in an E-MapReduce (EMR) cluster.

Use Pig to access OSS data

When you use Pig to access Object Storage Service (OSS) data, you must specify an OSS path in the following format:

```
oss://${accessKeyId}:${accessKeySecret}@${bucket}.${endpoint}/${path}
```

Parameters:

- `${accessKeyId}` : the AccessKey ID of your Alibaba Cloud account.
- `${accessKeySecret}` : the AccessKey secret that matches the AccessKey ID.
- `${bucket}` : the bucket that matches the AccessKey ID.
- `${endpoint}` : the network endpoint that is used to access OSS. It depends on the region where your cluster resides. The OSS bucket must be in the region where your cluster resides.

  For more information, see OSS endpoints.
- `${path}` : the path of a file in the bucket.

Procedure

Use the *script1-hadoop.pig* file in Pig as an example. Upload tutorial.jar and excite.log.bz2 in Pig to OSS. For example, the upload paths are *oss://emr/jars/tutorial.jar* and *oss://emr/data/excite.log.bz2*.

1. Prepare a script.

   Modify the JAR file path and the input and output paths in the script based on the preceding OSS paths.

   ```
   /*
    * Licensed to the Apache Software Foundation (ASF) under one
   * or more contributor license agreements.  See the NOTICE file
   * distributed with this work for additional information
   * regarding copyright ownership.  The ASF licenses this file
   * to you under the Apache License, Version 2.0 (the
   * "License"); you may not use this file except in compliance
   * with the License.  You may obtain a copy of the License at
   *
   *   http://www.apache.org/licenses/LICENSE-2.0
   *
   * Unless required by applicable law or agreed to in writing, software
   ```

```
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
-- Query Phrase Popularity (Hadoop cluster)
-- This script processes a search query log file from the Excite search engine and finds search phrases th
at occur with particular high frequency during certain times of the day.
-- Register the tutorial JAR file so that the included UDFs can be called in the script.
REGISTER oss://${AccessKeyId}:${AccessKeySecret}@${bucket}.${endpoint}/data/tutorial.jar;
-- Use the PigStorage function to load the excite log file into the raw bag as an array of records.
-- Input: (user,time,query)
raw = LOAD 'oss://${AccessKeyId}:${AccessKeySecret}@${bucket}.${endpoint}/data/excite.log.bz2' USIN
G PigStorage('\t') AS (user, time, query);
-- Call the NonURLDetector UDF to remove records if the query field is empty or a URL.
clean1 = FILTER raw BY org.apache.pig.tutorial.NonURLDetector(query);
-- Call the ToLower UDF to change the query field to lowercase.
clean2 = FOREACH clean1 GENERATE user, time,    org.apache.pig.tutorial.ToLower(query) as query;
-- Because the log file only contains queries for a single day, we are only interested in the hour.
-- The excite query log timestamp format is YYMMDDHHMMSS.
-- Call the ExtractHour UDF to extract the hour (HH) from the time field.
houred = FOREACH clean2 GENERATE user, org.apache.pig.tutorial.ExtractHour(time) as hour, query;
-- Call the NGramGenerator UDF to compose the n-grams of the query.
ngramed1 = FOREACH houred GENERATE user, hour, flatten(org.apache.pig.tutorial.NGramGenerator(q
uery)) as ngram;
-- Use the  DISTINCT command to get the unique n-grams for all records.
ngramed2 = DISTINCT ngramed1;
-- Use the  GROUP command to group records by n-gram and hour.
hour_frequency1 = GROUP ngramed2 BY (ngram, hour);
-- Use the  COUNT function to get the count (occurrences) of each n-gram.
hour_frequency2 = FOREACH hour_frequency1 GENERATE flatten($0), COUNT($1) as count;
-- Use the  GROUP command to group records by n-gram only.
-- Each group now corresponds to a distinct n-gram and has the count for each hour.
uniq_frequency1 = GROUP hour_frequency2 BY group::ngram;
-- For each group, identify the hour in which this n-gram is used with a particularly high frequency.
-- Call the ScoreGenerator UDF to calculate a "popularity" score for the n-gram.
uniq_frequency2 = FOREACH uniq_frequency1 GENERATE flatten($0), flatten(org.apache.pig.tutorial.Sc
oreGenerator($1));
-- Use the  FOREACH-GENERATE command to assign names to the fields.
uniq_frequency3 = FOREACH uniq_frequency2 GENERATE $1 as hour, $0 as ngram, $2 as score, $3 as cou
nt, $4 as mean;
-- Use the  FILTER command to move all records with a score less than or equal to 2.0.
filtered_uniq_frequency = FILTER uniq_frequency3 BY score > 2.0;
-- Use the  ORDER command to sort the remaining records by hour and score.
ordered_uniq_frequency = ORDER filtered_uniq_frequency BY hour, score;
-- Use the  PigStorage function to store the results.
-- Output: (hour, n-gram, score, count, average_counts_among_all_hours)
STORE ordered_uniq_frequency INTO 'oss://${AccessKeyId}:${AccessKeySecret}@${bucket}.${endpoint
}/data/script1-hadoop-results' USING PigStorage();
```

Upload the *script1-hadoop.pig* script to an OSS path, for example, *oss://emr/jars/*.

2. Create a job.

Create a Pig job on the Data Platform tab of the EMR console. For more information, see Configure a Pig job.

Content of the job:

```
-f ossref://emr/jars/script1-hadoop.pig
```

3. Run the job.

   Click **Run** to run the job. You can associate the job with an existing cluster. You can also enable the system to automatically create a cluster and associate the job with the cluster.

4.5. Hadoop Streaming

This topic describes how to use Python to submit a Hadoop Streaming job.

Prerequisites

An E-MapReduce (EMR) Hadoop cluster is created.

For more information about how to create a cluster, see Create a cluster.

Procedure

1. Log on to the Hadoop cluster in SSH mode. For more information, see Log on to a cluster.
2. Create a file named *mapper.py*.
   i. Run the following command to create a file named *mapper.py* and open the file:

   ```
   vim /home/hadoop/mapper.py
   ```

   ii. Press the `I` key to switch to the edit mode.
   iii. Add the following information to the *mapper.py* file:

   ```
   #!/usr/bin/env python
   import sys
   for line in sys.stdin:
     line = line.strip()
     words = line.split()
     for word in words:
       print '%s\t%s' % (word, 1)
   ```

   iv. Press `Esc` to exit the edit mode. Then, enter `:wq` to save and close the file.
3. Create a file named *reducer.py*.
   i. Run the following command to create a file named *reducer.py* and open the file:

   ```
   vim /home/hadoop/reducer.py
   ```

   ii. Press the `I` key to switch to the edit mode.

iii. Add the following information to the *reducer.py* file:

```
#!/usr/bin/env python
from operator import itemgetter
import sys
current_word = None
current_count = 0
word = None
for line in sys.stdin:
  line = line.strip()
  word, count = line.split('\t', 1)
  try:
    count = int(count)
  except ValueError:
    continue
  if current_word == word:
    current_count += count
  else:
    if current_word:
      print '%s\t%s' % (current_word, current_count)
    current_count = count
    current_word = word
if current_word == word:
  print '%s\t%s' % (current_word, current_count)
```

iv. Press `Esc` to exit the edit mode. Then, enter `:wq` to save and close the file.

4. Run the following command to upload the *hosts* file to HDFS:

```
hdfs dfs -put /etc/hosts /tmp/
```

5. Run the following command to submit a Hadoop Streaming job:

```
hadoop jar /usr/lib/hadoop-current/share/hadoop/tools/lib/hadoop-streaming-X.X.X.jar -file /home/hadoop/mapper.py -mapper mapper.py -file /home/hadoop/reducer.py -reducer reducer.py -input /tmp/hosts -output /tmp/output
```

Parameter	Description
input	The input path. In this example, the input path is */tmp/hosts*.
output	The output path. In this example, the output path is */tmp/output*.

> **Note** In *hadoop-streaming-X.X.X.jar*, `X.X.X` indicates the version of the JAR package. The version of the JAR package must be the same as the Hadoop version of your cluster. You can view the version of the JAR package in the */usr/lib/hadoop-current/share/hadoop/tools/lib/* directory.

4.6. Process Table Store data in Hive

This topic describes how to process Table Store data in Hive.

Connect Hive to Table Store

- Prepare a data table

  Create a table named pet and set the name field as the primary key.

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	

- The following example describes how to process Table Store data in Hive.

  i. Command line

  ```
  $ HADOOP_HOME=YourHadoopDir HADOOP_CLASSPATH=emr-tablestore-<version>.jar:tablestore-4.1.0-jar-with-dependencies.jar:joda-time-2.9.4.jar bin/hive
  ```

  ii. Create an external table

  ```
  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");
  ```

  iii. Insert data to the external table

```
INSERT INTO pet VALUES("Fluffy", "Harold", "cat", "f", "1993-02-04", null);
INSERT INTO pet VALUES("Claws", "Gwen", "cat", "m", "1994-03-17", null);
INSERT INTO pet VALUES("Buffy", "Harold", "dog", "f", "1989-05-13", null);
INSERT INTO pet VALUES("Fang", "Benny", "dog", "m", "1990-08-27", null);
INSERT INTO pet VALUES("Bowser", "Diane", "dog", "m", "1979-08-31", "1995-07-29");
INSERT INTO pet VALUES("Chirpy", "Gwen", "bird", "f", "1998-09-11", null);
INSERT INTO pet VALUES("Whistler", "Gwen", "bird", null, "1997-12-09", null);
INSERT INTO pet VALUES("Slim", "Benny", "snake", "m", "1996-04-29", null);
INSERT INTO pet VALUES("Puffball", "Diane", "hamster", "f", "1999-03-30", null);
```

iv. Query data

```
> SELECT * FROM pet;
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
> 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
```

Data type conversion

	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)			
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			
BOOLEAN							Supported		
STRING								Supported	

	TINYINT	SMALLINT	INT	BIGINT	FLOAT	DOUBLE	BOOLEAN	STRING	BINARY
BINARY									Supported

Appendix

For the complete sample code, see

- Process Table Store data in Hive

5.HBase

5.1. Access HBase

This topic describes how to configure an HBase cluster and use the HBase storage service.

Prerequisites

An E-MapReduce (EMR) cluster is created, and the HBase service is added to the cluster. For more information, see Create a cluster.

Configure an HBase cluster

When you create an HBase cluster, you can turn on **Custom Software Settings** in the **Advanced Settings** section of the **Software Settings** step and modify the default HBase configurations. Example:

```
{
 "configurations": [
  {
   "classification": "hbase-site",
   "properties": {
    "hbase.hregion.memstore.flush.size": "268435456",
    "hbase.regionserver.global.memstore.size": "0.5",
    "hbase.regionserver.global.memstore.lowerLimit": "0.6"
   }
  }
 ]
}
```

The following table lists the default HBase configurations.

Key	Value
zookeeper.session.timeout	180000
hbase.regionserver.global.memstore.size	0.35
hbase.regionserver.global.memstore.lowerLimit	0.3
hbase.hregion.memstore.flush.size	128MB

Access HBase

1.
2. Run the following command to access HBase Shell:

   ```
   hbase shell
   ```

   The following information is returned:

```
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/apps/ecm/service/hbase/1.4.9-1.0.0/package/hbase-1.4.9-1.0.0/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/apps/ecm/service/hadoop/2.8.5-1.5.3/package/hadoop-2.8.5-1.5.3/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
Version 1.4.9, r8214a16c5d80f077abf1aa01bb312851511a2b15, Thu Jan 31 20:35:22 CST 2019
hbase(main):001:0>
```

> **Note** If you use an instance created in the ECS console, see Use HBase Shell.

Examples

- Use Spark to access HBase

  For more information, see spark-hbase-connector.

- Use Hadoop to access HBase

  For more information, see HBase MapReduce Examples.

- Use Hive to access HBase

  i. Log on to the master node of a Hive cluster and add the following information to the *hosts* file.

  ```
  $zk_ip emr-cluster // $zk_ip indicates the IP address of the ZooKeeper node in the HBase cluster.
  ```

  ii. For more information about how to perform Hive-related operations, see Hive HBase Integration.

5.2. Back up an HBase cluster

This topic describes how to back up an E-MapReduce (EMR) HBase cluster.

Prerequisites

Two Hadoop clusters are created, and the HBase and ZooKeeper services are added to the clusters. For more information, see Create a cluster.

Procedure

1.
2. Create a table and add data to the table.

   i. Enable HBase Shell.

   ```
   hbase shell
   ```

   ii. Create a table.

   ```
   create 'test','cf'
   ```

iii. Add data to the table.

```
put 'test','a','cf:c1',1
put 'test','a','cf:c2',2
put 'test','b','cf:c1',3
put 'test','b','cf:c2',4
put 'test','c','cf:c1',5
put 'test','c','cf:c2',6
```

iv. Exit HBase Shell.

```
exit
```

3. Create a snapshot and query snapshot information.

i. Create a snapshot.

```
hbase snapshot create -n test_snapshot -t test
```

ii. Enable HBase Shell.

```
hbase shell
```

iii. Query snapshot information.

```
list_snapshots
```

The following information is returned:

```
SNAPSHOT                        TABLE + CREATION TIME
 test_snapshot                   test (Tue Aug 18 14:35:28 +0800 2020)
1 row(s) in 0.2450 seconds
=> ["test_snapshot"]
```

iv. Exit HBase Shell.

```
exit
```

4. Export the created snapshot to OSS.

```
hbase org.apache.hadoop.hbase.snapshot.ExportSnapshot -snapshot test_snapshot -copy-to oss://$accessKeyId:$accessKeySecret@$bucket.oss-cn-hangzhou-internal.aliyuncs.com/hbase/snapshot/test
```

> **Note** Use the internal endpoint to access OSS.

5. Log on to the other cluster by using SSH.
6. Export the snapshot from OSS.

```
hbase org.apache.hadoop.hbase.snapshot.ExportSnapshot -snapshot test_snapshot -copy-from oss://$accessKeyId:$accessKeySecret@$bucket.oss-cn-hangzhou-internal.aliyuncs.com/hbase/snapshot/test -copy-to /hbase/
```

7. Restore data from the snapshot and view data in the restored table.

i. Enable HBase Shell.

```
hbase shell
```

ii. Restore data from the snapshot.

```
restore_snapshot 'test_snapshot'
```

iii. View data in the restored table.

```
scan 'test'
```

The following information is returned:

```
ROW             COLUMN+CELL
 a              column=cf:c1, timestamp=1472992081375, value=1
 a              column=cf:c2, timestamp=1472992090434, value=2
 b              column=cf:c1, timestamp=1472992104339, value=3
 b              column=cf:c2, timestamp=1472992099611, value=4
 c              column=cf:c1, timestamp=1472992112657, value=5
 c              column=cf:c2, timestamp=1472992118964, value=6
3 row(s) in 0.0540 seconds
```

8. Create a table based on the snapshot and view data in the table.

i. Create a table based on the snapshot.

```
clone_snapshot 'test_snapshot','test_2'
```

ii. View data in the table.

```
scan 'test_2'
```

The following information is returned:

```
ROW             COLUMN+CELL
 a              column=cf:c1, timestamp=1472992081375, value=1
 a              column=cf:c2, timestamp=1472992090434, value=2
 b              column=cf:c1, timestamp=1472992104339, value=3
 b              column=cf:c2, timestamp=1472992099611, value=4
 c              column=cf:c1, timestamp=1472992112657, value=5
 c              column=cf:c2, timestamp=1472992118964, value=6
3 row(s) in 0.0540 seconds
```