Huawei Certification Big Data Course

HCIA-Big Data

Lab Guide

ISSUE: 3.5

HUAWEI TECHNOLOGIES CO., LTD

Trademarks and Permissions

 and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.

All other trademarks or registered trademarks mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change due to version upgrade or other reasons. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Technologies Co., Ltd.

Address: Huawei Industrial Base Bantian, Longgang Shenzhen 518129 People's Republic of China

Website: https://e.huawei.com

Huawei Certification System

The Huawei certification system is a platform for shared growth, part of a thriving partner ecosystem. There are two types of certification: one for ICT architectures and applications, and one for cloud services and platforms.

Huawei offers three levels of certification: Huawei Certified ICT Associate (HCIA), Huawei Certified ICT Professional (HCIP), and Huawei Certified ICT Expert (HCIE).

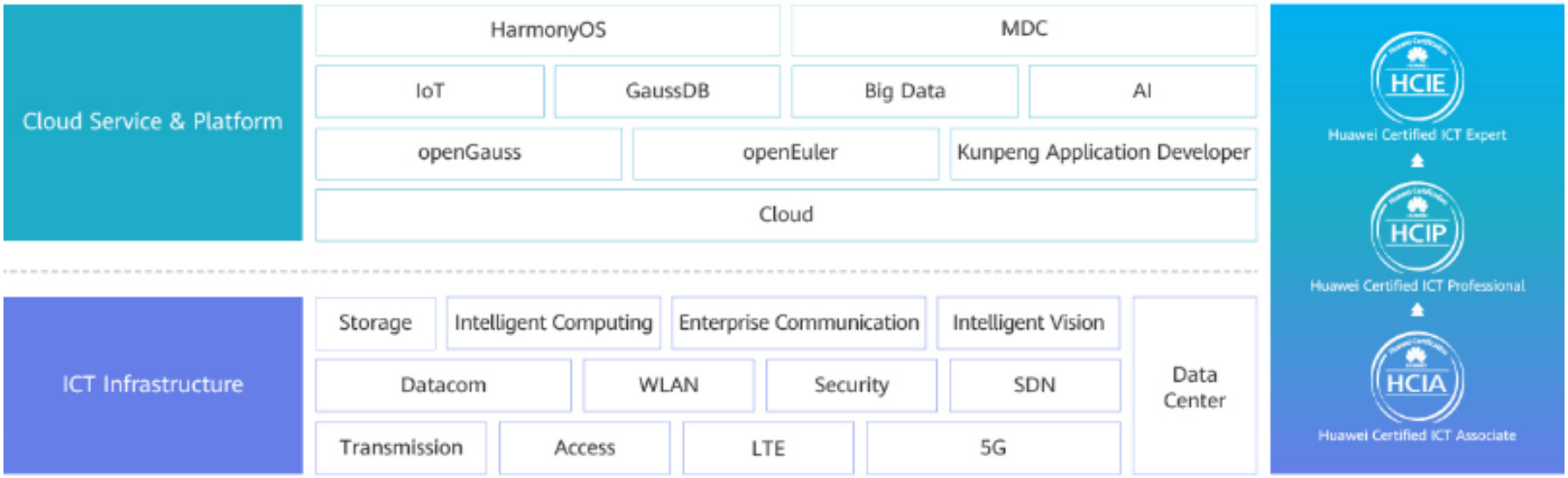
Huawei certification courses cover the entire ICT domain, with a focus on how today's architecture generates cloud-pipe-device synergy. The courses present the latest developments of all essential ICT aspects to foster a thriving ICT talent ecosystem for the digital age.

HCIA-Big Data V3.5 certification aims to train and certify engineers who are capable of using Huawei MRS big data platform.

Passing the HCIA-Big Data V3.5 certification means that you already master the technical principles and architectures of big data components, including HDFS, Hive, HBase, ClickHouse, MapReduce, Spark, Flink, Flume, and Kafka, are capable of using Huawei big data platform MRS, and are able to operate and develop Huawei MRS.

The Huawei certification system helps you embrace the up-to-date industry knowledge and trends, opens up a new horizon for you, and assists you during your pursuit for a new self.

Huawei Certification



About This Document

Introduction

This document uses Huawei Cloud MapReduce Service (MRS) as the exercise environment to guide students through related tasks and help them understand how to use big data components of MRS.

Exercises

This document consists of the following 10 exercises illustrating the usage of important big data components:

- HDFS Practices
- HBase Columnar Database Practices
- Hive Data Warehouse Practices
- ClickHouse Online Analysis Database Practices
- MapReduce Data Processing Practices
- Spark In-Memory Computing Practices
- Flink Real-Time Processing System Practices
- Flume Data Collection Practices
- Kafka Message Subscription Practices
- Cluster Comprehensive Exercise

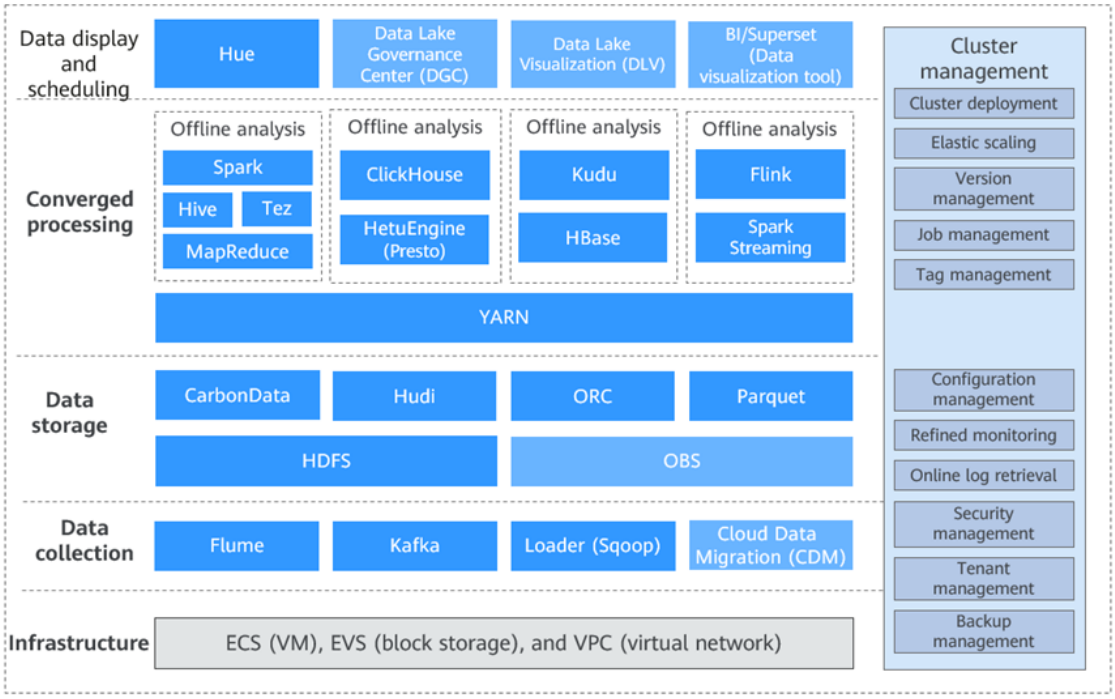
Preparations

- A Huawei Cloud account and real-name authentication are required.
- It is recommended that each trainee use one exercise environment.

Reference Document

To obtain the MRS documentation, visit https://support.huaweicloud.com/intl/en-us/mrs/index.html.

MRS Architecture



Contents

1 Distributed File System in Hadoop

1.1 About This Exercise

1.1.1 Overview

HDFS is the basis of other big data components, and stores Hive data, MapReduce and Spark computing data, and regions of HBase. On the HDFS shell client, you can perform various operations, such as uploading, downloading, and deleting data, and managing file systems. Mastering HDFS is fundamental to better understanding and usage of big data.

1.1.2 Objectives

Master common HDFS operations and HDFS file system management operations.

1.2 Tasks

1.2.1 Task 1: Performing Common HDFS Operations

Before operating MRS components, run the following command to set environment variables:

```
source   /opt/client/bigdata_env
```

Step 1 Run the **help** command.

This command is used to view the help document of a certain command.

```
hdfs   dfs   -help
```

```
[root@node-master1JuMN ~]# hdfs  dfs  -help
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/li
b/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/c
ommon/lib/slf4j-log4j12-1.7.30.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]
Usage: hadoop fs [generic options]
        [-appendToFile <localsrc> ... <dst>]
        [-cat [-ignoreCrc] <src> ...]
        [-checksum <src> ...]
        [-chgrp [-R] GROUP PATH...]
        [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
        [-chown [-R] [OWNER][:[GROUP]] PATH...]
        [-copyFromLocal [-f] [-p] [-l] [-d] [-t <thread count>] <localsrc> ... <
dst>]
```

Figure 1-1

Run the following command to check how to use the **ls** command:

```
hdfs  dfs  -help  ls
```

```
[root@node-master1JuMN ~]# hdfs  dfs  -help  ls
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/li
b/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/c
ommon/lib/slf4j-log4j12-1.7.30.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]
-ls [-C] [-d] [-h] [-q] [-R] [-t] [-S] [-r] [-u] [-e] [<path> ...] :
  List the contents that match the specified file pattern. If path is not
  specified, the contents of /user/<currentUser> will be listed. For a directory
 a
  list of its direct children is returned (unless -d option is specified).

  Directory entries are of the form:
        permissions - userId groupId sizeOfDirectory(in bytes)
  modificationDate(yyyy-MM-dd HH:mm) directoryName

  and file entries are of the form:
        permissions numberOfReplicas userId groupId sizeOfFile(in bytes)
  modificationDate(yyyy-MM-dd HH:mm) fileName

    -C  Display the paths of files and directories only.
    -d  Directories are listed as plain files.
    -h  Formats the sizes of files in a human-readable fashion
        rather than a number of bytes.
    -q  Print ? instead of non-printable characters.
    -R  Recursively list the contents of directories.
    -t  Sort files by modification time (most recent first).
    -S  Sort files by size.
    -r  Reverse the order of the sort.
    -u  Use time of last access instead of modification for
        display and sorting.
    -e  Display the erasure coding policy of files and directories.
[root@node-master1JuMN ~]#
```

Figure 1-2

Step 2 Run the **ls** command.

This command is used to display the directory information.

Run the following command to display the information in the root directory:

```
hdfs  dfs  -ls  /
```

```
[root@node-master1JuMN ~]# hdfs  dfs  -ls  /
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 14 items
drwxrwxrwx   - hdfs     hadoop               0 [redacted] /app-logs
drwxrwxrwx   - hive     hive                 0 [redacted] /apps
drwxr-xr-x   - hdfs     hadoop               0 [redacted] /datasets
drwxr-xr-x   - hdfs     hadoop               0 [redacted] /datastore
drwxrwx---   - flink    hadoop               0 [redacted] /flink
drwxr-x---   - flume    hadoop               0 [redacted] /flume
drwxrwx--T   - hbase    hadoop               0 [redacted] /hbase
drwxrwxrwx   - mapred   hadoop               0 [redacted] /mr-history
drwxrwxrwx   - hdfs     hadoop               0 [redacted] /mrs
drwxrwxrwt   - spark2x  hadoop               0 [redacted] /spark2xJobHistory2x
drwxrwxrwt   - spark    hadoop               0 [redacted] /sparkJobHistory
drwx--x--x   - admin    supergroup           0 [redacted] /tenant
drwxrwxrwx   - hdfs     hadoop               0 [redacted] /tmp
drwxrwxrwx   - hdfs     hadoop               0 [redacted] /user
[root@node-master1JuMN ~]#
```

Figure 1-3

Step 3 Run the **mkdir** command.

This command is used to create directories in HDFS.

To create the **stu01** folder in the **user** folder of the **root** directory, view the content in the **user** folder, create the folder, and then run the **ls** command. The **stu01** folder is displayed.

Run the following command to create the **stu01** folder:

```
hdfs  dfs  -mkdir  /user/stu01
```

```
[root@node-master1JuMN ~]# hdfs  dfs  -mkdir  /user/stu01
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.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]
[root@node-master1JuMN ~]# hdfs  dfs  -ls  /user
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 8 items
drwxrwxrwx   - hive   hive            0 [redacted] /user/hive
drwxrwxrwx   - loader hadoop          0 [redacted] /user/loader
drwxr-xr-x   - mapred hadoop          0 [redacted] /user/mapred
drwx------   - omm    hadoop          0 [redacted] /user/omm
drwxr-xr-x   - oozie  hadoop          0 [redacted] /user/oozie
drwxrwxrwx   - omm    hadoop          0 [redacted] /user/spark2x
drwxr-xr-x   - root   hadoop          0 [redacted] /user/stu01
drwxr-xr-x   - omm    hadoop          0 [redacted] /user/yarn
[root@node-master1JuMN ~]#
```

Figure 1-4

Step 4 Run the **put** command.

This command is used to upload a file in the Linux system to a specified directory in HDFS. Before running this command, run the following **vi** command to edit a file in a local Linux PC:

```
vi stu01.txt
```

```
[root@node-master1JuMN ~]# vi stu01.txt
[root@node-master1JuMN ~]#
```

Figure 1-5

Press **i** to enter the editing mode, enter the content, and press **Esc**. Then, press **Shift** and a colon (:), enter **wq** to save the settings, and exit. The following is a file content example:

```
hello world
hello Hadoop
hello MRS
```

The following figure shows the command output.

```
[root@node-master1JuMN ~]# cat stu01.txt
hello world
hello hadoop
hello MRS
[root@node-master1JuMN ~]#
```

Figure 1-6

Run the following command to upload the file:

```
hdfs  dfs  -put  stu01.txt  /user/stu01/
```

Run the **ls** command to check whether the **stu01.txt** file has been uploaded to the **/user/stu01** directory.

```
[root@node-master1JuMN ~]# hdfs  dfs  -put  stu01.txt  /user/stu01/
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.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]
[root@node-master1JuMN ~]# hdfs  dfs  -ls  /user/stu01
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 1 items
-rw-r--r--   2 root hadoop         35 [redacted] /user/stu01/stu01.txt
```

Figure 1-7

Step 5 Run the **cat** command.

This command is used to display the file content.

```
hdfs  dfs  -cat /user/stu01/stu01.txt
```

```
[root@node-master1JuMN ~]# hdfs  dfs  -cat /user/stu01/stu01.txt
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.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]
hello world
hello hadoop
hello MRS
[root@node-master1JuMN ~]#
```

Figure 1-8

Step 6 Run the **text** command.

This command is used to print the content of a file as characters.

```
hdfs  dfs  -text /user/stu01/stu01.txt
```

```
[root@node-master1JuMN ~]# hdfs  dfs  -text /user/stu01/stu01.txt
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.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]
hello world
hello hadoop
hello MRS
[root@node-master1JuMN ~]#
```

Figure 1-9

Step 7 Run the **moveFromLocal** command.

This command is used to cut and paste data from the local PC to HDFS.

Run the following **vi** command to create a data file **stu02.txt** on the local Linux PC:

```
vi stu02.txt
cat stu02.txt
```

```
[root@node-master1JuMN ~]# vi stu02.txt
[root@node-master1JuMN ~]# cat stu02.txt
hello HDFS
[root@node-master1JuMN ~]#
```

Figure 1-10

Cut the **stu02.txt** file and paste it to the **stu01** directory and run the following command:

```
hdfs  dfs  -moveFromLocal stu02.txt  /user/stu01/
```

```
[root@node-master1JuMN ~]# hdfs  dfs  -moveFromLocal stu02.txt  /user/stu01/
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.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]
[root@node-master1JuMN ~]# hdfs  dfs  -ls  /user/stu01
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 2 items
-rw-r--r--   2 root hadoop         35 [redacted] /user/stu01/stu01.txt
-rw-r--r--   2 root hadoop         11 [redacted] /user/stu01/stu02.txt
```

Figure 1-11

From the output, you can see the **stu02.txt** file has been uploaded to the **/user/stu01** directory. Run the **ls** command to check the Linux local PC. The **stu02.txt** file does not

exist, indicating that the file is cut and pasted to HDFS. However, if you run the **put** command, the local file is only copied to HDFS and still exists on the Linux PC.

Step 8 Run the **appendToFile** command.

This command is used to add a file to the end of an existing file.

Run the **vi** command to edit data file **stu03.txt** on the local Linux PC. The file content is as follows:

```
vi stu03.txt
cat stu03.txt
```

```
[root@node-master1JuMN ~]# vi stu03.txt
[root@node-master1JuMN ~]# cat stu03.txt
Hello
huawei
[root@node-master1JuMN ~]#
```

Figure 1-12

Run the following command:

```
hdfs dfs -appendToFile stu03.txt /user/stu01/stu02.txt
```

Add the content of the **stu03.txt** file to the **stu02.txt** file in HDFS.

Run the **cat** command to view the result.

```
[root@node-master1JuMN ~]# hdfs dfs -appendToFile stu03.txt /user/stu01/stu02.t
xt
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.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]
[root@node-master1JuMN ~]# hdfs dfs -cat /user/stu01/stu02.txt
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.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]
hello HDFS
Hello
huawei
[root@node-master1JuMN ~]#
```

Figure 1-13

Step 9 Run the **cp** command.

This command is used to copy a file from one HDFS directory to another HDFS directory.

Run the **vi** command to edit the **stu04.txt** file on the local Linux PC, and run the **put** command to upload the file to the HDFS root directory:

```
vi stu04.txt
cat stu04.txt
hdfs dfs -put stu04.txt /user/stu01
```

The following figure shows the command output.

```
[root@node-master1JuMN ~]#  vi stu04.txt
[root@node-master1JuMN ~]# cat stu04.txt
hello
this is stu04.txt
[root@node-master1JuMN ~]# hdfs dfs -put stu04.txt /user/stu01
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.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]
[root@node-master1JuMN ~]#
```

Figure 1-14

Run the **cp** command:

```
hdfs  dfs  -cp  /user/stu01/stu04.txt  /user/stu02/
```

```
[root@node-master1JuMN ~]# hdfs  dfs  -mkdir  /user/stu02
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.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]
[root@node-master1JuMN ~]# hdfs  dfs  -cp  /user/stu01/stu04.txt  /user/stu02/
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.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]
[root@node-master1JuMN ~]# hdfs  dfs  -ls  /user/stu02
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 1 items
-rw-r--r--   2 root hadoop         24 [redacted] /user/stu02/stu04.txt
[root@node-master1JuMN ~]# hdfs  dfs  -ls  /user/stu01
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 3 items
-rw-r--r--   2 root hadoop         35 [redacted] /user/stu01/stu01.txt
-rw-r--r--   2 root hadoop         24 [redacted] /user/stu01/stu02.txt
-rw-r--r--   2 root hadoop         24 [redacted] /user/stu01/stu04.txt
```

Figure 1-15

From the output, you can see that the **stu04.txt** file exists in the **/user/stu02** directory and the **/user/stu01** directory.

Step 10 Run the **mv** command.

This command is used to move files in the HDFS directory.

Run the **vi** command on the local Linux PC to edit the **stu05.txt** file, and run the **put** command to upload the file to the **/user/stu01** directory of HDFS:

```
vi stu05.txt
cat stu05.txt
hdfs dfs -put stu05.txt /user/stu01
```

The following figure shows the command output.

```
[root@node-master1JuMN ~]# vi stu05.txt
[root@node-master1JuMN ~]# cat stu05.txt
this is stu05.txt
[root@node-master1JuMN ~]# hdfs dfs -put stu05.txt /user/stu01
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.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]
[root@node-master1JuMN ~]#
```

Figure 1-16

Run the **mv** command.

```
hdfs dfs -mv /user/stu01/stu05.txt /user/stu02/
```

```
[root@node-master1JuMN ~]# hdfs dfs -ls /user/stu01
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 3 items
-rw-r--r--   2 root hadoop         35 [redacted] /user/stu01/stu01.txt
-rw-r--r--   2 root hadoop         24 [redacted] /user/stu01/stu02.txt
-rw-r--r--   2 root hadoop         24 [redacted] /user/stu01/stu04.txt
[root@node-master1JuMN ~]# hdfs dfs -ls /user/stu02
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 2 items
-rw-r--r--   2 root hadoop         24 [redacted] /user/stu02/stu04.txt
-rw-r--r--   2 root hadoop         18 [redacted] /user/stu02/stu05.txt
```

Figure 1-17

From the output, you can see that the **stu05.txt** file exists in the **/user/stu02** directory, but the file does not exist in the **/user/stu01** directory.

Step 11 Run the **get** command.

Similar to **copyToLocal**, this command is used to download files from HDFS to a local PC.

Run the following command to view the original **stu05.txt** file:

```
[root@node-master1JuMN ~]# ls
env_file  stu01.txt  stu03.txt  stu04.txt  stu05.txt
[root@node-master1JuMN ~]# rm -rf stu05.txt
[root@node-master1JuMN ~]# ls
env_file  stu01.txt  stu03.txt  stu04.txt
[root@node-master1JuMN ~]#
```

Figure 1-18

Run the **copyToLocal** command:

```
hdfs  dfs  -copyToLocal  /user/stu02/stu05.txt  .
```

```
[root@node-master1JuMN ~]# hdfs  dfs  -copyToLocal  /user/stu02/stu05.txt
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.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]
[root@node-master1JuMN ~]# ls
env_file  stu01.txt  stu03.txt  stu04.txt  stu05.txt
[root@node-master1JuMN ~]#
```

Figure 1-19

The **stu05.txt** file exists on the local Linux PC. Note that the period (.) at the end of the HDFS command indicates the current directory. You can specify another directory to save the file.

Step 12 Run the **getmerge** command.

This command is used to download a combination of multiple files.

Run the **ls** command to view the files in the **/user/stu01/** directory.

```
[root@node-master1JuMN ~]# hdfs dfs -ls /user/stu01
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 3 items
-rw-r--r--   2 root hadoop         35 [redacted] /user/stu01/stu01.txt
-rw-r--r--   2 root hadoop         24 [redacted] /user/stu01/stu02.txt
-rw-r--r--   2 root hadoop         24 [redacted] /user/stu01/stu04.txt
```

Figure 1-20

Run the following command:

```
hdfs dfs -getmerge /user/stu01/* ./merge.txt
```

```
[root@node-master1JuMN ~]# hdfs dfs -getmerge /user/stu01/* ./merge.txt
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.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]
[root@node-master1JuMN ~]# ls
env_file  merge.txt  stu01.txt  stu03.txt  stu04.txt  stu05.txt
[root@node-master1JuMN ~]# cat merge.txt
hello world
hello hadoop
hello MRS
hello HDFS
Hello
huawei
hello
this is stu04.txt
[root@node-master1JuMN ~]#
```

Figure 1-21

The **merge.txt** file is generated in the current directory. The content in the file is the combination of the files in the **/user/stu01/** directory.

Step 13 Run the **rm** command.

This command is used to delete an HDFS file or folder.

```
hdfs dfs -rm /user/stu02/stu05.txt
```

```
[root@node-master1JuMN ~]# hdfs dfs -ls /user/stu02
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 2 items
-rw-r--r--   2 root hadoop         24 [redacted] /user/stu02/stu04.txt
-rw-r--r--   2 root hadoop         18 [redacted] /user/stu02/stu05.txt
[root@node-master1JuMN ~]# hdfs dfs -rm /user/stu02/stu05.txt
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.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]
2022-08-24 11:28:36,902 INFO fs.TrashPolicyDefault: Moved: 'hdfs://hacluster/user/s
tu02/stu05.txt' to trash at: hdfs://hacluster/user/root/.Trash/Current/user/stu02/s
tu05.txt
[root@node-master1JuMN ~]# hdfs dfs -ls /user/stu02
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 1 items
-rw-r--r--   2 root hadoop         24 [redacted] /user/stu02/stu04.txt
```

Figure 1-22

The **stu05.txt** file does not exist in the **/user/stu02/** directory.

Step 14 Run the **df** command.

This command is used to collect information on the available space of a file system.

```
hdfs  dfs  -df  -h  /
```

```
[root@node-master1JuMN ~]# hdfs  dfs  -df  -h  /
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.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]
Filesystem         Size   Used  Available  Use%
hdfs://hacluster  1.6 T  6.4 G      1.5 T    0%
```

Figure 1-23

Step 15 Run the **du** command.

This command is used to calculate the folder size.

```
hdfs  dfs  -du  -s  -h  /user/stu01
```

```
[root@node-master1JuMN ~]# hdfs dfs -ls /user/stu01
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 3 items
-rw-r--r--   2 root hadoop         35 [redacted] /user/stu01/stu01.txt
-rw-r--r--   2 root hadoop         24 [redacted] /user/stu01/stu02.txt
-rw-r--r--   2 root hadoop         24 [redacted] /user/stu01/stu04.txt
[root@node-master1JuMN ~]# hdfs  dfs  -du  -s  -h  /user/stu01
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.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]
83  166  /user/stu01
```

Figure 1-24

Step 16 Run the **count** command.

This command is used to count the number of file nodes in a specified directory.

```
hdfs  dfs  -count -v /user/stu01
```

```
[root@node-master1JuMN ~]# hdfs  dfs  -du  -s  -h  /user/stu01
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.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]
83  166  /user/stu01
[root@node-master1JuMN ~]# hdfs  dfs  -count -v /user/stu01
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.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]
   DIR_COUNT   FILE_COUNT       CONTENT_SIZE PATHNAME
           1            3                 83 /user/stu01
[root@node-master1JuMN ~]#
```

Figure 1-25

1.2.2 Task 2: Using the Recycle Bin

Files may be deleted by mistake in daily work. In this case, you can find deleted files in the recycle bin of HDFS. By default, deleted files are stored in the recycle bin for seven days. For example, if the **stu05.txt** file in the **/user/stu02/** directory is deleted by mistake, it is moved to the recycle bin.

Run the following command:

```
hdfs dfs -ls /user/root/.Trash/Current/user/stu02/
```

You can view the **stu05.txt** file in the recycle bin.

```
[root@node-master1JuMN ~]# hdfs dfs -ls /user/root/.Trash/Current/user/stu02/
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 1 items
-rw-r--r--   2 root hadoop         18 [redacted] /user/root/.Trash/Current/us
er/stu02/stu05.txt
```

Figure 1-26

Note that deleted data is retained for seven days by default.

Run the following **mv** command to move the file back to the **/user/stu02/** directory:

```
hdfs dfs -mv /user/root/.Trash/Current/user/stu02stu05.txt   /user/stu02
```

```
[root@node-master1JuMN ~]# hdfs dfs -mv /user/root/.Trash/Current/user/stu02/stu05.
txt  /user/stu02
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.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]
[root@node-master1JuMN ~]# hdfs dfs -ls /user/stu02
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/s
lf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/comm
on/lib/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 2 items
-rw-r--r--   2 root hadoop         24 [redacted] /user/stu02/stu04.txt
-rw-r--r--   2 root hadoop         18 [redacted] /user/stu02/stu05.txt
```

Figure 1-27

1.3 Exercise Summary

1.3.1 Quiz

Which nodes does HDFS have in a basic Hadoop cluster?

1.3.2 Summary

This exercise mainly describes common operations on HDFS. After completing this exercise, you will be able to perform common HDFS operations.

2 HBase Columnar Database Practices

2.1 About This Exercise

2.1.1 Overview

HBase is an important big data component and is the most commonly used NoSQL database in the industry. Banks can store new customer information in HBase and update or delete out-of-date data in HBase.

2.1.2 Objectives

Master common HBase operations, region operations, and filter usage.

2.2 Tasks

2.2.1 Task 1: Performing Common HBase Operations

2.2.1.1 Accessing the Shell Client

Run the following command to set environment variables:

```
source   /opt/client/bigdata_env
```

Run the following command to go to the HBase Shell client:

```
hbase   shell
```

```
[root@node-masterlnFFO ~]# source  /opt/client/bigdata_env
[root@node-masterlnFFO ~]# hbase  shell
```

Figure 2-1

2.2.1.2 Creating Common Tables

Run the following command to create a table:

```
create   'cx_table_stu01' , 'cf1'
```

```
hbase:001:0>
hbase:002:0> create  'cx_table_stu01' , 'cf1'
[illegible],203 INFO  [main] client.HBaseAdmin: Operation: CREATE, Table Name: defau
lt:cx_table_stu01, procId: 18 completed
Created table cx_table_stu01
Took 1.9559 seconds
=> Hbase::Table - cx_table_stu01
hbase:003:0>
```

Figure 2-2

Run the following command to display all tables:

```
list
```

```
hbase:004:0> list
TABLE
cx_table_stu01
1 row(s)
Took 0.0309 seconds
=> ["cx_table_stu01"]
hbase:005:0>
```

Figure 2-3

2.2.1.3 Adding Data

Run the following commands to add table data:

```
put  'cx_table_stu01','20200001','cf1:name','tom'
put  'cx_table_stu01','20200001','cf1:gender','male'
put  'cx_table_stu01','20200001', 'cf1:age','20'
put  'cx_table_stu01','20200002', 'cf1:name','hanmeimei'
put  'cx_table_stu01','20200002', 'cf1:gender','female'
put  'cx_table_stu01','20200002', 'cf1:age','19'
```

```
hbase:009:0> put  'cx_table_stu01','20200002', 'cf1:name','hanmeimei'
Took 0.0089 seconds
hbase:010:0> put  'cx_table_stu01','20200002', 'cf1:gender','female'
Took 0.0137 seconds
hbase:011:0> put  'cx_table_stu01','20200002', 'cf1:age','19'
Took 0.0107 seconds
hbase:012:0> scan 'cx_table_stu01'
ROW                    COLUMN+CELL
 20200001              column=cf1:age, timestamp=[illegible]T17:43:42.127, value=20
 20200001              column=cf1:gender, timestamp=[illegible]T17:43:32.247, value=male
 20200001              column=cf1:name, timestamp=[illegible]T17:42:56.023, value=tom
 20200002              column=cf1:age, timestamp=[illegible]T17:44:22.988, value=19
 20200002              column=cf1:gender, timestamp=[illegible]T17:44:14.404, value=female
 20200002              column=cf1:name, timestamp=[illegible]T17:44:04.117, value=hanmeimei
2 row(s)
Took 0.0324 seconds
hbase:013:0>
```

Figure 2-4

2.2.1.4 Querying Data in Scan Mode

Run the following SCAN commands to query table data:

```
scan   'cx_table_stu01',{COLUMNS=>'cf1'}            #Query only the data in the cf1 column family.
scan   'cx_table_stu01',{COLUMNS=>'cf1:name'}    #Query only the name information in the cf1 column family.
```

```
hbase:013:0> scan  'cx_table_stu01',{COLUMNS=>'cf1'}
ROW                     COLUMN+CELL
 20200001               column=cf1:age, timestamp=[redacted]T17:43:42.127, value=20
 20200001               column=cf1:gender, timestamp=[redacted]T17:43:32.247, value=male
 20200001               column=cf1:name, timestamp=[redacted]T17:42:56.023, value=tom
 20200002               column=cf1:age, timestamp=[redacted]T17:44:22.988, value=19
 20200002               column=cf1:gender, timestamp=[redacted]T17:44:14.404, value=female
 20200002               column=cf1:name, timestamp=[redacted]T17:44:04.117, value=hanmeimei
2 row(s)
Took 0.0125 seconds
hbase:014:0> scan  'cx_table_stu01',{COLUMNS=>'cf1:name'}
ROW                     COLUMN+CELL
 20200001               column=cf1:name, timestamp=[redacted]T17:42:56.023, value=tom
 20200002               column=cf1:name, timestamp=[redacted]T17:44:04.117, value=hanmeimei
2 row(s)
Took 0.0062 seconds
```

Figure 2-5

2.2.1.5 Querying Data Using the GET Commands

Run the following GET commands to query data based on the rowkey:

```
get   'cx_table_stu01','20200001'
get   'cx_table_stu01','20200001','cf1:name'
```

```
hbase:015:0> get  'cx_table_stu01','20200001'
COLUMN                  CELL
 cf1:age                timestamp=[redacted]T17:43:42.127, value=20
 cf1:gender             timestamp=[redacted]T17:43:32.247, value=male
 cf1:name               timestamp=[redacted]T17:42:56.023, value=tom
1 row(s)
Took 0.0307 seconds
hbase:016:0> get  'cx_table_stu01','20200001','cf1:name'
COLUMN                  CELL
 cf1:name               timestamp=[redacted]T17:42:56.023, value=tom
1 row(s)
Took 0.0054 seconds
hbase:017:0>
```

Figure 2-6

2.2.1.6 Querying Data by Specified Criteria

Run the following commands:

```
scan 'cx_table_stu01',{STARTROW=>'20200001','LIMIT'=>2,STOPROW=>'20200002'}
scan 'cx_table_stu01',{STARTROW=>'20200001','LIMIT'=>2,COLUMNS=>'cf1:name'}
```

```
hbase:017:0> scan 'cx_table_stu01',{STARTROW=>'20200001','LIMIT'=>2,STOPROW=>'20200002'}
ROW                    COLUMN+CELL
 20200001              column=cf1:age, timestamp=[redacted]T17:43:42.127, value=20
 20200001              column=cf1:gender, timestamp=[redacted]T17:43:32.247, value=male
 20200001              column=cf1:name, timestamp=[redacted]T17:42:56.023, value=tom
1 row(s)
Took 0.0055 seconds
hbase:018:0> scan 'cx_table_stu01',{STARTROW=>'20200001','LIMIT'=>2,COLUMNS=>'cf1:name'}
ROW                    COLUMN+CELL
 20200001              column=cf1:name, timestamp=[redacted]T17:42:56.023, value=tom
 20200002              column=cf1:name, timestamp=[redacted]T17:44:04.117, value=hanmeime
                       i
2 row(s)
Took 0.0070 seconds
```

Figure 2-7

Note: In addition to the **COLUMNS** modifier, HBase supports **Limit** (limiting the number of rows in the query result) and **STARTROW** (start row, by which the system locates the region and then scans the region backwards.), **STOPROW** (end row), **TIMERANGE** (timestamp range), **VERSIONS** (number of versions), and **FILTER** (filtered rows based on conditions).

2.2.1.7 Querying Multiversion Data

HBase can store data of historical versions. You can set **VERSIONS** to specify the number of versions to be stored.

Run the following commands to add data:

```
put   'cx_table_stu01','20200001','cf1:name','ZhangSan'
put   'cx_table_stu01','20200001','cf1:name','LiSi'
put   'cx_table_stu01','20200001','cf1:name','WangWu'
```

Run the following command to scan the table and display the data:

```
scan 'cx_table_stu01'
```

```
hbase:019:0> put   'cx_table_stu01','20200001','cf1:name','ZhangSan'
Took 0.0087 seconds
hbase:020:0> put   'cx_table_stu01','20200001','cf1:name','LiSi'
Took 0.0060 seconds
hbase:021:0> put   'cx_table_stu01','20200001','cf1:name','WangWu'
Took 0.0079 seconds
hbase:022:0> scan 'cx_table_stu01'
ROW                    COLUMN+CELL
 20200001              column=cf1:age, timestamp=[redacted]T17:43:42.127, value=20
 20200001              column=cf1:gender, timestamp=[redacted]T17:43:32.247, value=male
 20200001              column=cf1:name, timestamp=[redacted]T17:51:18.599, value=WangWu
 20200002              column=cf1:age, timestamp=[redacted]T17:44:22.988, value=19
 20200002              column=cf1:gender, timestamp=[redacted]T17:44:14.404, value=femal
 20200002              column=cf1:name, timestamp=[redacted]T17:44:04.117, value=hanmeim
                       i
2 row(s)
Took 0.0084 seconds
```

Figure 2-8

Run the following command to specify multiple versions to be queried:

```
get   'cx_table_stu01','20200001',{COLUMNS=>'cf1',VERSIONS=>5}
```

```
hbase:023:0> get  'cx_table_stu01','20200001',{COLUMNS=>'cf1',VERSIONS=>5}
COLUMN                  CELL
 cf1:age                timestamp=[redacted]T17:43:42.127, value=20
 cf1:gender             timestamp=[redacted]T17:43:32.247, value=male
 cf1:name               timestamp=[redacted]T17:51:18.599, value=WangWu
1 row(s)
Took 0.0091 seconds
```

Figure 2-9

The version is specified during the query, but the last record is still displayed. Although **VERSIONS** is specified, only one record is returned. This is because the default value of **VERSIONS** is **1** during table creation.

Run the following command to view the table attributes:

```
desc  'cx_table_stu01'
```

```
hbase:025:0> desc  'cx_table_stu01'
Table cx_table_stu01 is ENABLED
cx_table_stu01
COLUMN FAMILIES DESCRIPTION
{NAME => 'cf1', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', VERSIONS => '1', KEEP_DELETED_CEL
LS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', COMPRESSION => 'NONE', TTL => 'FOREVER', MIN_VE
RSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}

1 row(s)

QUOTAS
0 row(s)
Took 0.1148 seconds
hbase:026:0>
```

Figure 2-10

To view data of multiple versions, run the following statement to change the value of **VERSIONS** of the table or specify its value during table creation:

```
alter 'cx_table_stu01',{NAME=>'cf1','VERSIONS'=>5}
```

Run the following commands to insert multiple data records:

```
put  'cx_table_stu01','20200001','cf1:name','ZhangSan'
put  'cx_table_stu01','20200001','cf1:name','LiSi'
put  'cx_table_stu01','20200001','cf1:name','WangWu'
```

Run the following command to view the value of **name**:

```
get  'cx_table_stu01','20200001',{COLUMNS=>'cf1',VERSIONS=>5}
```

```
hbase:026:0> alter 'cx_table_stu01',{NAME=>'cf1','VERSIONS'=>5}
Updating all regions with the new schema...
1/1 regions updated.
Done.
Took 2.7900 seconds
hbase:027:0> put  'cx_table_stu01','20200001','cf1:name','ZhangSan'
Took 0.0104 seconds
hbase:028:0> put  'cx_table_stu01','20200001','cf1:name','LiSi'
Took 0.0062 seconds
hbase:029:0> put  'cx_table_stu01','20200001','cf1:name','WangWu'
Took 0.0077 seconds
hbase:030:0> get  'cx_table_stu01','20200001',{COLUMNS=>'cf1',VERSIONS=>5}
COLUMN                  CELL
 cf1:age                timestamp=[redacted]T17:43:42.127, value=20
 cf1:gender             timestamp=[redacted]T17:43:32.247, value=male
 cf1:name               timestamp=[redacted]T17:56:56.693, value=WangWu
 cf1:name               timestamp=[redacted]T17:56:56.661, value=LiSi
 cf1:name               timestamp=[redacted]T17:56:56.634, value=ZhangSan
 cf1:name               timestamp=[redacted]T17:51:18.599, value=WangWu
1 row(s)
Took 0.0155 seconds
```

Figure 2-11

2.2.1.8 Deleting Data

Run the following commands to delete data from a column family:

```
delete 'cx_table_stu01','20200002','cf1:age'
get 'cx_table_stu01','20200002'
```

```
hbase:031:0> delete 'cx_table_stu01','20200002','cf1:age'
Took 0.0120 seconds
hbase:032:0> get 'cx_table_stu01','20200002'
COLUMN                  CELL
 cf1:gender             timestamp=[redacted]T17:44:14.404, value=female
 cf1:name               timestamp=[redacted]T17:44:04.117, value=hanmeimei
1 row(s)
Took 0.0162 seconds
```

Figure 2-12

Run the following commands to delete a row of data:

```
deleteall   'cx_table_stu01','20200002'
get 'cx_table_stu01','20200002'
```

```
hbase:033:0> deleteall  'cx_table_stu01','20200002'
Took 0.0071 seconds
hbase:034:0> get 'cx_table_stu01','20200002'
COLUMN                  CELL
0 row(s)
Took 0.0079 seconds
hbase:035:0>
```

Figure 2-13

2.2.1.9 Deleting a Table

You can run the **drop** command to delete a table. Make sure the table is disabled before being deleted.

Run the **disable'***Table name*' command to disable the table, and run the **drop'***Table name*' command to delete the table:

```
disable 'cx_table_stu01'
drop 'cx_table_stu01'
```

```
hbase:035:0> disable 'cx_table_stu01'
[redacted],377 INFO  [main] client.HBaseAdmin: Started disable of cx_table_stu01
[redacted],175 INFO  [main] client.HBaseAdmin: Operation: DISABLE, Table Name: defau
lt:cx_table_stu01, procId: 26 completed
Took 0.8269 seconds
hbase:036:0> drop 'cx_table_stu01'
[redacted],189 INFO  [main] client.HBaseAdmin: Operation: DELETE, Table Name: defaul
t:cx_table_stu01, procId: 29 completed.
Took 0.8779 seconds
hbase:037:0> list
TABLE
0 row(s)
Took 0.0103 seconds
=> []
```

Figure 2-14

2.2.2 Pre-splitting Regions During Table Creation

By default, HBase creates a table with only one region. The rowkey of the region has no start key or end key. All data is written to the default region. As the data volume increases, the region cannot handle the increasing data. Therefore, the region is split into two regions. During this process, the following problems may occur:

1. When data is written to a region, data hotspots may occur.
2. Region splitting consumes valuable cluster I/O resources.

To resolve these problems, create multiple empty regions during table creation, and set the start and end row keys of each region. In this way, as long as the rowkeys can ensure the data is evenly written to each region, there is no write hotspot problem, and the probability of splitting is greatly reduced. HBase provides two pre-splitting algorithms: HexStringSplit and UniformSplit. HexStringSplit applies to the rowkey of hexadecimal characters, and UniformSplit applies to the rowkey of random byte arrays.

2.2.2.1 Splitting Regions into Four Partitions Randomly by Rowkey

Step 1 Create a table.

Run the following command to create a table:

```
create 'cx_table_stu02','cf2', {NUMREGIONS => 4 , SPLITALGO => 'UniformSplit'}
```

```
hbase:038:0> create 'cx_table_stu02','cf2', {NUMREGIONS => 4 , SPLITALGO => 'UniformSplit'}
[redacted],419 INFO  [main] client.HBaseAdmin: Operation: CREATE, Table Name: defaul
t:cx_table_stu02, procId: 30 completed
Created table cx_table_stu02
Took 2.2354 seconds
=> Hbase::Table - cx_table_stu02
```

Figure 2-15

Region name format: [*Table*],[*Region start key*],[*Region ID*]

Step 2 Log in to the HBase web UI and check the table partitions.

Log in to FusionInsight Manager, choose **HBase** under the **mrs_hcia** cluster on the **Homepage** page.

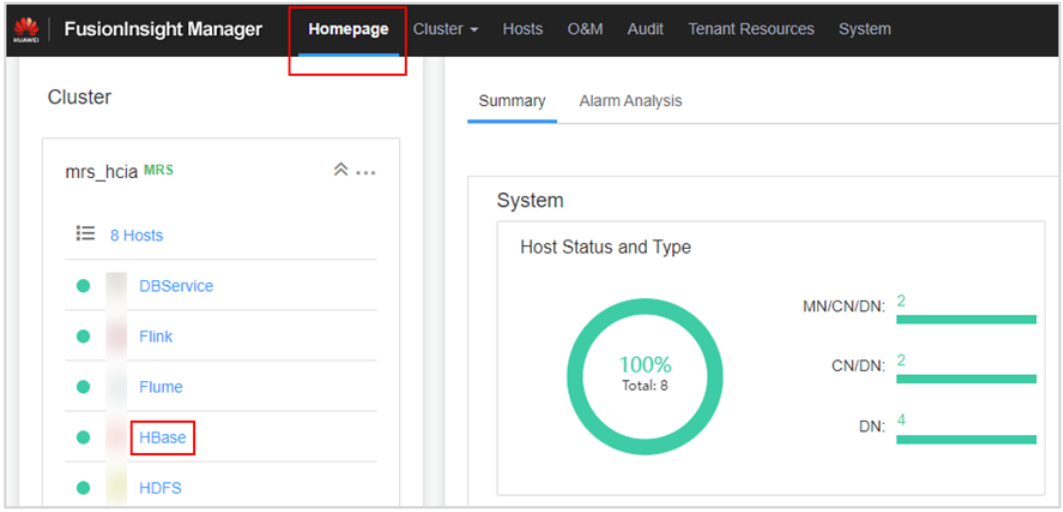


Figure 2-16

Click **HMaster(Active)** to go to the HMaster web UI.

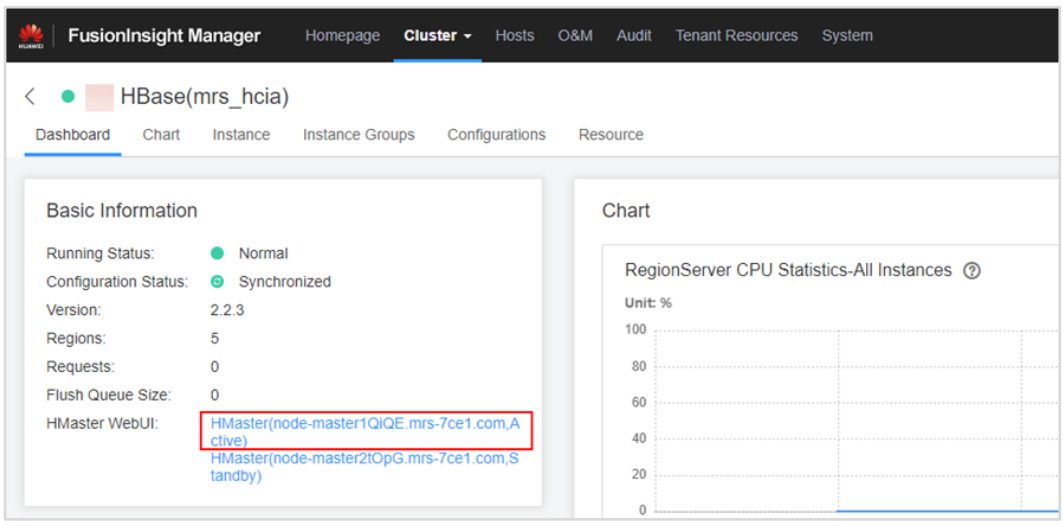


Figure 2-17

Click **cx_table_stu02** on the **User Tables** tab to go to the **Tables Regions** page.

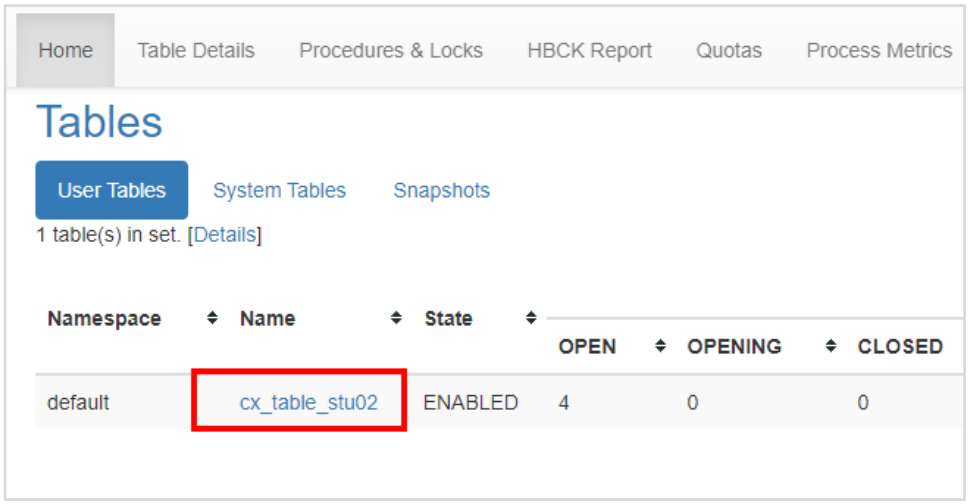


Figure 2-18

As you can see, the **cx_table_stu02** table has four partitions.

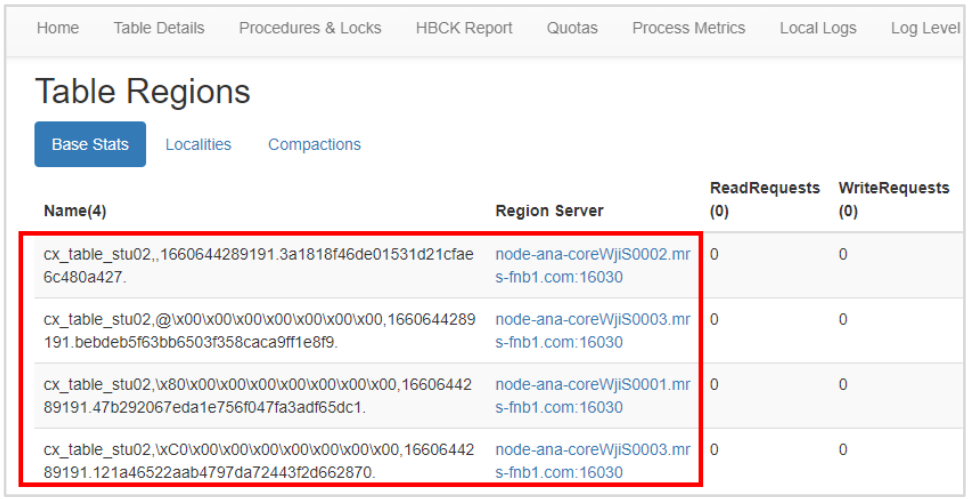


Figure 2-19

2.2.2.2 Checking the Start Key and End Key of a Specified Region

Run the following command to create a table:

```
create 'cx_table_stu03', 'cf3', SPLITS => ['10000', '20000', '30000']
```

Perform the same operations to check the table partitions.

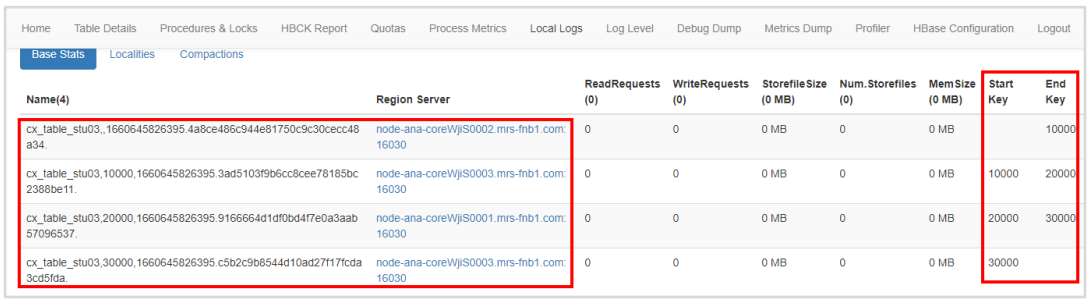


Figure 2-20

2.2.3 Using Filters

Run the following commands:

```
scan 'cx_table_stu01',{FILTER=>"ValueFilter(=,'binary:20')"}
scan 'cx_table_stu01',{FILTER=>"ValueFilter(=,'binary:tom')"}
scan 'cx_table_stu01',FILTER=>"ColumnPrefixFilter('gender')"
scan 'cx_table_stu01',{FILTER=>"ColumnPrefixFilter('name') AND ValueFilter(=,'binary:hanmeimei')"}
```

```
hbase:055:0> scan 'cx_table_stu01',{FILTER=>"ValueFilter(=,'binary:20')"}
ROW                     COLUMN+CELL
 20200001               column=cf1:age, timestamp=[illegible]T18:36:01.224, value=20
1 row(s)
Took 0.0454 seconds
hbase:056:0>
hbase:057:0> scan 'cx_table_stu01',{FILTER=>"ValueFilter(=,'binary:tom')"}
ROW                     COLUMN+CELL
 20200001               column=cf1:name, timestamp=[illegible]T18:36:01.130, value=tom
1 row(s)
Took 0.0056 seconds
hbase:058:0>
hbase:059:0> scan 'cx_table_stu01',FILTER=>"ColumnPrefixFilter('gender')"
ROW                     COLUMN+CELL
 20200001               column=cf1:gender, timestamp=[illegible]T18:36:01.178, value=male
 20200002               column=cf1:gender, timestamp=[illegible]T18:36:01.307, value=female
2 row(s)
Took 0.0159 seconds
hbase:060:0>
hbase:061:0> scan 'cx_table_stu01',{FILTER=>"ColumnPrefixFilter('name') AND ValueFilter(=,'bi
nary:hanmeimei')"}
ROW                     COLUMN+CELL
 20200002               column=cf1:name, timestamp=[illegible]T18:36:01.269, value=hanmeimei
1 row(s)
Took 0.0112 seconds
```

Figure 2-21

Note: If the **cx_table_stu01** table is deleted in the previous exercise, recreate the table and insert data.

2.3 Exercise Summary

2.3.1 Quiz

What are the application scenarios of HBase? What are its advantages and disadvantages?

2.3.2 Summary

This exercise describes how to create and delete HBase tables and add, delete, modify, and query data, how to pre-split regions, and how to use filters to query data. After completing this exercise, you will better understand and use HBase.

3 Hive Data Warehouse Practices

3.1 About This Exercise

3.1.1 Overview

Hive is a data warehouse tool and plays an important role in data mining, data aggregation, and statistical analysis. In telecom services, Hive can be used to collect statistics on users' data usage and phone bills, and mine user consumption models to help carriers better design packages.

3.1.2 Objectives

Master common Hive operations and learn how to run HQL on Hue.

3.2 Tasks

3.2.1 Creating Hive Tables

3.2.1.1 Table Creation Statements

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table_name
  [(col_name data_type [COMMENT col_comment], ...)]
  [COMMENT table_comment]
  [PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
  [CLUSTERED BY (col_name, col_name, ...)
  [SORTED BY (col_name [ASC|DESC], ...)] INTO num_buckets BUCKETS]
  [ROW FORMAT row_format]
  [STORED AS file_format]
  [LOCATION hdfs_path]
```

3.2.1.2 Creating Internal Tables

Run the following command to set environment variables:

```
source   /opt/client/bigdata_env
```

Enter **beeline** and press **Enter** to go to Hive.

```
beeline
```

Note: All statements in Hive must end with a semicolon (;). Otherwise, the statements cannot be executed.

You can run the following command to filter the output of INFO logs:

```
beeline  --hiveconf hive.server2.logging.operation.level=NONE
```

```
[root@node-masterledGF ~]# source  /opt/client/bigdata_env
[root@node-masterledGF ~]# beeline  --hiveconf hive.server2.logging.operation.level=NONE
```

Figure 3-1

Statement for creating internal tables (If multiple users share the same environment, it is recommended that users name tables with the first letters of their last and first names to differentiate tables.)

```
create  table  cx_stu01(name string,gender string,age int) row format delimited fields terminated by ',' stored as textfile;
show tables;       #Display all tables.
```

```
0: jdbc:hive2://192.168.0.182:10000/> create  table  cx_stu01(name string,gender string,age int) row format delimited fields te
rminated by ',' stored as textfile ;
No rows affected (0.311 seconds)
0: jdbc:hive2://192.168.0.182:10000/> show tables;
+-----------+
| tab_name  |
+-----------+
| cx_stu01  |
+-----------+
1 row selected (0.136 seconds)
0: jdbc:hive2://192.168.0.182:10000/>
```

Figure 3-2

3.2.1.3 Creating External Tables

Run the following command to create an external table:

```
create external table cx_stu02(name string,gender string,age int) row format delimited fields terminated by ',' stored as textfile;
```

```
0: jdbc:hive2://192.168.0.182:10000/> create external table cx_stu02(name string,gender string,age int) row format delim
ited fields terminated by ',' stored as textfile;
No rows affected (0.07 seconds)
0: jdbc:hive2://192.168.0.182:10000/> show tables;
+-----------+
| tab_name  |
+-----------+
| cx_stu01  |
| cx_stu02  |
+-----------+
2 rows selected (0.046 seconds)
0: jdbc:hive2://192.168.0.182:10000/>
```

Figure 3-3

3.2.1.4 Loading HDFS Data

Press **Ctrl+C** to exit Hive (or open a new shell window), and run the following **vi** command to edit the **cx_stu01.txt** file on the local Linux host:

```
[root@node-masterledGF ~]# source /opt/client/bigdata_env
[root@node-masterledGF ~]# vi cx_stu01.txt
[root@node-masterledGF ~]# cat cx_stu01.txt
tom,male,19
hanmeimei,female,20
jack,female,22
lilei,female,18
lily,male,23
```

Figure 3-4

Run the following **put** command to upload data to the **/user/stu01/** directory of HDFS:

```
hdfs dfs -put cx_stu01.txt /user/stu01/
```

```
[root@node-masterledGF ~]# hdfs dfs -put cx_stu01.txt /user/stu01/
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib
class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/cc
rBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
[root@node-masterledGF ~]# hdfs dfs -ls /user/stu01/
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib
class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/cc
rBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 5 items
-rw-r--r--   2 root hadoop         76 [redacted] /user/stu01/cx_stu01.txt
-rw-r--r--   2 root hadoop         25 [redacted] /user/stu01/stu01.txt
-rw-r--r--   2 root hadoop         50 [redacted] /user/stu01/stu02.txt
-rw-r--r--   2 root hadoop         16 [redacted] /user/stu01/stu04.txt
-rw-r--r--   2 root hadoop         16 [redacted] /user/stu01/stu05.txt
```

Figure 3-5

Run the **beeline** command to go to Hive and run the following command to import data to the external table:

```
load data inpath '/user/stu01/cx_stu01.txt' into table cx_stu02;
```

```
0: jdbc:hive2://192.168.0.4:10000/> load data inpath '/user/stu01/cx_stu01.txt' into table cx_stu02;
No rows affected (6.674 seconds)
0: jdbc:hive2://192.168.0.4:10000/> select * from cx_stu02;
+----------------+-----------------+---------------+
| cx_stu02.name  | cx_stu02.gender | cx_stu02.age  |
+----------------+-----------------+---------------+
| tom            | male            | 19            |
| hanmeimei      | female          | 20            |
| jack           | female          | 22            |
| lilei          | female          | 18            |
| lily           | male            | 23            |
+----------------+-----------------+---------------+
5 rows selected (0.545 seconds)
0: jdbc:hive2://192.168.0.4:10000/>
```

Figure 3-6

3.2.2 Performing Basic Hive Queries

3.2.2.1 Fuzzy Queries

Run the following statement:

```
show tables like 'cx_stu*';
```

```
0: jdbc:hive2://192.168.0.4:10000/> show tables like 'cx_stu*';
+-----------+
| tab_name  |
+-----------+
| cx_stu01  |
| cx_stu02  |
+-----------+
2 rows selected (0.133 seconds)
0: jdbc:hive2://192.168.0.4:10000/>
```

Figure 3-7

3.2.2.2 Simple Queries

Step 1 Run the **limit** command.

Run the following statement:

```
select * from cx_stu02 limit 2;
```

```
0: jdbc:hive2://192.168.0.4:10000/> select * from cx_stu02 limit 2;
+----------------+-----------------+---------------+
| cx_stu02.name  | cx_stu02.gender | cx_stu02.age  |
+----------------+-----------------+---------------+
| tom            | male            | 19            |
| hanmeimei      | female          | 20            |
+----------------+-----------------+---------------+
2 rows selected (0.102 seconds)
0: jdbc:hive2://192.168.0.4:10000/>
```

Figure 3-8

Step 2 Run the **where** command.

Run the following statement:

```
select * from cx_stu02 where gender ='male' limit 2;
```

```
0: jdbc:hive2://192.168.0.4:10000/> select * from cx_stu02 where gender ='male' limit 2;
+----------------+-----------------+---------------+
| cx_stu02.name  | cx_stu02.gender | cx_stu02.age  |
+----------------+-----------------+---------------+
| tom            | male            | 19            |
| lily           | male            | 23            |
+----------------+-----------------+---------------+
2 rows selected (23.006 seconds)
0: jdbc:hive2://192.168.0.4:10000/>
```

Figure 3-9

Step 3 Run the **order** command.

Run the following statement:

```
select * from cx_stu02 where gender ='female' order by age limit 2;
```

```
0: jdbc:hive2://192.168.0.4:10000/> select * from cx_stu02 where gender ='female' order by age limit 2;
+----------------+------------------+---------------+
| cx_stu02.name  | cx_stu02.gender  | cx_stu02.age  |
+----------------+------------------+---------------+
| lilei          | female           | 18            |
| hanmeimei      | female           | 20            |
+----------------+------------------+---------------+
2 rows selected (29.018 seconds)
0: jdbc:hive2://192.168.0.4:10000/>
```

Figure 3-10

3.2.2.3 Complex Queries

Step 1 Run the **vi** command to edit the **cx_stu03.txt** data file on the local Linux host:

```
[root@node-masterledGF ~]# vi cx_stu03.txt
[root@node-masterledGF ~]# cat cx_stu03.txt
1001,Jack,Chinese,78
1002,Jack,English,82
1003,Jack,Math,87
1004,Mark,Chinese,69
1005,Mark,English,89
1006,Mark,Math,73
1007,Hanke,Chinese,89
1008,Hanke,English,85
1009,Hanke,Math,75
[root@node-masterledGF ~]#
```

Figure 3-11

Step 2 Upload data to HDFS.

Run the following command to upload data to HDFS:

```
hdfs dfs -put cx_stu03.txt /user/stu01/
```

```
[root@node-masterledGF ~]# hdfs  dfs  -put  cx_stu03.txt   /user/stu01/
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/li
class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/c
rBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
[root@node-masterledGF ~]# hdfs  dfs  -ls /user/stu01/
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/li
class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/c
rBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 5 items
-rw-r--r--   2 root hadoop        183 [redacted] /user/stu01/cx_stu03.txt
-rw-r--r--   2 root hadoop         25 [redacted] /user/stu01/stu01.txt
-rw-r--r--   2 root hadoop         50 [redacted] /user/stu01/stu02.txt
-rw-r--r--   2 root hadoop         16 [redacted] /user/stu01/stu04.txt
-rw-r--r--   2 root hadoop         16 [redacted] /user/stu01/stu05.txt
```

Figure 3-12

Step 3 Create a table and import data to the table.

Run the **beeline** command to go to Hive and run the following command to create a table:

```
create external table cx_table_stu03(id int,name string ,subject string,score   float)   row format
delimited fields terminated by ',' stored as textfile ;
```

Run the following command to import data:

```
load   data   inpath   '/user/stu01/cx_stu03.txt'   into   table   cx_table_stu03;
```

```
0: jdbc:hive2://192.168.0.4:10000/> create external table cx_table_stu03(id int,name string ,subject string,score  float)  row format
 delimited fields terminated by ',' stored as textfile ;
No rows affected (0.141 seconds)
0: jdbc:hive2://192.168.0.4:10000/> load  data  inpath  '/user/stu01/cx_stu03.txt'  into  table  cx_table_stu03;
No rows affected (0.12 seconds)
0: jdbc:hive2://192.168.0.4:10000/> select * from cx_table_stu03;
+--------------------+----------------------+-------------------------+-----------------------+
| cx_table_stu03.id  | cx_table_stu03.name  | cx_table_stu03.subject  | cx_table_stu03.score  |
+--------------------+----------------------+-------------------------+-----------------------+
| 1001               | Jack                 | Chinese                 | 78.0                  |
| 1002               | Jack                 | English                 | 82.0                  |
| 1003               | Jack                 | Math                    | 87.0                  |
| 1004               | Mark                 | Chinese                 | 69.0                  |
| 1005               | Mark                 | English                 | 89.0                  |
| 1006               | Mark                 | Math                    | 73.0                  |
| 1007               | Hanke                | Chinese                 | 89.0                  |
| 1008               | Hanke                | English                 | 85.0                  |
| 1009               | Hanke                | Math                    | 75.0                  |
+--------------------+----------------------+-------------------------+-----------------------+
9 rows selected (0.102 seconds)
0: jdbc:hive2://192.168.0.4:10000/>
```

Figure 3-13

Step 4 Run the **sum** command.

Run the following command to calculate the total score of each student:

```
select name, sum(score) total_score from cx_table_stu03 group by name ;
```

```
0: jdbc:hive2://192.168.0.182:10000/> select  name, sum(score)  total_score  from  cx_table_stu0
3  group  by  name ;
+--------+--------------+
|  name  | total_score  |
+--------+--------------+
| Hanke  | 249.0        |
| Jack   | 247.0        |
| Mark   | 231.0        |
+--------+--------------+
3 rows selected (25.756 seconds)
```

Figure 3-14

To calculate the total score of each student and filter out students whose total score is greater than 230, run the following statement:

```
select name, sum(score) total_score from cx_table_stu03 group by name having
total_score > 235;
```

```
0: jdbc:hive2://192.168.0.182:10000/> select  name, sum(score)  total_score  from  cx_table_stu0
3  group  by  name  having  total_score > 235;
+--------+--------------+
|  name  | total_score  |
+--------+--------------+
| Hanke  | 249.0        |
| Jack   | 247.0        |
+--------+--------------+
2 rows selected (26.75 seconds)
```

Figure 3-15

Step 5 Run the **max** command.

Run the following command to display the highest score of each course:

```
select subject,max(score) from cx_table_stu03 group by subject;
```

```
0: jdbc:hive2://192.168.0.182:10000/> select  subject,max(score)  from  cx_table_stu03  group by
 subject;
+----------+-------+
| subject  |  _c1  |
+----------+-------+
| Chinese  | 89.0  |
| English  | 89.0  |
| Math     | 87.0  |
+----------+-------+
3 rows selected (24.275 seconds)
```

Figure 3-16

Step 6 Run the **count** command.

Run the following command to calculate the number of trainees taking the exam in each course:

```
select subject,count(1) from cx_table_stu03 group by subject;
```

```
0: jdbc:hive2://192.168.0.182:10000/> select subject,count(1) from cx_table_stu03 group by subje
ct;
+----------+------+
| subject  | _c1  |
+----------+------+
| Chinese  | 3    |
| English  | 3    |
| Math     | 3    |
+----------+------+
3 rows selected (27.565 seconds)
```

Figure 3-17

3.2.3 Performing Hive Join Operations

Hive supports common SQL join commands, such as INNER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, and map-side JOIN which is Hive-exclusive.

Step 1 Create a table and import data to the table.

Create tables **cx_table_employee** (employee table), **cx_table_department** (department table), and **cx_table_salary** (salary table), and import data to them. Note: For details, see the previous exercises.

The statement for creating the **cx_table_employee** table is as follows:

```
create table if not exists cx_table_employee(user_id int, username string, dept_id int) row format
delimited fields terminated by ','  stored as textfile ;
```

```
0: jdbc:hive2://192.168.0.182:10000/> create table if not exists cx_table_employee(user_id int,
username string, dept_id int) row format delimited fields terminated by ','  stored as textfile
;
No rows affected (0.24 seconds)
0: jdbc:hive2://192.168.0.182:10000/>
```

Figure 3-18

The statement for creating the **cx_table_department** table is as follows:

```
create table if not exists cx_table_department(dept_id int, dept_name string) row format delimited
fields terminated by ','  stored as textfile ;
```

```
0: jdbc:hive2://192.168.0.182:10000/> create table if not exists cx_table_department(dept_id int
, dept_name string) row format delimited fields terminated by ','  stored as textfile ;
No rows affected (0.063 seconds)
```

Figure 3-19

The statement for creating the **cx_table_salary** table is as follows:

```
create table if not exists cx_table_salary(userid int, dept_id int, salarys double) row format delimited
fields terminated by ',' stored as textfile ;
```

```
0: jdbc:hive2://192.168.0.182:10000/> create table if not exists cx_table_salary(userid int, dep
t_id int, salarys double) row format delimited fields terminated by ',' stored as textfile ;
No rows affected (0.096 seconds)
0: jdbc:hive2://192.168.0.182:10000/>
```

Figure 3-20

The data in the three tables is as follows:

cx_table_ employee (employee table):

```
1,zhangsas,1
2,lisi,2
3,wangwu,3
4,tom,1
5,lily,2
6,amy,3
7,lilei,1
8,hanmeimei,2
9,poly,3
```

cx_table_department (department table):

```
1,Technical
2,sales
3,HR
4,marketing
```

cx_table_salary (salary table):

```
1,1,20000
2,2,16000
3,3,20000
4,1,50000
5,2,18900
6,3,12098
7,1,21900
```

Step 2 Perform INNER JOIN.

When INNER JOIN is performed on multiple tables, only the data that matches the **on** condition in all tables is displayed. For example, the following SQL statement joins the employee table and the department table. The **on** condition is **dept_id**, that is, only data with the same **dept_id** is displayed.

Run the following statement:

```
select e.username,e.dept_id,d.dept_name,d.dept_id from cx_table_employee e join
cx_table_department d on e.dept_id = d.dept_id;
```

```
0: jdbc:hive2://192.168.0.182:10000/> select e.username,e.dept_id,d.dept_name,d.dept_id from
 cx_table_employee e join cx_table_department d on e.dept_id = d.dept_id;
+-------------+------------+-------------+------------+
| e.username  | e.dept_id  | d.dept_name | d.dept_id  |
+-------------+------------+-------------+------------+
| zhangsas    | 1          | Technical   | 1          |
| lisi        | 2          | sales       | 2          |
| wangwu      | 3          | HR          | 3          |
| tom         | 1          | Technical   | 1          |
| lily        | 2          | sales       | 2          |
| amy         | 3          | HR          | 3          |
| lilei       | 1          | Technical   | 1          |
| hanmeimei   | 2          | sales       | 2          |
| poly        | 3          | HR          | 3          |
+-------------+------------+-------------+------------+
9 rows selected (29.424 seconds)
0: jdbc:hive2://192.168.0.182:10000/>
```

Figure 3-21

You can join two or more tables. Run the following SQL statement to query the employee names, departments, and salaries: Run the following statement:

```
select e.username,d.dept_name,s.salarys from cx_table_employee e join cx_table_department d on
e.dept_id = d.dept_id join cx_table_salary s on e.user_id = s.userid;
```

```
0: jdbc:hive2://192.168.0.182:10000/> select e.username,d.dept_name,s.salarys from cx_table_
employee e join cx_table_department d on e.dept_id = d.dept_id join cx_table_salary s on e.u
ser_id = s.userid;
+-------------+--------------+-------------+
| e.username  | d.dept_name  | s.salarys   |
+-------------+--------------+-------------+
| zhangsas    | Technical    | 20000.0     |
| lisi        | sales        | 16000.0     |
| wangwu      | HR           | 20000.0     |
| tom         | Technical    | 50000.0     |
| lily        | sales        | 18900.0     |
| amy         | HR           | 12980.0     |
| lilei       | Technical    | 21900.0     |
+-------------+--------------+-------------+
7 rows selected (31.943 seconds)
0: jdbc:hive2://192.168.0.182:10000/>
```

Figure 3-22

Generally, a MapReduce job is generated for a join operation. If more than two tables are joined, Hive associates the tables from left to right. For the preceding SQL statement, a MapReduce job task is firstly started to join the employee and department tables, and then the second MapReduce job is started to join the output of the first MapReduce job with the salary table. This is contrary to the standard SQL statement, which performs the join operation from right to left. Therefore, in Hive SQL, small tables are written on the left to improve the execution efficiency.

In Hive, you can use the **/*+STREAMTALBE*/** syntax to specify a large table. For example, in the following SQL statement, **dept** is specified as a large table. Otherwise, Hive considers the rightmost table as a large table.

Run the following statement:

```
select /*+STREAMTABlE(d)*/ e.username,e.dept_id,d.dept_name,d.dept_id from cx_table_employee e
join cx_table_department d on e.dept_id = d.dept_id;
```

```
0: jdbc:hive2://192.168.0.182:10000/> select /*+STREAMTABLE(d)*/ e.username,e.dept_id,d.dept
_name,d.dept_id from cx_table_employee e join cx_table_department d on e.dept_id = d.dept_id
;
+-------------+------------+--------------+------------+
| e.username  | e.dept_id  | d.dept_name  | d.dept_id  |
+-------------+------------+--------------+------------+
| zhangsas    | 1          | Technical    | 1          |
| lisi        | 2          | sales        | 2          |
| wangwu      | 3          | HR           | 3          |
| tom         | 1          | Technical    | 1          |
| lily        | 2          | sales        | 2          |
| amy         | 3          | HR           | 3          |
| lilei       | 1          | Technical    | 1          |
| hanmeimei   | 2          | sales        | 2          |
| poly        | 3          | HR           | 3          |
+-------------+------------+--------------+------------+
9 rows selected (28.756 seconds)
0: jdbc:hive2://192.168.0.182:10000/>
```

Figure 3-23

Generally, the number of MapReduce jobs to be started is determined by the number of tables to be joined. However, if the join keys of the **on** condition are the same, only one MapReduce job is started.

Step 3 Perform LEFT OUTER JOIN.

LEFT OUTER JOIN, same as the standard SQL statement, uses the left table as a baseline. If the right table matches the **on** condition, the data is displayed. Otherwise, NULL is displayed.

Run the following statement:

```
select e.user_id,e.username,s.salarys from cx_table_employee e left outer join cx_table_salary s on e.user_id = s.userid;
```

```
0: jdbc:hive2://192.168.0.182:10000/> select e.user_id,e.username,s.salarys from cx_table_em
ployee e left outer join cx_table_salary s on e.user_id = s.userid;
+------------+-------------+------------+
| e.user_id  | e.username  | s.salarys  |
+------------+-------------+------------+
| 1          | zhangsas    | 20000.0    |
| 2          | lisi        | 16000.0    |
| 3          | wangwu      | 20000.0    |
| 4          | tom         | 50000.0    |
| 5          | lily        | 18900.0    |
| 6          | amy         | 12980.0    |
| 7          | lilei       | 21900.0    |
| 8          | hanmeimei   | NULL       |
| 9          | poly        | NULL       |
+------------+-------------+------------+
9 rows selected (27.763 seconds)
0: jdbc:hive2://192.168.0.182:10000/>
```

Figure 3-24

As shown in the preceding figure, all records in the employee table on the left are displayed, and the data that meets the **on** condition in the salary table on the right is displayed. The data that does not meet the **on** condition is displayed as NULL.

Step 4 Perform RIGHT OUTER JOIN.

LEFT OUTER JOIN is opposite to LEFT OUTER JOIN. It uses the table on the right as a baseline. If the table on the left matches the **on** condition, the data is displayed. Otherwise, NULL is displayed.

Hive is a component for big data processing. It is often used to process hundreds of GB or even TB data. Therefore, you are advised to use the **where** condition to filter out data that does not meet the condition when compiling SQL statements. However, for LEFT and RIGHT OUTER JOINs, the **where** condition is executed after the **on** condition. Therefore, to optimize the Hive SQL execution efficiency, use subqueries in scenarios where OUTER JOINs are required. In subqueries, use the **where** condition to filter out data that does not meet the conditions.

Run the following statement:

```
select e1.user_id,e1.username,s.salarys from (select e.* from cx_table_employee e where e.user_id < 8)
e1 left outer join cx_table_salary s on e1.user_id = s.userid;
```

```
0: jdbc:hive2://192.168.0.182:10000/> select e1.user_id,e1.username,s.salarys from (select e
.* from cx_table_employee e where e.user_id < 8) e1 left outer join cx_table_salary s on e1.
user_id = s.userid;
+-------------+--------------+-------------+
| e1.user_id  | e1.username  | s.salarys   |
+-------------+--------------+-------------+
| 1           | zhangsas     | 20000.0     |
| 2           | lisi         | 16000.0     |
| 3           | wangwu       | 20000.0     |
| 4           | tom          | 50000.0     |
| 5           | lily         | 18900.0     |
| 6           | amy          | 12980.0     |
| 7           | lilei        | 21900.0     |
+-------------+--------------+-------------+
7 rows selected (27.504 seconds)
```

Figure 3-25

In the preceding SQL statement, the data whose **user_id** is greater than or equal to **8** is filtered out in the subquery.

Step 5 Perform FULL OUTER JOIN.

FULL OUTER JOIN returns all the data that meets the **where** condition in the table. The data that does not meet the condition is displayed as NULL.

Run the following statement:

```
select e.user_id,e.username,s.salarys from cx_table_employee e full outer join cx_table_salary s on
e.user_id = s.userid where e.user_id > 0;
```

```
0: jdbc:hive2://192.168.0.182:10000/> select e.user_id,e.username,s.salarys from cx_table_em
ployee e full outer join cx_table_salary s on e.user_id = s.userid where e.user_id > 0;
+-------------+--------------+-------------+
| e.user_id   | e.username   | s.salarys   |
+-------------+--------------+-------------+
| 1           | zhangsas     | 20000.0     |
| 2           | lisi         | 16000.0     |
| 3           | wangwu       | 20000.0     |
| 4           | tom          | 50000.0     |
| 5           | lily         | 18900.0     |
| 6           | amy          | 12980.0     |
| 7           | lilei        | 21900.0     |
| 8           | hanmeimei    | NULL        |
| 9           | poly         | NULL        |
+-------------+--------------+-------------+
9 rows selected (28.335 seconds)
0: jdbc:hive2://192.168.0.182:10000/>
```

Figure 3-26

The results of FULL OUTER JOIN and LEFT OUTER JOIN are the same.

Step 6 Perform LEFT SEMI JOIN.

LEFT SEMI JOIN is used to query the data that only meets the requirements of the left table.

Run the following statement:

```
select e.* from cx_table_employee e LEFT SEMI JOIN cx_table_salary s on e.user_id=s.userid;
```

```
0: jdbc:hive2://192.168.0.182:10000/> select e.* from cx_table_employee e LEFT SEMI JOIN cx_
table_salary s on e.user_id=s.userid;
+-------------+--------------+--------------+
| e.user_id   | e.username   | e.dept_id    |
+-------------+--------------+--------------+
| 1           | zhangsas     | 1            |
| 2           | lisi         | 2            |
| 3           | wangwu       | 3            |
| 4           | tom          | 1            |
| 5           | lily         | 2            |
| 6           | amy          | 3            |
| 7           | lilei        | 1            |
+-------------+--------------+--------------+
7 rows selected (27.031 seconds)
0: jdbc:hive2://192.168.0.182:10000/>
```

Figure 3-27

LEFT SEMI JOIN is an optimization of INNER JOIN. When a data record in the left table exists in the right table, Hive stops scanning. Therefore, the efficiency is higher than that of INNER JOIN. However, the **select** and **where** conditions in LEFT SEMI JOIN can contain only the fields in the left table. Hive does not support RIGHT SEMI JOIN.

Step 7 Perform CARTESIAN JOIN.

CARTESIAN JOIN is used to multiply the data in the left table by the data in the right table.

Run the following statement:

```
select e.user_id,e.username,s.salarys from cx_table_employee e join cx_table_salary s;
```

```
0: jdbc:hive2://192.168.0.182:10000/> select e.user_id,e.username,s.salarys from cx_table_em
ployee e join cx_table_salary s;
+-------------+--------------+-------------+
| e.user_id   | e.username   | s.salarys   |
+-------------+--------------+-------------+
| 1           | zhangsas     | 20000.0     |
| 1           | zhangsas     | 16000.0     |
| 1           | zhangsas     | 20000.0     |
| 1           | zhangsas     | 50000.0     |
| 1           | zhangsas     | 18900.0     |
| 1           | zhangsas     | 12980.0     |
| 1           | zhangsas     | 21900.0     |
| 2           | lisi         | 20000.0     |
| 2           | lisi         | 16000.0     |
| 2           | lisi         | 20000.0     |
| 2           | lisi         | 50000.0     |
| 2           | lisi         | 18900.0     |
| 2           | lisi         | 12980.0     |
| 2           | lisi         | 21900.0     |
| 3           | wangwu       | 20000.0     |
| 3           | wangwu       | 16000.0     |
| 3           | wangwu       | 20000.0     |
| 3           | wangwu       | 50000.0     |
| 3           | wangwu       | 18900.0     |
| 3           | wangwu       | 12980.0     |
| 3           | wangwu       | 21900.0     |
| 4           | tom          | 20000.0     |
| 4           | tom          | 16000.0     |
| 4           | tom          | 20000.0     |
| 4           | tom          | 50000.0     |
| 4           | tom          | 18900.0     |
| 4           | tom          | 12980.0     |
| 4           | tom          | 21900.0     |
| 5           | lily         | 20000.0     |
| 5           | lily         | 16000.0     |
| 5           | lily         | 20000.0     |
| 5           | lily         | 50000.0     |
| 5           | lily         | 18900.0     |
| 5           | lily         | 12980.0     |
| 5           | lily         | 21900.0     |
| 6           | amy          | 20000.0     |
| 6           | amy          | 16000.0     |
| 6           | amy          | 20000.0     |
| 6           | amy          | 50000.0     |
| 6           | amy          | 18900.0     |
| 6           | amy          | 12980.0     |
| 6           | amy          | 21900.0     |
| 7           | lilei        | 20000.0     |
| 7           | lilei        | 16000.0     |
| 7           | lilei        | 20000.0     |
| 7           | lilei        | 50000.0     |
| 7           | lilei        | 18900.0     |
| 7           | lilei        | 12980.0     |
| 7           | lilei        | 21900.0     |
| 8           | hanmeimei    | 20000.0     |
| 8           | hanmeimei    | 16000.0     |
| 8           | hanmeimei    | 20000.0     |
| 8           | hanmeimei    | 50000.0     |
| 8           | hanmeimei    | 18900.0     |
| 8           | hanmeimei    | 12980.0     |
| 8           | hanmeimei    | 21900.0     |
| 9           | poly         | 20000.0     |
| 9           | poly         | 16000.0     |
| 9           | poly         | 20000.0     |
| 9           | poly         | 50000.0     |
| 9           | poly         | 18900.0     |
| 9           | poly         | 12980.0     |
| 9           | poly         | 21900.0     |
+-------------+--------------+-------------+
63 rows selected (29.41 seconds)
0: jdbc:hive2://192.168.0.182:10000/>
```

Figure 3-28

The execution result of the preceding SQL statement is the data in the employee table multiplied by that in the salary table.

Step 8 Perform map-side JOIN.

Map-side JOIN is an optimization of Hive SQL. Hive converts SQL statements into MapReduce jobs. Therefore, the map-side JOIN corresponds to the map-side JOIN in the Hadoop JOIN. Small tables are loaded to the memory to accelerate the Hive SQL execution. You can use either of the following methods to use map-side JOIN of Hive SQL.

Method 1: Use **/*+ MAPJOIN*/** in the following statement:

```
select /*+ MAPJOIN(d)*/ e.username,e.dept_id,d.dept_name,d.dept_id from cx_table_employee e join cx_table_department d on e.dept_id = d.dept_id;
```

```
0: jdbc:hive2://192.168.0.182:10000/> select /*+ MAPJOIN(d)*/ e.username,e.dept_id,d.dept_n
me,d.dept_id from cx_table_employee e join cx_table_department d on e.dept_id = d.dept_id;
+-------------+------------+--------------+------------+
| e.username  | e.dept_id  | d.dept_name  | d.dept_id  |
+-------------+------------+--------------+------------+
| zhangsas    | 1          | Technical    | 1          |
| lisi        | 2          | sales        | 2          |
| wangwu      | 3          | HR           | 3          |
| tom         | 1          | Technical    | 1          |
| lily        | 2          | sales        | 2          |
| amy         | 3          | HR           | 3          |
| lilei       | 1          | Technical    | 1          |
| hanmeimei   | 2          | sales        | 2          |
| poly        | 3          | HR           | 3          |
+-------------+------------+--------------+------------+
9 rows selected (28.681 seconds)
0: jdbc:hive2://192.168.0.182:10000/>
```

Figure 3-29

Method 2: Set **hive.auto.convert.join** to **true**.

3.2.4 Using Hue to Execute HQL

Step 1 Log in to FusionInsight Manager.

In the cluster component list, click **Hue**. The Hue page is displayed.

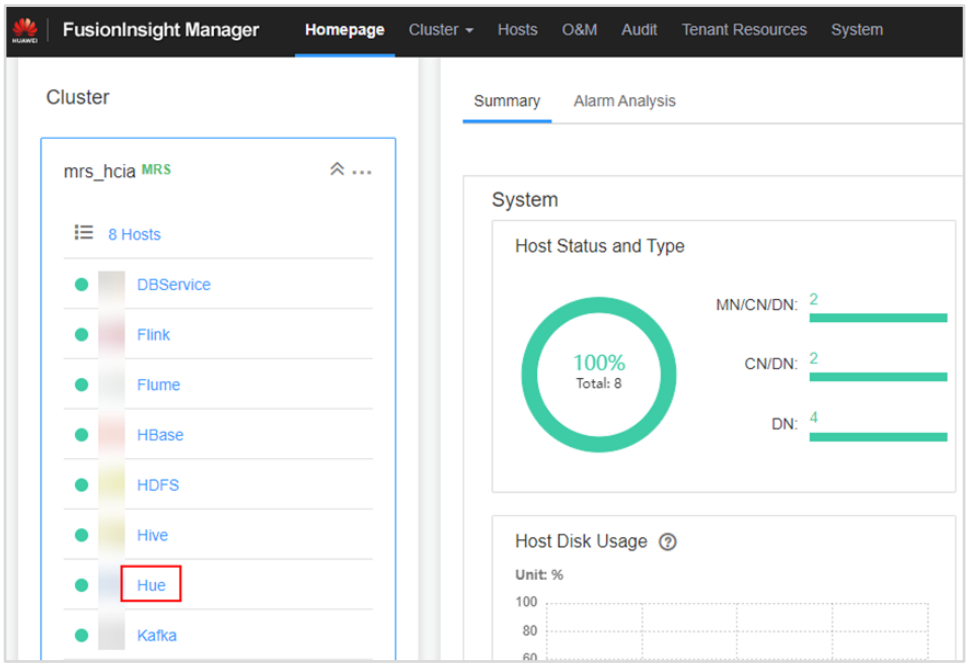


Figure 3-30

Click the active Hue node to go to the Hive operation page.

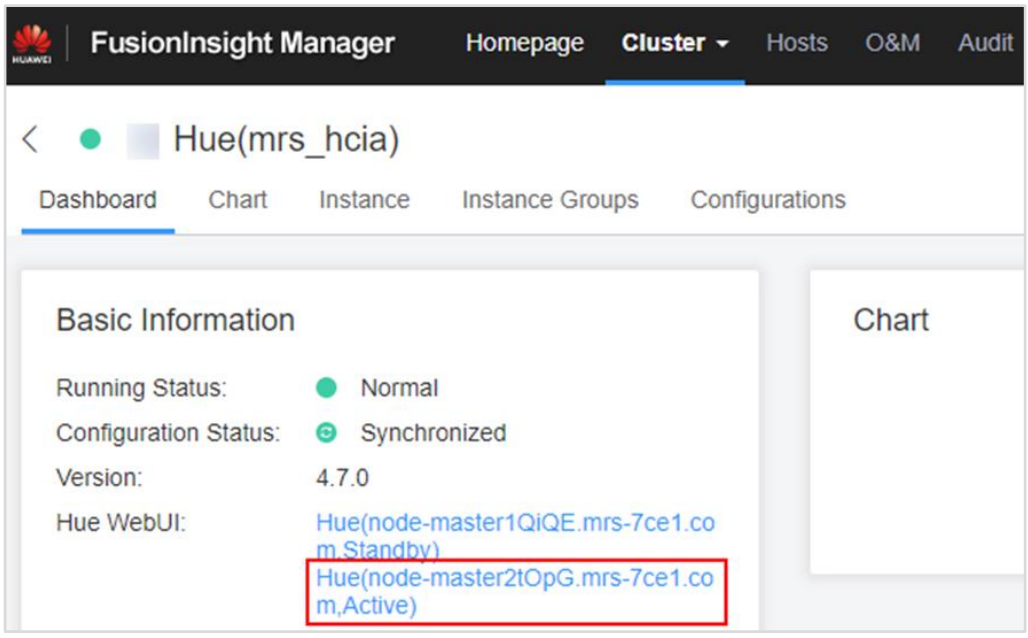


Figure 3-31

The following figure shows the Hive operation page.

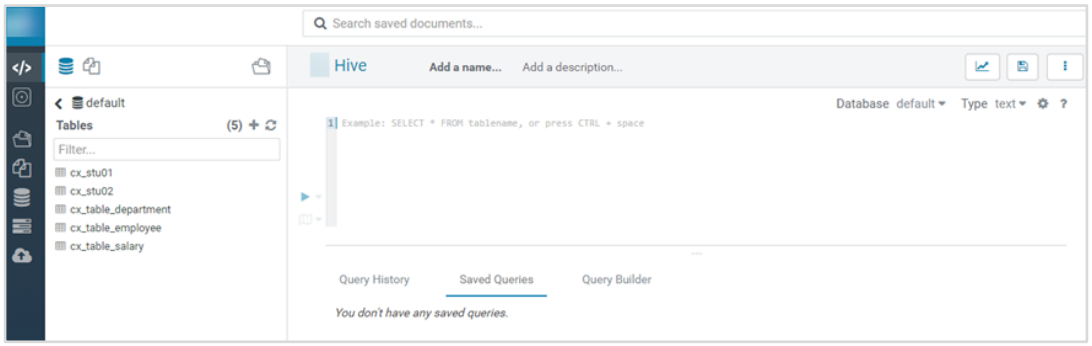


Figure 3-32

Step 2 Edit HQL statements.

Edit HQL statements in the editing area.

```
select * from cx_table_employee;
```

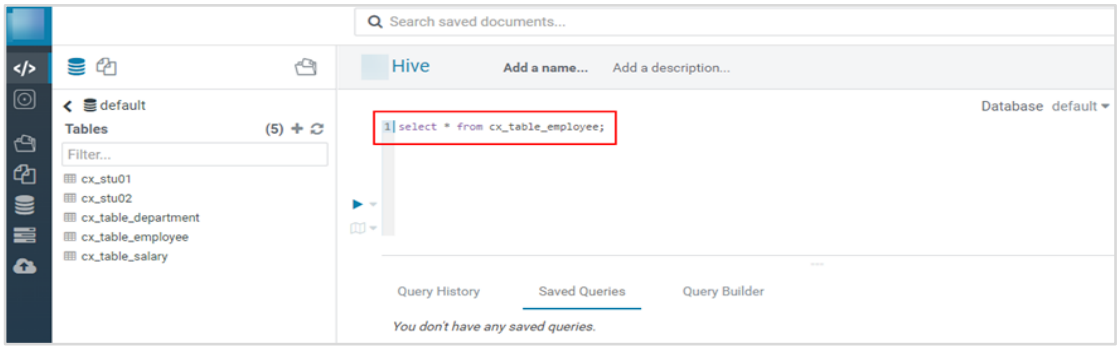


Figure 3-33

Step 3 Execute a query.

Click the triangle button to execute HQL.

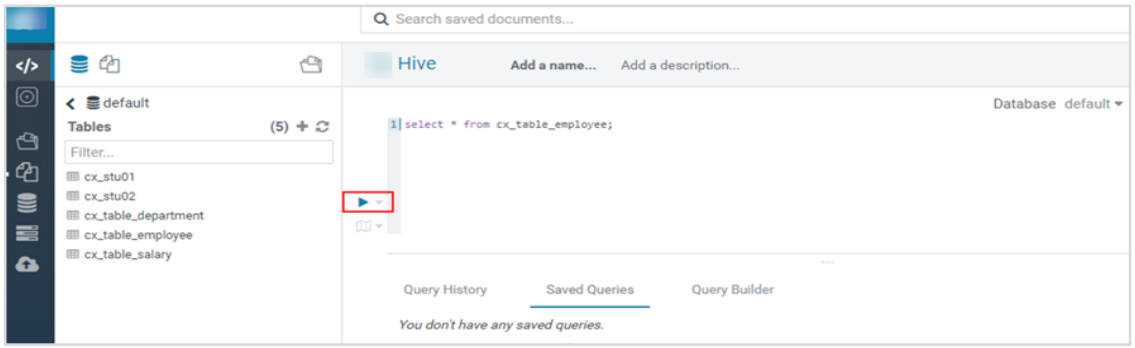
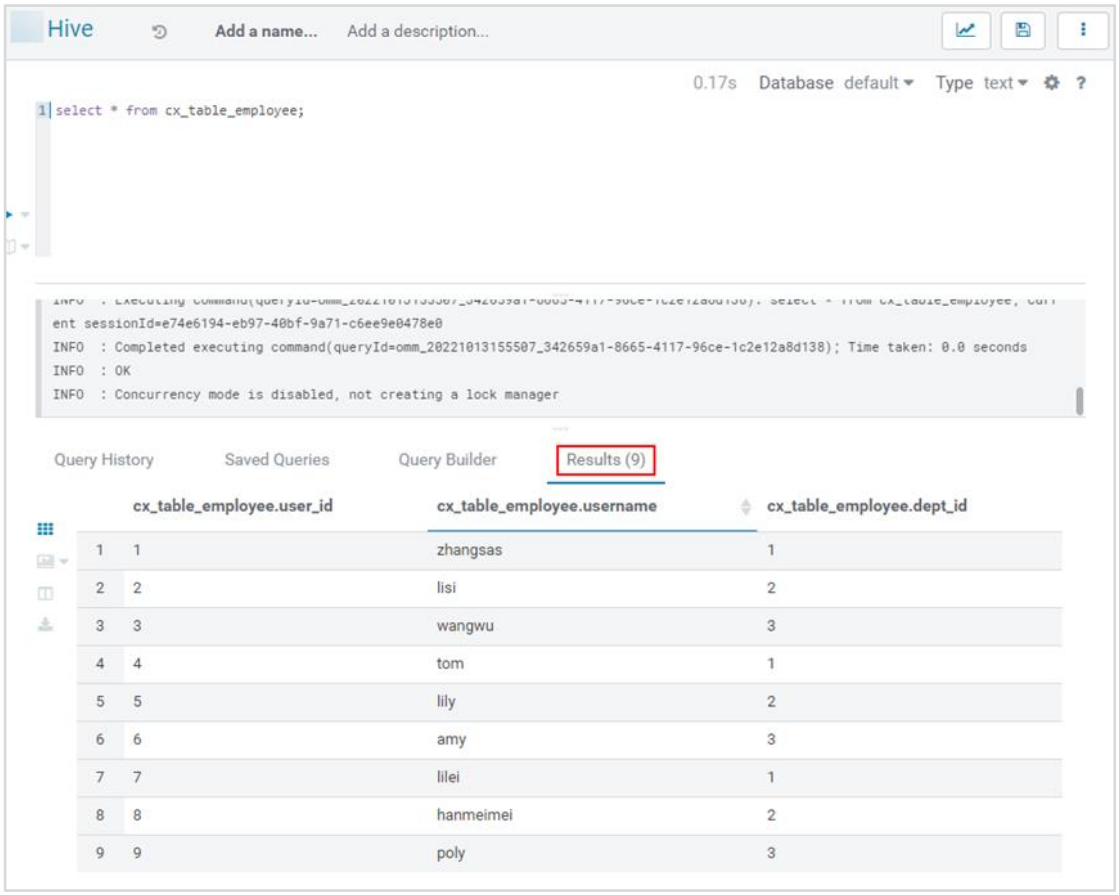


Figure 3-34

Step 4 View the result.

The following figure shows the result.



	cx_table_employee.user_id	cx_table_employee.username	cx_table_employee.dept_id
1	1	zhangsas	1
2	2	lisi	2
3	3	wangwu	3
4	4	tom	1
5	5	lily	2
6	6	amy	3
7	7	lilei	1
8	8	hanmeimei	2
9	9	poly	3

Figure 3-35

3.3 Exercise Summary

3.3.1 Quiz

What are the differences and relationships between the star schema, snowflake schema, and fact constellation schema in Hive?

3.3.2 Summary

This exercise describes the add, delete, modify, and query operations of Hive and introduces various join methods to help trainees understand the join types and differences. This exercise aims to help trainees better understand and use Hive.

4 ClickHouse Online Analysis Database Practices

4.1 About This Exercise

4.1.1 Overview

ClickHouse is an open-source columnar database oriented to online analysis and processing. It is independent of the Hadoop big data system and features ultimate compression rate and fast query performance. In addition, ClickHouse supports SQL query and provides good query performance, especially the aggregation analysis and query performance based on large and wide tables. The query speed is one order of magnitude faster than that of other analytical databases.

4.1.2 Objectives

Master ClickHouse cluster creation and common SQL operations of ClickHouse.

4.2 Tasks

4.2.1 Task1: Creating a ClickHouse Cluster

Step 1 Enable the MRS service and create a ClickHouse cluster.

Log in to the Huawei Cloud website, hover the cursor over **Products** and click **MapReduce Service (MRS)** to access the MRS service page. Click **Buy Now**.

Configure the basic information of the cluster as follows:

Choose **Quick Config**.

Region: Select **CN-Hong Kong**. (This document uses this region as an example.)

Billing Mode: Click **Pay-per-use**.

Cluster Name: Enter a cluster name, for example, **mrs-ch**.

Version Type: The default value is **Normal**.

Cluster Version: The default value is the latest version. This exercise uses **MRS 3.1.0** as an example.

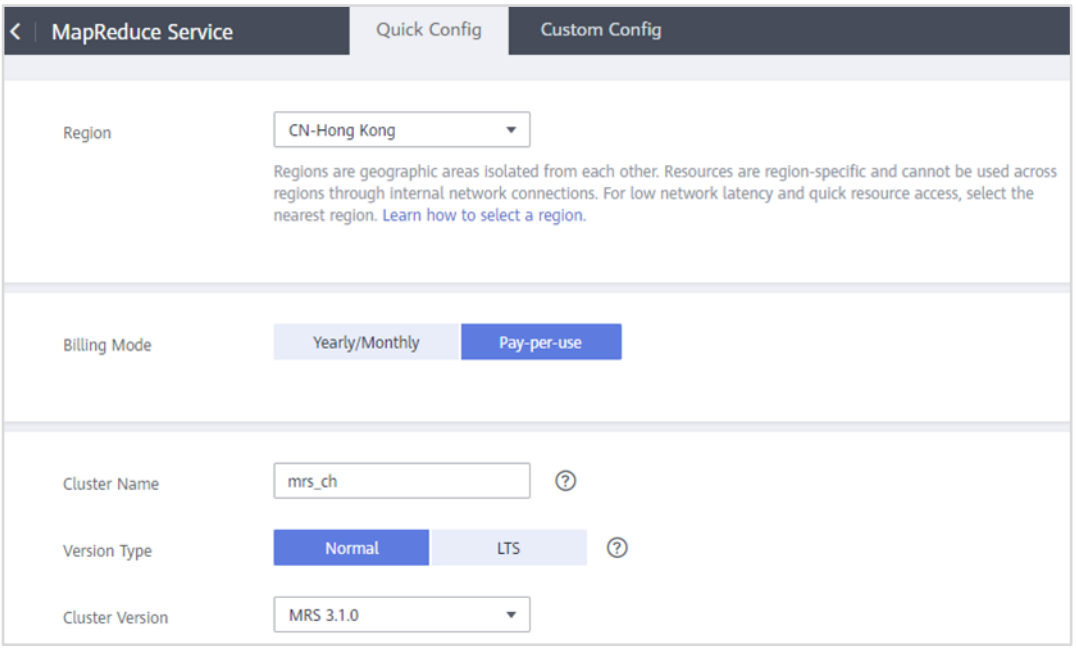


Figure 4-1

Component: Select **ClickHouse Cluster** or **Real-time Analysis Cluster**. (The two types of clusters contain ClickHouse.)

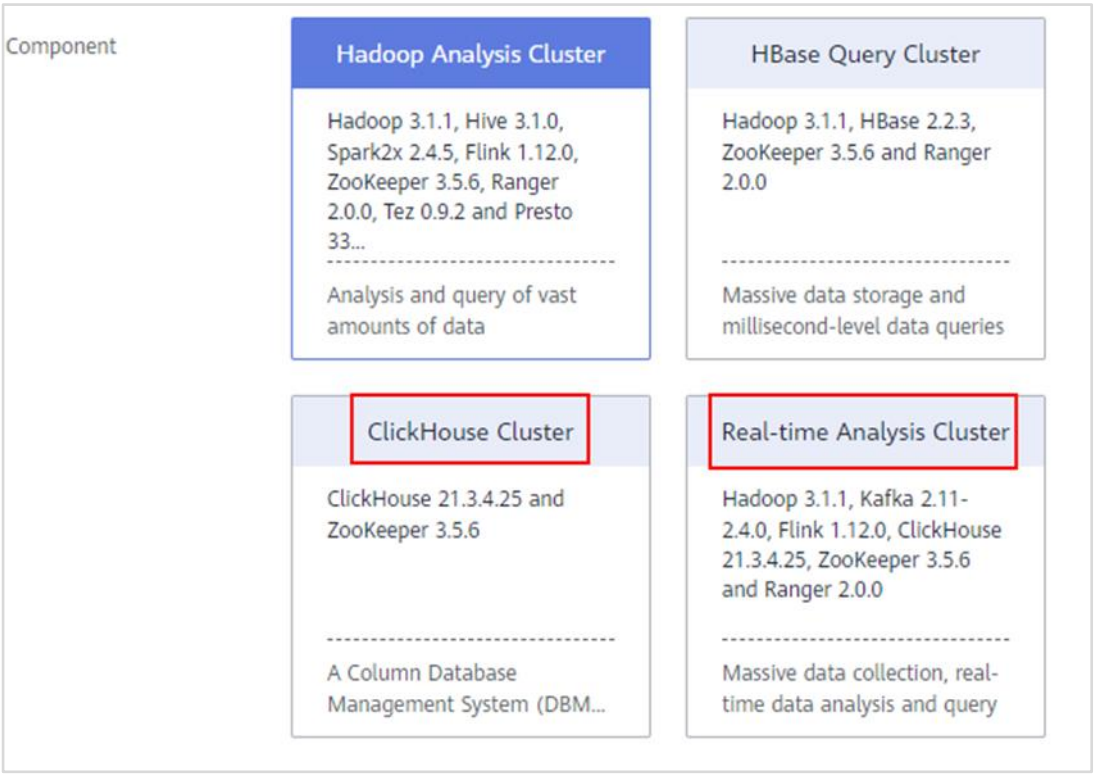


Figure 4-2

AZ: Physical region where resources use independent power supplies and networks. Use the default value.

VPC: Select a VPC. (If no VPC has been created, click **View VPC** to go to the network console. Click **Create VPC**, keep all default values unchanged, and click **Create Now**. Return to the cluster purchase page and click the refresh button for the VPC you create to be selected.)

Subnet: After a VPC is created, the subnet is automatically created and selected.

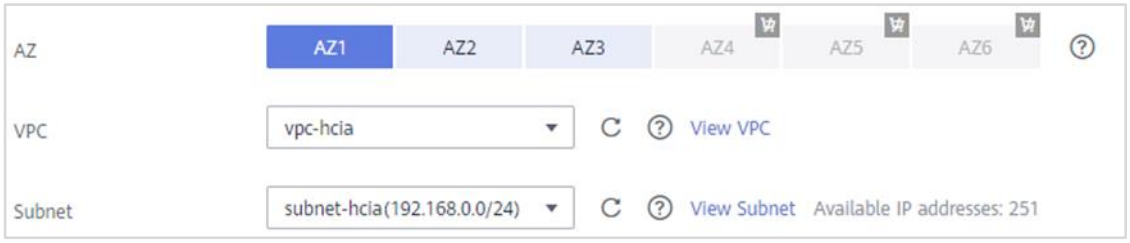


Figure 4-3

Configure instances:

Cluster Node: The master node has 16 vCPUs and 64 GB memory, and the number of instances is 3.

A ClickHouse node has 16 vCPUs and 64 GB memory, and the number of instances is 2. Keep other default values unchanged.

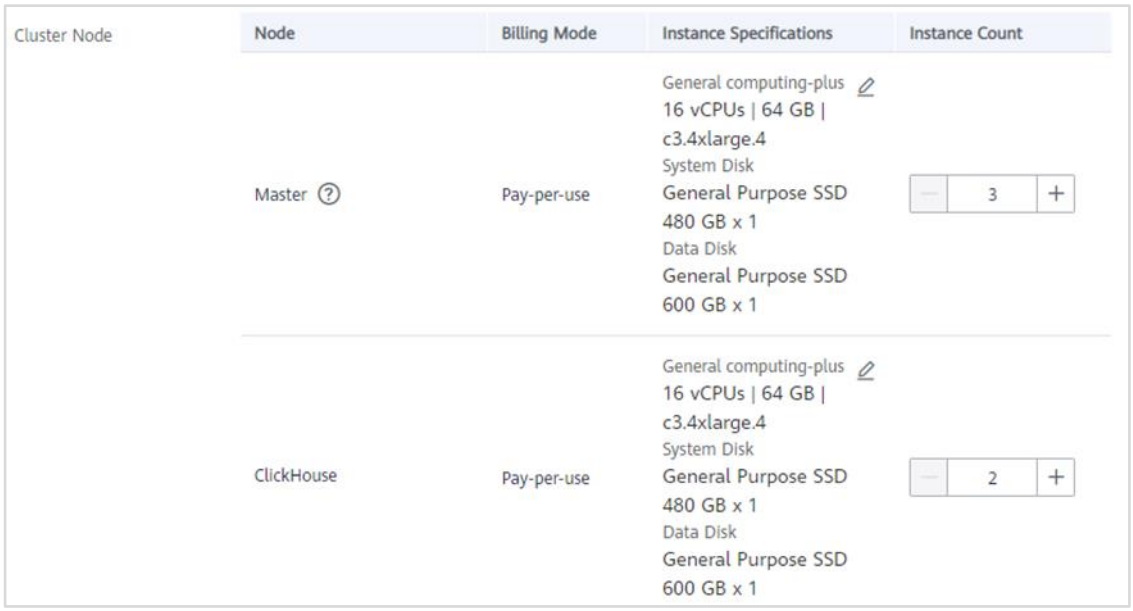


Figure 4-4

Kerberos Authentication: Disable this function.

Username: The default username is **root/admin**.

Password: Customized by the user.

Confirm Password: Enter the password again.

Secure Communications: Select **Enable**.

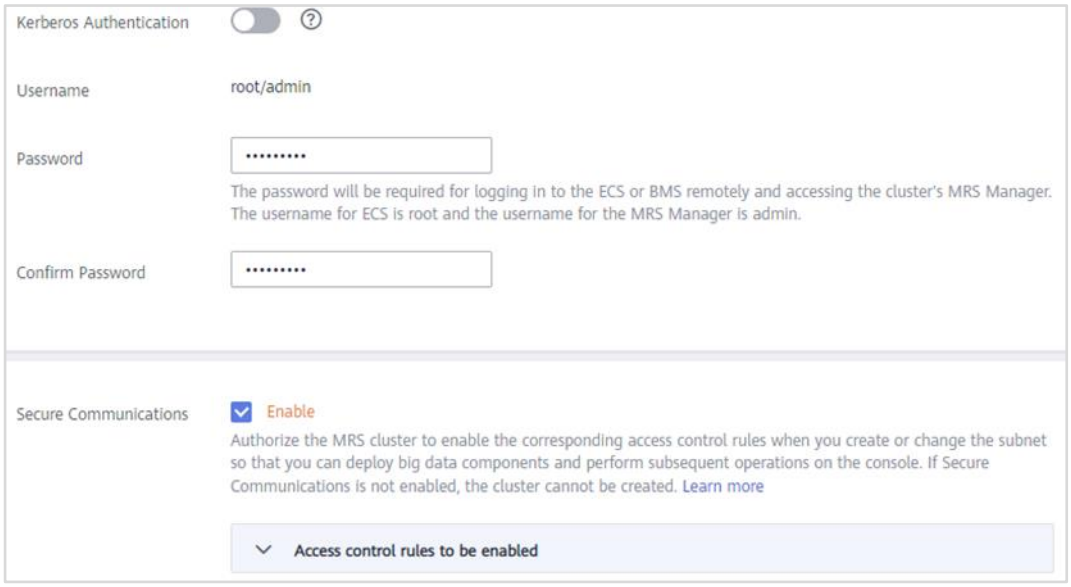


Figure 4-5

Step 2 Wait until the cluster is started.

Click **Buy Now** to return to the cluster list. You can see that the cluster is being started, and the operation takes about 20 minutes.

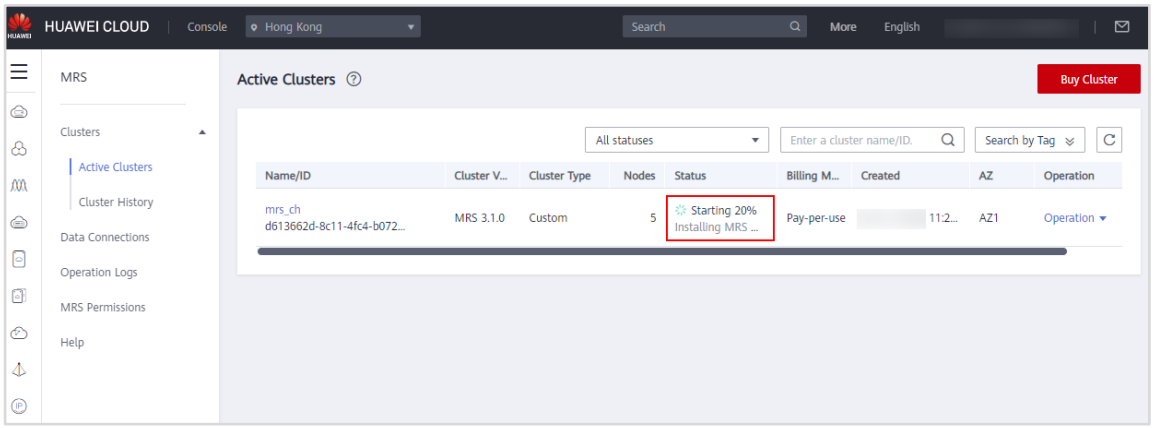


Figure 4-6

The cluster is started.

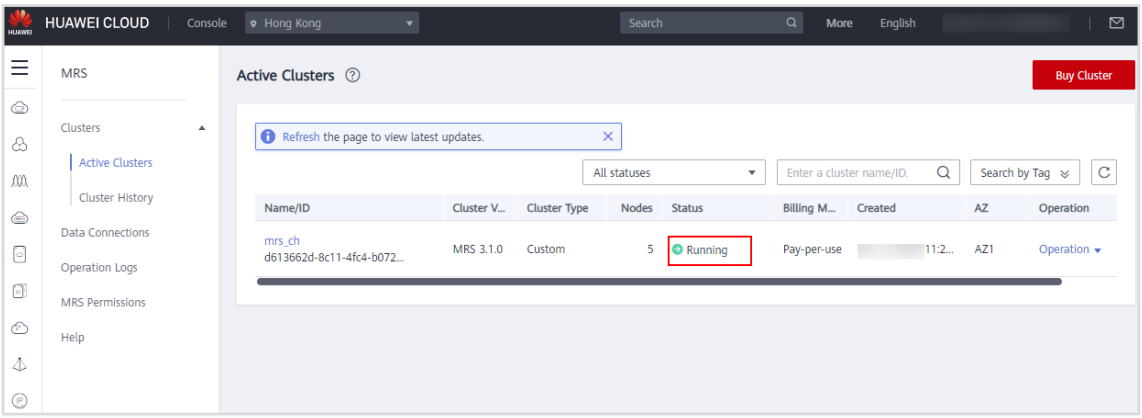


Figure 4-7

Step 3 Apply for an EIP and bind it.

Note: If you have purchased an EIP and bound it to a node when purchasing a cluster, go to step 4.

Click the cluster name to go to the cluster list page, click **Nodes**, expand the **master_node_default_group** node group and click the name of the server whose node name contains **master1** (**master1** management node). On the ECS console, click **EIPs**.

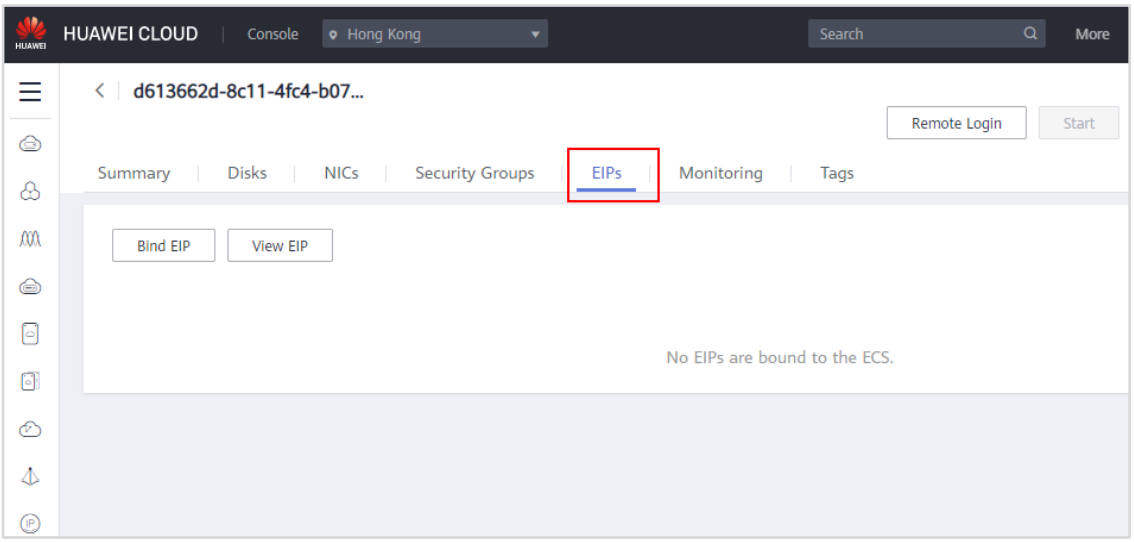


Figure 4-8

Click **Bind EIP**. If no EIP can be bound, click **Buy EIP** to go to the EIP purchase page.

On the displayed page, configure basic parameters as follows:

Billing Mode: Select **Pay-per-use**.

Region: Retain the default region, which must be the same as the region selected during the cluster purchase.

EIP Type: Select **Dynamic BGP**.

Billed By: Select **Bandwidth**.

Bandwidth (Mbit/s): Retain the default value.

Bandwidth Name: Retain the default value.

Tag: Do not configure.

Quantity: Select the number of EIPs to be purchased based on the actual requirements.

After the purchase is successful, go back to the page for binding EIPs. Click **Refresh the EIP list**, choose the default NIC, select an EIP, and click **OK**.

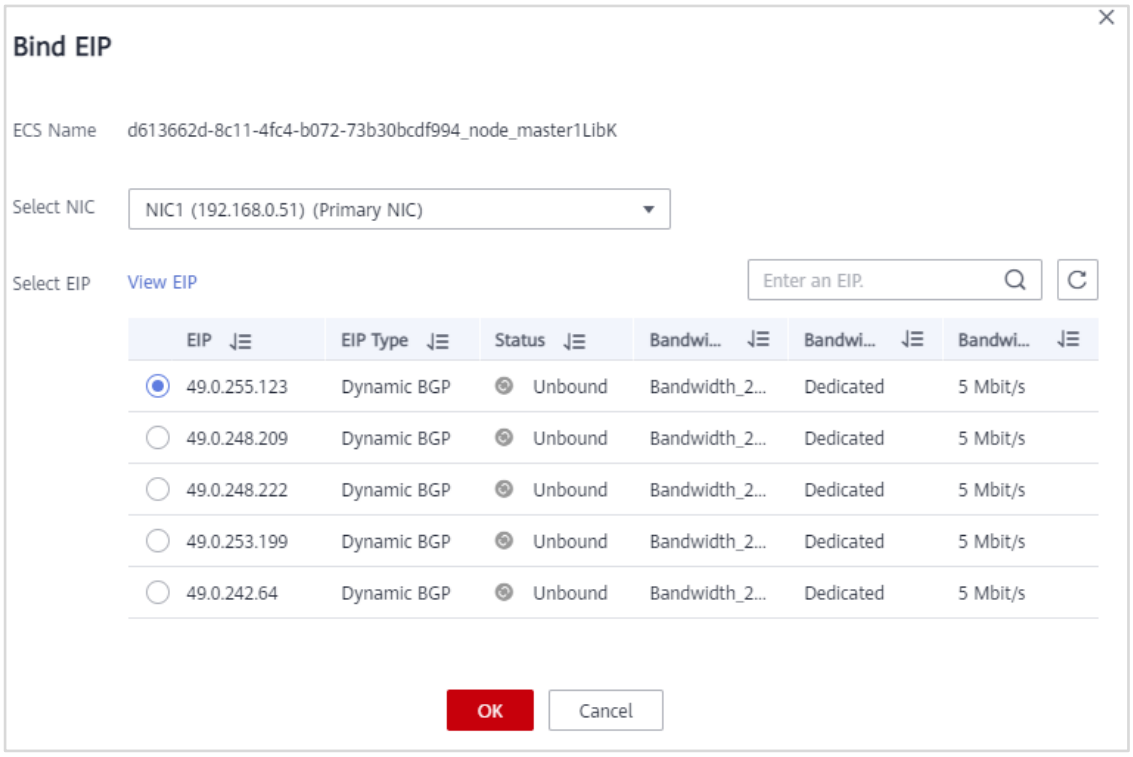


Figure 4-9

Refresh the page and you can see the EIP you select is bound to the ECS.

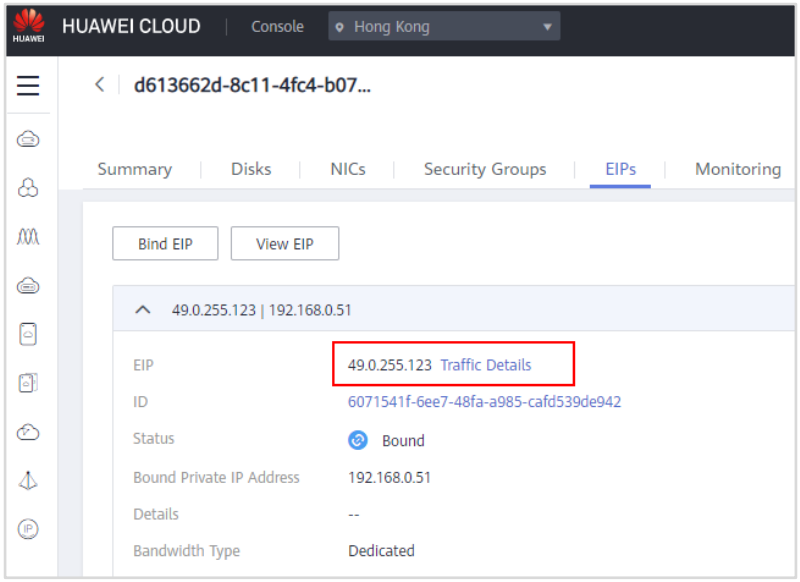


Figure 4-10

You can bind EIPs to all nodes in the cluster in sequence.

Step 4 Set the cluster security group.

Click **Security Groups**, expand the page, click **Manage Rule**, click **Inbound Rules**, and then **Add Rule**. On the displayed page, choose **Protocols** > **All**, and click **OK**.

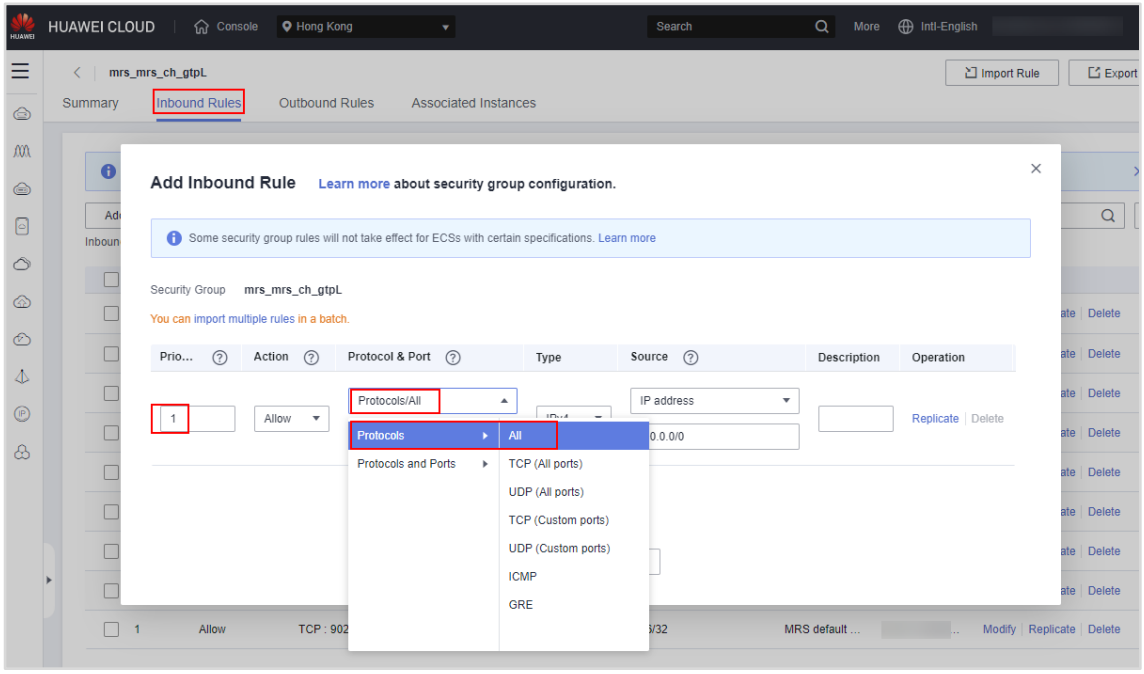


Figure 4-11

The configuration of the cluster security group is complete.

Step 5 Configure the FusionInsight Manager security group.

On the MRS cluster list page, click a cluster name. On the **Dashboard** page, click **Add Security Group Rule**, select **OK**, and click **OK**.

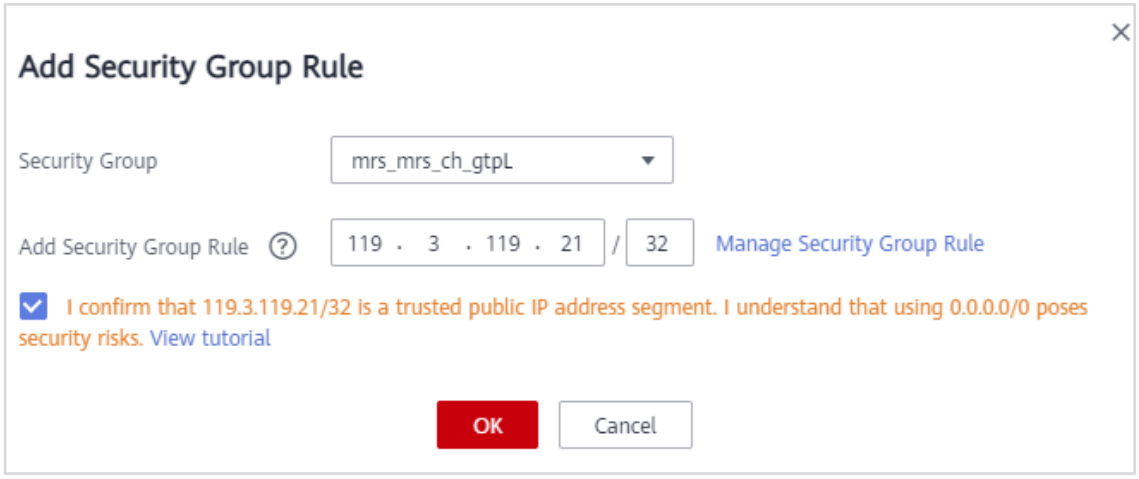


Figure 4-12

The configuration of the MRS security group is complete.

Step 6 Access the cluster management page.

On the **mrs_ch** cluster details page, click **Access Manager**. In the displayed dialog box, select the EIP (if no EIP is available, you need to purchase one), select the check box, and click **OK**. Click **Advanced** to access FusionInsight Manager. In the displayed dialog box, enter the username **admin** and its password. (The password is set during the cluster purchase.) Click **Log In** to go to the FusionInsight Manager page.

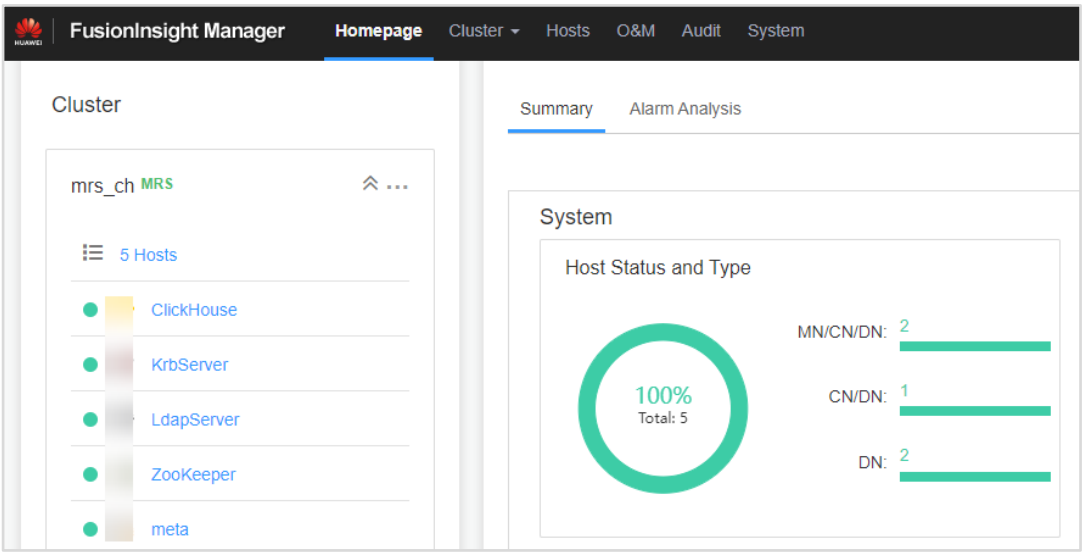


Figure 4-13

Step 7 Download a client.

Log in to FusionInsight Manager. In the cluster list, click the button next to **mrs_ch**. Click **Download Client**.

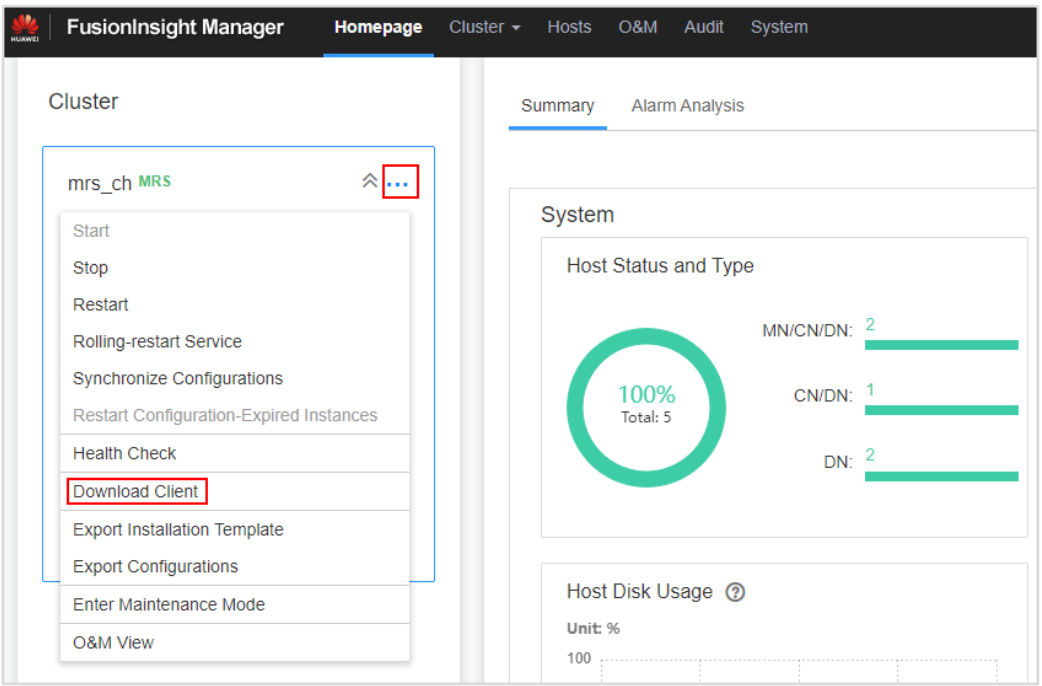


Figure 4-14

In the displayed dialog box, set **Select Client Type** to **Complete Client**. Set **Select Platform Type** to **x86_64**, select **Save to Path**, and click **OK** to generate the client file.

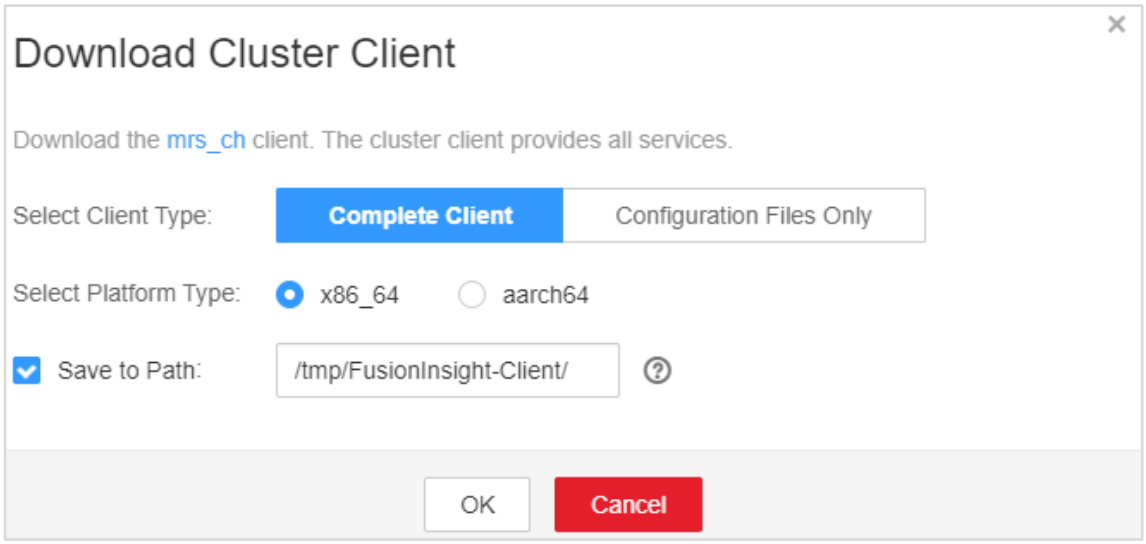


Figure 4-15

The generated file is saved in the **/tmp/FusionInsight-Client** directory on the active management node by default.

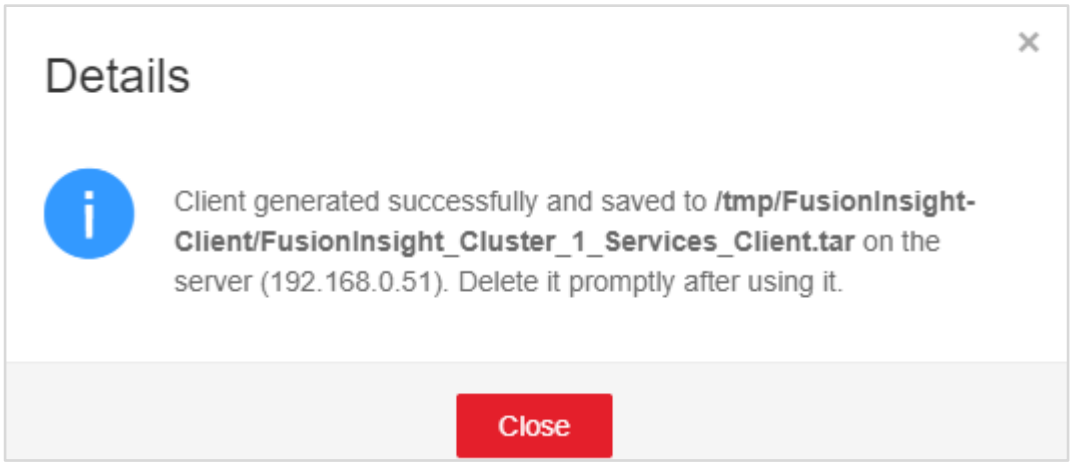


Figure 4-16

Step 8 Install the agent.

Go to the directory where the installation package is stored and run the decompression command to decompress the installation package to a local directory.

```
cd /tmp/FusionInsight-Client
tar -xvf FusionInsight_Cluster_1_Services_Client.tar
```

```
[root@node-masterlsnPA ~]# cd /tmp/FusionInsight-Client
[root@node-masterlsnPA FusionInsight-Client]# tar -xvf FusionInsight_Cluster_1_
ervices_Client.tar
FusionInsight_Cluster_1_Services_ClientConfig.tar.sha256
FusionInsight_Cluster_1_Services_ClientConfig.tar
```

Figure 4-17

Run the following command to verify the decompressed file and check whether the command output is consistent with the information in the **sha256** file.

```
sha256sum -c FusionInsight_Cluster_1_Services_ClientConfig.tar.sha256
```

```
[root@node-masterlsnPA FusionInsight-Client]# sha256sum -c FusionInsight_Cluste
r_1_Services_ClientConfig.tar.sha256
FusionInsight_Cluster_1_Services_ClientConfig.tar: OK
```

Figure 4-18

Run the following command to decompress the obtained installation file:

```
tar -xvf FusionInsight_Cluster_1_Services_ClientConfig.tar
```

Go to the directory where the installation package is stored, and run the following command to install the client to a specified directory (an absolute path), for example, **/opt/client**:

```
cd /tmp/FusionInsight-Client/FusionInsight_Cluster_1_Services_ClientConfig
./install.sh /opt/client
```

```
[root@node-masterlsnPA FusionInsight-Client]# cd /tmp/FusionInsight-Client/Fusi
onInsight_Cluster_1_Services_ClientConfig
[root@node-masterlsnPA FusionInsight_Cluster_1_Services_ClientConfig]# ./instal
l.sh /opt/client
```

Figure 4-19

If the following information is displayed, the installation is successful.

```
[                    ]: ZooKeeper installation is complete.
[                    ]: The component client is installed successfully
[                    ]: Persisting client information to the database...
[                    08]: Platform=x86_64
[                    08]: ClientIp=192.168.0.50
[                    08]: InstallUser=root
[                    08]: ClientPath=/opt/client
[                    09]: Do Save client information to database Successfully.
[root@node-masterlsnPA FusionInsight_Cluster_1_Services_ClientConfig]#
```

Figure 4-20

Step 9 Access a cluster.

Run the following command to set environment variables:

```
source /opt/client/bigdata_env
```

Run the following command to view the command help information of the ClickHouse component:

```
clickhouse -h
```

```
[root@node-masterlsnPA ~]# source /opt/client/bigdata_env
[root@node-masterlsnPA ~]# clickhouse -h
Use one of the following commands:
clickhouse local [args]
clickhouse client [args]
clickhouse benchmark [args]
clickhouse server [args]
clickhouse extract-from-config [args]
clickhouse compressor [args]
clickhouse format [args]
clickhouse copier [args]
clickhouse obfuscator [args]
clickhouse git-import [args]
clickhouse install [args]
clickhouse start [args]
clickhouse stop [args]
clickhouse status [args]
clickhouse restart [args]
clickhouse hash-binary [args]
[root@node-masterlsnPA ~]#
```

Figure 4-21

4.2.2 Task 2: Performing Common ClickHouse Operations

Step 1 Access the ClickHouse cluster.

```
clickhouse client --host 122.9.69.102 --user default --port 9000
```

```
[root@node-masterlsnPA ~]# clickhouse client --host 122.9.69.102 --user default --port 9000

ClickHouse client version 21.3.4.25.
Connecting to 122.9.69.102:9000.
Connected to ClickHouse server version 21.3.4 revision 54447.

ClickHouseFLVP0001.mrs-whuq.com :)
```

Figure 4-22

Step 2 View databases.

Run the following command to view the database of the current cluster:

```
show databases
```

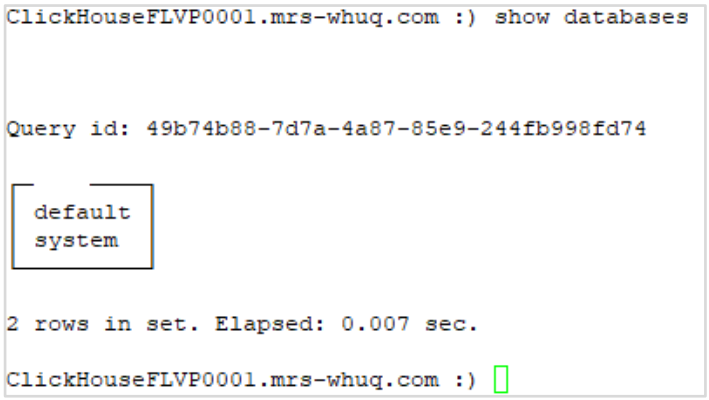


Figure 4-23

Step 3 Check the database in use.

```
select currentDatabase()
```

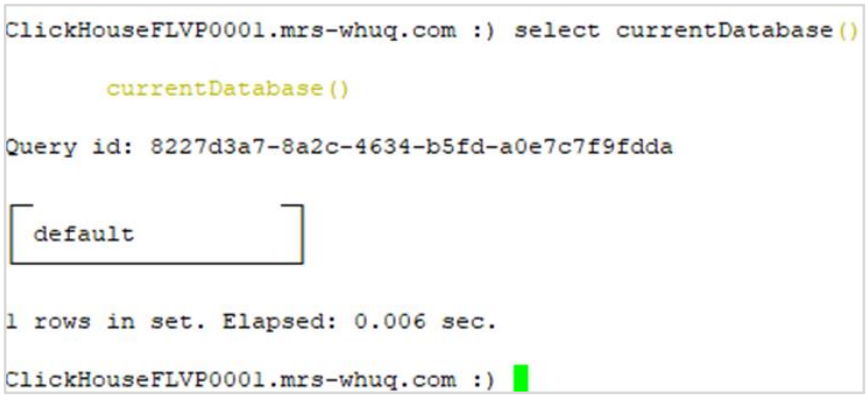


Figure 4-24

Step 4 Create a database.

```
create database is not exists test
```

```
ClickHouseFLVP0001.mrs-whuq.com :) CREATE DATABASE if not exists test

                                test

Query id: 1lee389d-65f1-46db-8720-18870c32c461

Ok.

0 rows in set. Elapsed: 0.006 sec.

ClickHouseFLVP0001.mrs-whuq.com :) show databases


Query id: a6004ad2-51a8-49d1-b00f-1b8e39d7aac0

┌─name────┐
│ default │
│ system  │
│ test    │
└─────────┘

3 rows in set. Elapsed: 0.008 sec.

ClickHouseFLVP0001.mrs-whuq.com :)
```

Figure 4-25

Step 5 Create a table.

```
create table if not exists test.t1(id UInt16,name String) ENGINE = Memory;
```

```
ClickHouseFLVP0001.mrs-whuq.com :) create table if not exists test t1(id UInt16
 name String) ENGINE   Memory

                            test.t1
(
    `id` UInt16,
    `name` String
)
       = Memory

Query id: 6df9872f-bda1-46e4-8466-1d2e2b54a7cd

Ok.

0 rows in set. Elapsed: 0.010 sec.
```

Figure 4-26

Step 6 View a table structure.

```
desc test.t1
```

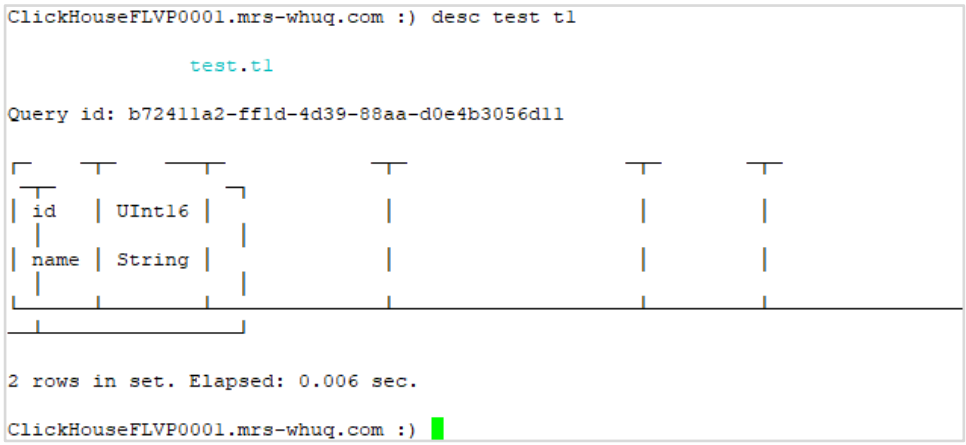


Figure 4-27

Step 7 Insert data.

Run the following command to insert two data records into the **t1** table:

```
insert into test.t1(id,name) values(1,'tom'),(2,'lily');
```

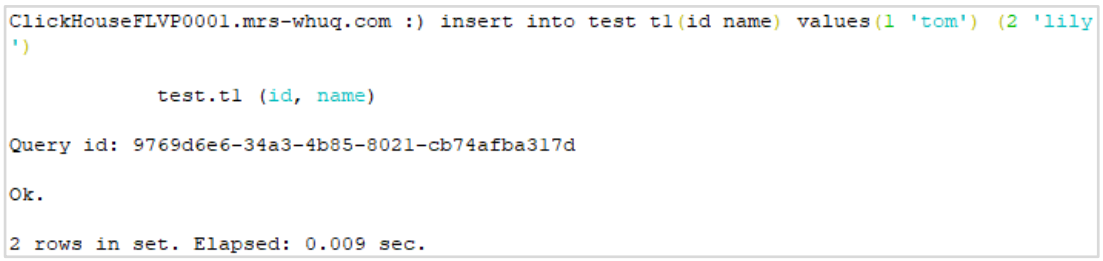


Figure 4-28

Step 8 Query the table data.

Run the following command to query the data in the **t1** table:

```
select * from test.t1
```

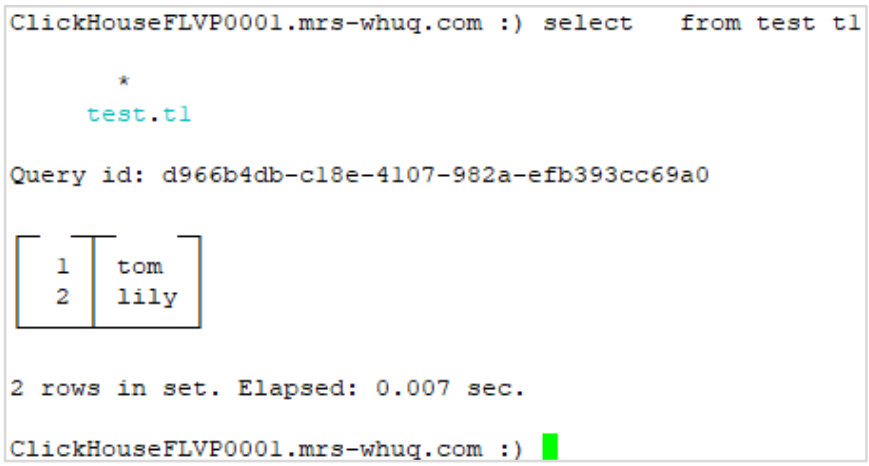


Figure 4-29

Step 9 Modify the table structure.

Run the following commands to create the **t2** table and add a column to its structure:

```
CREATE TABLE test.t2(id UInt64,eventDate DateTime) ENGINE = MergeTree() PARTITION BY
toYYYYMM(eventDate) ORDER BY id
alter table test.t2 add column cost UInt32 default 0;
```

```
ClickHouseFLVP0001.mrs-whuq.com :) CREATE TABLE test t2(id UInt64 eventDate DateTime) EN
GINE   MergeTree() PARTITION BY toYYYYMM(eventDate) ORDER BY id

            test.t2
(
    `id` UInt64,
    `eventDate` DateTime
)
       = MergeTree
            toYYYYMM(eventDate)
         id

Query id: ae5e9c85-af60-49c5-91d0-02335ad5591d

Ok.

0 rows in set. Elapsed: 0.020 sec.

ClickHouseFLVP0001.mrs-whuq.com :) desc test t2

              test.t2

Query id: b8bbda44-6b69-45c9-a563-f4c9278f8121

┌─name──────┬─type─────┬─default_type─┬─default_expression─┬─comment─┬─codec_expression─┐
│ id        │ UInt64   │              │                    │         │                  │
│ eventDate │ DateTime │              │                    │         │                  │
└───────────┴──────────┴──────────────┴────────────────────┴─────────┴──────────────────┘

2 rows in set. Elapsed: 0.009 sec.

ClickHouseFLVP0001.mrs-whuq.com :) alter table test t2 add column cost UInt32 default 0


           test.t2
              `cost` UInt32         0

Query id: 1d466262-ea6f-4fdb-8cd6-344a7a9f93b5

Ok.

0 rows in set. Elapsed: 0.012 sec.

ClickHouseFLVP0001.mrs-whuq.com :) desc test t2

              test.t2

Query id: 6d1c8da2-7829-4b8c-a1a7-1415bbf090d2

┌─name──────┬─type─────┬─default_type─┬─default_expression─┬─comment─┬─codec_expression─┐
│ id        │ UInt64   │              │                    │         │                  │
│ eventDate │ DateTime │              │                    │         │                  │
│ cost      │ UInt32   │ DEFAULT      │ 0                  │         │                  │
└───────────┴──────────┴──────────────┴────────────────────┴─────────┴──────────────────┘

3 rows in set. Elapsed: 0.007 sec.

ClickHouseFLVP0001.mrs-whuq.com :)
```

Figure 4-30

Step 10 Delete a table.

Run the following command to delete the **t1** table:

```
drop table if exists test.t1
```

```
ClickHouseFLVP0001.mrs-whuq.com :) use test

    test

Query id: 912a3e88-ebc0-46e6-bf70-93a4b3cf29da

Ok.

0 rows in set. Elapsed: 0.005 sec.

ClickHouseFLVP0001.mrs-whuq.com :) show tables


Query id: 81731a2a-d49c-4681-95fe-8d84d10945e2

┌─name─┐
│ t1   │
│ t2   │
└──────┘

2 rows in set. Elapsed: 0.007 sec.

ClickHouseFLVP0001.mrs-whuq.com :) drop table if exists test t1

                      test.t1

Query id: ecc96220-dae8-4e62-b6cc-e598d6e828fa

Ok.

0 rows in set. Elapsed: 0.006 sec.

ClickHouseFLVP0001.mrs-whuq.com :) show tables


Query id: 7f156b92-eef3-4a97-bee1-84cabe9b57ab

┌─name─┐
│ t2   │
└──────┘

1 rows in set. Elapsed: 0.007 sec.

ClickHouseFLVP0001.mrs-whuq.com :)
```

Figure 4-31

Step 11 Delete a database.

Run the following command to delete the **test** database:

```
drop database test
```

```
ClickHouseFLVP0001.mrs-whuq.com :) show databases


Query id: fc7f0cea-7d05-447e-b0a6-20767b293871

┌─name───┐
│ default │
│ system │
│ test   │
└────────┘

3 rows in set. Elapsed: 0.006 sec.

ClickHouseFLVP0001.mrs-whuq.com :) drop database test

                test

Query id: 643fedaf-27ee-4b9b-b81f-77c48f0b9294

Ok.

0 rows in set. Elapsed: 0.007 sec.

ClickHouseFLVP0001.mrs-whuq.com :) show databases


Query id: 041b0ed0-f807-4509-9cb8-82e50877ce10

┌─name───┐
│ default │
│ system │
└────────┘

2 rows in set. Elapsed: 0.008 sec.

ClickHouseFLVP0001.mrs-whuq.com :)
```

Figure 4-32

4.3 Exercise Summary

4.3.1 Quiz

Can the structure of any table be modified by using the **Alter table** keyword in ClickHouse?

4.3.2 Summary

This exercise describes how to use MRS to create a ClickHouse cluster and describes common SQL syntax of ClickHouse. After completing this exercise, you will be able to master common SQL operations of ClickHouse.

5 MapReduce Data Processing Practices

5.1 About This Exercise

5.1.1 Overview

This exercise mainly introduces how to use MapReduce to count words.

5.1.2 Objectives

Understand the principles of MapReduce programming.

5.2 Tasks

5.2.1 Task 1: MapReduce Shell Practice

Step 1 Log in to an ECS.

Use PuTTY to log in to the ECS and set environment variables.

```
source   /opt/client/bigdata_env
```

```
[root@node-masterlnFFO ~]# source  /opt/client/bigdata_env
[root@node-masterlnFFO ~]#
```

Figure 5-1

Edit a data file.

Run the following commands to edit the **cx_wd.txt** data file on the local Linux host:

```
vi cx_wd.txt
more cx_wd.txt
```

The following figure shows the file content.

```
[root@node-masterlnFFO ~]# vi cx_wd.txt
[root@node-masterlnFFO ~]# more cx_wd.txt
hadoop  hive    hadoop
hbase   spark   hive    hadoop
spark
[root@node-masterlnFFO ~]#
```

Figure 5-2

Step 2 Upload the file to HDFS.

Run the following command to upload the file to the **/user/stu01** directory of HDFS:

```
hdfs dfs -put cx_wd.txt /user/stu01
```

```
[root@node-masterlnFFO ~]# hdfs dfs -ls /user/stu01
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/
slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/com
mon/lib/slf4j-log4j12-1.7.30.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]
[root@node-masterlnFFO ~]# hdfs dfs -put cx_wd.txt /user/stu01
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/
slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/com
mon/lib/slf4j-log4j12-1.7.30.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]
[root@node-masterlnFFO ~]# hdfs dfs -ls /user/stu01
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/
slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/com
mon/lib/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 1 items
-rw-r--r--   2 root hadoop          49 [redacted] /user/stu01/cx_wd.txt
```

Figure 5-3

Step 3 Execute the JAR file program.

```
yarn jar /opt/client/Yarn/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.1-
hw-ei-310013.jar wordcount /user/stu01/cx_wd.txt /user/stu01/output01
```

```
[root@node-masterlnFFO ~]# yarn  jar  /opt/client/Yarn/hadoop/share/hadoop/mapredu
ce/hadoop-mapreduce-examples-3.1.1-hw-ei-310013.jar  wordcount  /user/stu01/cx_wd.
txt   /user/stu01/output01
WARNING: YARN_CONF_DIR has been replaced by HADOOP_CONF_DIR. Using value of YARN_C
ONF_DIR.
WARNING: YARN_ROOT_LOGGER has been replaced by HADOOP_ROOT_LOGGER. Using value of
YARN_ROOT_LOGGER.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/
slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/com
mon/lib/slf4j-log4j12-1.7.30.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]
[redacted] 10:15:34,877 INFO mapreduce.JobResourceUploader: Disabling Erasure Codi
ng for path: hdfs://hacluster/tmp/hadoop-yarn/staging/root/.staging/job_1660620439
866_0003
[redacted] 10:15:35,102 INFO input.FileInputFormat: Total input files to process :
 1
[redacted] 10:15:35,229 INFO mapreduce.JobSubmitter: number of splits:1
[redacted] 10:15:35,330 INFO Configuration.deprecation: hadoop.http.rmwebapp.sched
uler.page.classes is deprecated. Instead, use yarn.http.rmwebapp.scheduler.page.cl
ass
[redacted] 10:15:35,330 INFO Configuration.deprecation: yarn.resourcemanager.syste
m-metrics-publisher.enabled is deprecated. Instead, use yarn.system-metrics-publis
her.enabled
[redacted] 10:15:35,529 INFO mapreduce.JobSubmitter: Submitting tokens for job: jo
b_1660620439866_0003
[redacted] 10:15:35,531 INFO mapreduce.JobSubmitter: Executing with tokens: []
[redacted] 10:15:35,698 INFO conf.Configuration: found resource resource-types.xml
```

Figure 5-4

Note: This JAR package is a built-in sample JAR package of the Hadoop framework. The default file separator is the Tab key. The **output01** folder does not exist. The program automatically creates the folder.

Step 4 View the statistics result.

The result file is saved in the **output01** folder. The system automatically generates a **part-r-00000** file. Run the following commands:

```
hdfs dfs -ls /user/stu01/output01
hdfs dfs -cat /user/stu01/output01/part-r-00000
```

The following figure shows the command output.

```
[root@node-master1nFFO ~]# hdfs dfs -ls /user/stu01/output01
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/slf4j-lo
g4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/common/lib/
slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 2 items
-rw-r--r--   2 root hadoop          0 [redacted] 10:15 /user/stu01/output01/_SUCCESS
-rw-r--r--   2 root hadoop         32 [redacted] 10:15 /user/stu01/output01/part-r-00000
[root@node-master1nFFO ~]# hdfs dfs -cat /user/stu01/output01/part-r-00000
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/slf4j-lo
g4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/common/lib/
slf4j-log4j12-1.7.30.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]
hadoop  3
hbase   1
hive    2
spark   2
```

Figure 5-5

The file statistics collection are complete.

Step 5 Parse the source code of the WordCount JAR package.

```
package com.huawei.bigdata.mapreduce.examples;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
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;

public class WordCountDemo {
  public static class MyMapper extends Mapper<LongWritable, Text, Text, LongWritable>{
    @Override
    protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text,
LongWritable>.Context context)
        throws IOException, InterruptedException {
          String line = value.toString();
          String[] splited = line.split("\t");
          for (String word : splited) {
            Text k2 = new Text(word);
            LongWritable v2 = new LongWritable(1);
            context.write(k2, v2);
          }
        }
  }
  public static class MyReducer extends Reducer<Text, LongWritable, Text, LongWritable> {
    @Override
    protected void reduce(Text k2, Iterable<LongWritable> v2s,
        Reducer<Text, LongWritable, Text, LongWritable>.Context context)
        throws IOException, InterruptedException {
```

```
    long count = 0L;
    for (LongWritable times : v2s) {
      count += times.get();
    }
    LongWritable v3 = new LongWritable(count);
    context.write(k2, v3);
  }
 }
 public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  Job job = Job.getInstance(conf, WordCountDemo.class.getSimpleName());
  // Mandatory
  job.setJarByClass(WordCountDemo.class);
  // Specify the data input path.
  FileInputFormat.setInputPaths(job, args[0]);
  // Specify the path that stores the custom mapper.
  job.setMapperClass(MyMapper.class);
  // Specify the type of <k2,v2> output by the mapper.
  job.setMapOutputKeyClass(Text.class);
  job.setMapOutputValueClass(LongWritable.class);
  // Specify the path that stores the custom reducer.
  job.setReducerClass(MyReducer.class);
  // Specify the type of <k3,v3> output by the reducer.
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(LongWritable.class);
  // Specify the data output path.
  FileOutputFormat.setOutputPath(job, new Path(args[1]));
  // true indicates that information such as the running progress is displayed in a timely manner.
  job.waitForCompletion(true);
 }
}
```

--The exercise is complete.

5.2.2 Task 2: MapReduce Java Practice: Collecting Statistics on the Online Duration (Optional)

Prerequisites: The Java development environment has been installed and the MRS 3.1.0 sample project has been imported. For details, see Appendix 1.

Step 1 Import the sample project.

The following figure shows the directory structure of the imported sample project.

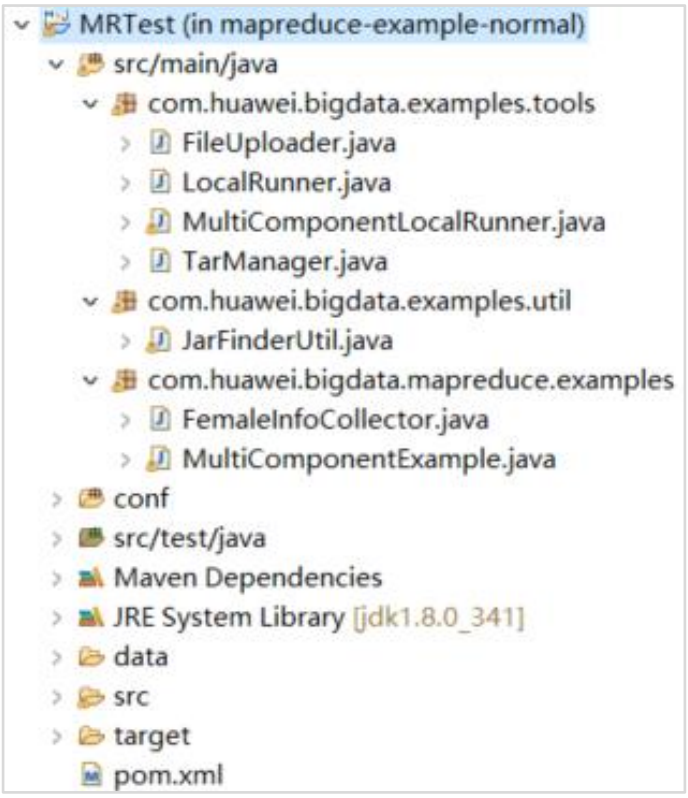


Figure 5-6

Step 2 Understand the scenario.

Develop a MapReduce application to perform the following operations on logs about time-on-page of netizens for shopping online.

1. Collect statistics on female netizens whose time-on-page for online shopping is more than 2 hours on the weekend.
2. The first column in the log file records names, the second column records gender, and the third column records the time-on-page in the unit of minute. Three columns are separated by comma (,).

log1.txt: logs collected on Saturday

```
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
FangBo,female,50
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60
```

log2.txt: logs collected on Sunday

```
LiuYang,female,20
```

```
YuanJing,male,10
CaiXuyu,female,50
FangBo,female,50
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
CaiXuyu,female,50
FangBo,female,50
LiuYang,female,20
YuanJing,male,10
FangBo,female,50
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60
```

Step 3 Plan data.

Save the original log files in HDFS.

1. Create two text files on the local host, copy the content in **log1.txt** to **cx_input_data1.txt**, and copy the content in **log2.txt** to **cx_input_data2.txt**.

```
vi   cx_input_data1.txt
cat cx_input_data1.txt
vi   cx_input_data2.txt
cat cx_input_data2.txt
```

2. Create folder **/user/stu01/input** in HDFS and upload **cx_input_data1.txt** and **cx_input_data2.txt** to the directory.

On the HDFS client of the Linux OS, run the following command to create the input folder:

```
hdfs dfs -mkdir /user/stu01/input
```

On the HDFS client of the Linux OS, run the following commands to upload files:

```
hdfs dfs -put   cx_input_data1.txt /user/stu01/input
hdfs dfs -put   cx_input_data2.txt /user/stu01/input
```

After the operation is complete, the files in the corresponding HDFS directory are as follows.

```
[root@node-master1Llmv ~]# hdfs dfs -put  cx_input_data1.txt /user/stu01/input
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/sl
f4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/commo
n/lib/slf4j-log4j12-1.7.30.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]
[root@node-master1Llmv ~]# hdfs dfs -put  cx_input_data2.txt /user/stu01/input
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/sl
f4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/commo
n/lib/slf4j-log4j12-1.7.30.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]
[root@node-master1Llmv ~]# hdfs dfs -ls /user/stu01/input
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/sl
f4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/commo
n/lib/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 2 items
-rw-r--r--   2 root hadoop        188 20[redacted]48 /user/stu01/input/cx_input_da
ta1.txt
-rw-r--r--   2 root hadoop        258 20[redacted]:48 /user/stu01/input/cx_input_da
ta2.txt
```

Figure 5-7

Step 4 Understand the development approaches.

Collect statistics on female netizens who dwell on online shopping for more than two hours at weekends.

To achieve the objective, the process is as follows:

1. Read the original file data.
2. Filter data about the time-on-page of the female netizens.
3. Summarize the total time-on-page of each female.
4. Filter information about female netizens whose time-on-page is more than two hours.

Parse the sample code. The file in the sample project is **FemaleInfoCollector.java**.

Collect statistics on female netizens who dwell on online shopping for more than two hours at weekends.

It consists of three parts:

1. Filter the time-on-page of female netizens in original files using the **CollectionMapper** class inherited from the **Mapper** abstract class.
2. Summarize the time-on-page of each female netizen, and output information about female netizens whose time-on-page is more than 2 hours using the **CollectionReducer** class inherited from the **Reducer** abstract class.
3. Use the **main** method to create a MapReduce job and submit the MapReduce job to the Hadoop cluster.

Step 5 Package the MapReduce program.

Open the cmd window, go to the directory where the project is located, and run the **mvn package** command to package the project.

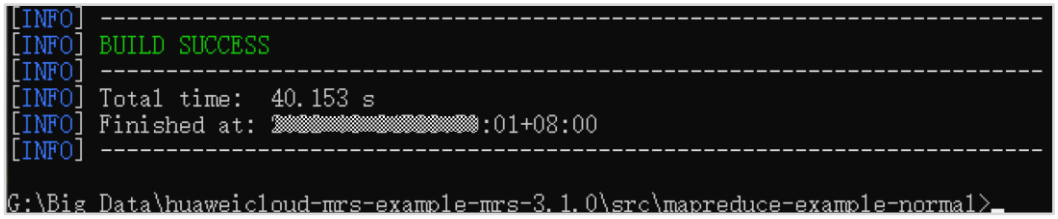


Figure 5-8

Run the **mvn package** command to generate a JAR package and obtain it from the target directory in the project directory.

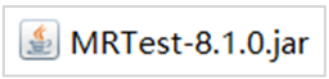


Figure 5-9

Step 6 Upload the MapReduce program to a cluster.

Use WinSCP to log in to the ECS and upload the JAR package to the **/root** directory.

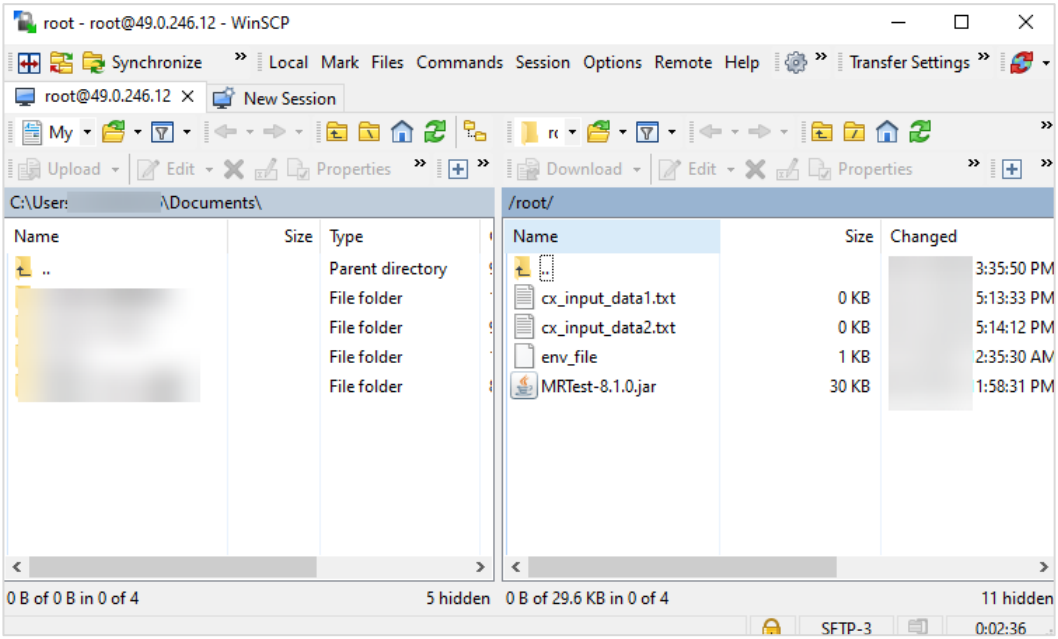


Figure 5-10

Step 7 Run the MapReduce program.

Use PuTTY to log in to the ECS and set environment variables.

```
source /opt/client/bigdata_env
```

Run the following command to run the MapReduce program:

```
yarn jar /root/MRTest-8.1.0.jar com.huawei.bigdata.mapreduce.examples.FemaleInfoCollector
/user                                                                    /stu01/input
/user/stu01/output2
```

Note: **/output2** must not exist.

```
[root@node-master1Fbkf ~]# yarn  jar  /root/MRTest-8.1.0.jar com.huawei.bigdata.mapreduce.examples.FemaleInfoCollector
/user/stu01/input  /user/stu01/output2
WARNING: YARN_CONF_DIR has been replaced by HADOOP_CONF_DIR. Using value of YARN_CONF_DIR.
WARNING: YARN_ROOT_LOGGER has been replaced by HADOOP_ROOT_LOGGER. Using value of YARN_ROOT_LOGGER.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.30.jar!/org/slf4j/im
pl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.30.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]
[redacted] 09:29:46,717 INFO client.ConfiguredRMFailoverProxyProvider: Failing over to 18
[redacted] 09:29:46,780 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: hdfs://hacluster/tmp/hado
op-yarn/staging/root/.staging/job_1661385635664_0001
[redacted] 09:29:46,994 INFO input.FileInputFormat: Total input files to process : 2
[redacted] 09:29:47,067 INFO mapreduce.JobSubmitter: number of splits:2
[redacted] 09:29:47,098 INFO Configuration.deprecation: hadoop.http.rmwebapp.scheduler.page.classes is deprecated. Inste
ad, use yarn.http.rmwebapp.scheduler.page.class
[redacted] 09:29:47,098 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is depreca
ted. Instead, use yarn.system-metrics-publisher.enabled
[redacted] 09:29:47,199 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1661385635664_0001
[redacted] 09:29:47,201 INFO mapreduce.JobSubmitter: Executing with tokens: []
[redacted] 09:29:47,355 INFO conf.Configuration: found resource resource-types.xml at file:/opt/client/Yarn/config/resou
rce-types.xml
[redacted] 09:29:47,356 INFO resource.ResourceUtils: Adding resource type - name = yarn.io/gpu, units = , type = COUNTAB
LE
[redacted] 09:29:47,753 INFO impl.YarnClientImpl: Submitted application application_1661385635664_0001
[redacted] 09:29:47,786 INFO mapreduce.Job: The url to track the job: http://node-master2WoaX.mrs-86cy.com:8088/proxy/ap
plication_1661385635664_0001/
[redacted] 09:29:47,787 INFO mapreduce.Job: Running job: job_1661385635664_0001
[redacted] 09:29:55,946 INFO mapreduce.Job: Job job_1661385635664_0001 running in uber mode : false
[redacted] 09:29:55,947 INFO mapreduce.Job:  map 0% reduce 0%
[redacted] 09:30:04,079 INFO mapreduce.Job:  map 100% reduce 0%
[redacted] 09:30:08,138 INFO mapreduce.Job:  map 100% reduce 100%
[redacted] 09:30:08,158 INFO mapreduce.Job: Job job_1661385635664_0001 completed successfully
[redacted] 09:30:08,306 INFO mapreduce.Job: Counters: 53
        File System Counters
                FILE: Number of bytes read=75
                FILE: Number of bytes written=856715
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=680
                HDFS: Number of bytes written=23
                HDFS: Number of read operations=11
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
```

Figure 5-11

Step 8 View the result.

The MapReduce output result is stored in the **/output2** directory. A result file **part-r-00000** is generated. Run the **cat** command to view the result.

```
hdfs dfs –ls /user/stu01/output2
hdfs dfs –cat /user/stu01/output2/part-r-0000
```

```
[root@node-master1Fbkf ~]# hdfs dfs -ls /user/stu01/output2
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.30.jar!/org/slf4j/im
pl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.30.jar!/org/
slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 2 items
-rw-r--r--   2 root hadoop          0 [redacted] /user/stu01/output2/_SUCCESS
-rw-r--r--   2 root hadoop         23 [redacted] /user/stu01/output2/part-r-00000
[root@node-master1Fbkf ~]# hdfs dfs -cat /user/stu01/output2/part-r-00000
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.30.jar!/org/slf4j/im
pl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.30.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]
CaiXuyu 300
FangBo  320
```

Figure 5-12

The total time-on-page of the two persons exceeds 2 hours.

5.3 Exercise Summary

5.3.1 Quiz

What is the programming model of MapReduce?

5.3.2 Summary

This exercise describes the MapReduce programming process in shell and Java modes and explains the source code to help you quickly understand MapReduce.

6 Spark In-memory Computing Practices

6.1 About This Exercise

6.1.1 Overview

Spark is implemented in the Scala language, and uses Scala as its application framework. Different from Hadoop, Spark can be tightly integrated with Scala. Scala can operate Resilient Distributed Datasets (RDDs) so easily as operating local combined objects. This exercise describes how to use Scala to operate Spark RDDs and Spark SQL.

6.1.2 Objective

Understand how to program Spark by exercising Spark RDDs and Spark SQL.

6.2 Tasks

6.2.1 Task 1: Spark RDD Programming

This exercise introduces Spark RDD programming to help you understand the working principles and core mechanism of Spark Core.

- Master how to create an RDD.
- Master common RDD operator operations.
- Understand how to use Scala project code to complete RDD operations.

Step 1 Load data from a file system to create an RDD.

Spark uses the **textFile()** method to load data from a file system to create an RDD.

This method takes the URI of the file as a parameter, which can be the address of the local file system, the address of the HDFS, the address of Amazon S3, or more.

Connect to the cluster, start PuTTY or another connection software, load environment variables, and enter spark-shell.

```
source /opt/client/bigdata_env
spark-shell
```

```
[redacted] 09:51:55,106 | WARN  | main | METASTORE_FILTER_HOOK will be ignored, si
nce hive.security.authorization.manager is set to instance of HiveAuthorizerFactor
y. | org.apache.hadoop.hive.ql.session.SessionState.setAuthorizerV2Config(SessionS
tate.java:1045)
Spark context Web UI available at http://192.168.0.172:22684
Spark context available as 'sc' (master = local[*], app id = local-1660701112129).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.5-hw-ei-310012
      /_/

Using Scala version 2.11.12 (OpenJDK 64-Bit Server VM, Java 1.8.0_272)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

Figure 6-1

Step 2 Load data from a Linux local file system.

```
scala> val lines = sc.textFile("file:///root/log1.txt")
lines.count()
lines.collect()
```

```
scala> val lines = sc.textFile("file:///root/log1.txt")
lines: org.apache.spark.rdd.RDD[String] = file:///root/log1.txt MapPartitionsRDD[3] at textFi
le at <console>:24

scala> lines.count()
[redacted] 15:56:35,932 | ERROR | main | UnsupportedOperationException: java.lang.Unsupported
OperationException: Not implemented by the RawLocalFileSystem FileSystem implementation | org
.apache.hadoop.fs.FileSystem.isBlobStore(FileSystem.java:903)
res1: Long = 2

scala> lines.collect()
res2: Array[String] = Array(hello hadoop, hello spark)

scala>
```

Figure 6-2

Step 3 Load data from HDFS. (If the file does not exist, run the **put** command to load it again.)

```
scala> val lines1 = sc.textFile("hdfs://hacluster/user/stu01/input/cx_input_data1.txt")
scala> val lines2 = sc.textFile("/user/stu01/input/cx_input_data1.txt ")
```

You can use either of them but the second one is recommended.

```
scala> val lines1 = sc.textFile("hdfs://hacluster/user/stu01/input/cx_input_data1.txt")
lines1: org.apache.spark.rdd.RDD[String] = hdfs://hacluster/user/stu01/input/cx_input_data1.t
xt MapPartitionsRDD[5] at textFile at <console>:24

scala> lines1.collect()
res3: Array[String] = Array(LiuYang,female,20, YuanJing,male,10, GuoYijun,male,5, CaiXuyu,fem
ale,50, Liyuan,male,20, FangBo,female,50, LiuYang,female,20, YuanJing,male,10, GuoYijun,male,
50, CaiXuyu,female,50, FangBo,female,60)

scala> val lines2 = sc.textFile("hdfs://hacluster/user/stu01/input/cx_input_data1.txt")
lines2: org.apache.spark.rdd.RDD[String] = hdfs://hacluster/user/stu01/input/cx_input_data1.t
xt MapPartitionsRDD[7] at textFile at <console>:24

scala> lines2.collect()
res4: Array[String] = Array(LiuYang,female,20, YuanJing,male,10, GuoYijun,male,5, CaiXuyu,fem
ale,50, Liyuan,male,20, FangBo,female,50, LiuYang,female,20, YuanJing,male,10, GuoYijun,male,
50, CaiXuyu,female,50, FangBo,female,60)

scala>
```

Figure 6-3

Step 4 Create an RDD using a parallel set (array).

You can call the **parallelize()** method of SparkContext to create an RDD on an existing set (array) in the driver.

```
scala> val array = Array(1,2,3,4,5)
array: Array[Int] = Array(1, 2, 3, 4, 5)

scala> val rdd = sc.parallelize(array)
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[14] at parallelize at <console>:26

scala> rdd.collect()
res9: Array[Int] = Array(1, 2, 3, 4, 5)

scala> val list = List(1,2,3,4,5)
list: List[Int] = List(1, 2, 3, 4, 5)

scala> val rdd = sc.parallelize(list)
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[15] at parallelize at <console>:26

scala> rdd.collect()
res10: Array[Int] = Array(1, 2, 3, 4, 5)
scala>
```

```
scala> val array = Array(1,2,3,4,5)
array: Array[Int] = Array(1, 2, 3, 4, 5)

scala> val rdd = sc.parallelize(array)
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[8] at parallelize at <console>:26

scala> rdd.collect()
res5: Array[Int] = Array(1, 2, 3, 4, 5)

scala>
```

Figure 6-4

Alternatively, you can create an RDD from the list.

```
scala> rdd.collect()
res5: Array[Int] = Array(1, 2, 3, 4, 5)

scala> val list = List(1,2,3,4,5)
list: List[Int] = List(1, 2, 3, 4, 5)

scala> val rdd = sc.parallelize(list)
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[9] at parallelize at <console>:26

scala> rdd.collect()
res6: Array[Int] = Array(1, 2, 3, 4, 5)

scala>
```

Figure 6-5

6.2.2 Task 2: RDD Shell Operations

Common transformations

Transformation	Description
map(func)	Returns a new RDD formed by passing each element of the source through a function **func**.
filter(func)	Returns a new RDD formed by selecting those elements of the source on which **func** returns **true**.
flatMap(func)	Similar to Map, but each input item can be mapped to 0 or more output items (so **func** should return a Seq instead of a single item).
mapPartitions(func)	Resembles Map but runs independently on each fragment of the RDD. Therefore, when the operator runs on the RDD whose type is **T**, the type of the **func** must be **Iterator[T] => Iterator[U]**.
mapPartitionsWithIndex(func)	Similar to mapPartitions, but also provides **func** with an integer value representing the index of the partition, so **func** must be of type **(Int, Iterator<T>) => Iterator<U>** when running on an RDD of type T.
union(otherDataset)	Returns a new RDD that contains the union of the elements in the source RDD and the argument.
intersection(otherDataset)	Returns a new RDD that contains the intersection of the elements in the source RDD and the argument.
distinct([numTasks]))	Returns a new RDD after deduplicating the source RDD.
groupByKey([numTasks])	When called on an RDD of (K, V) pairs, returns an RDD of (K, Iterable<V>) pairs.
reduceByKey(func,	When called on an RDD of (K, V) pairs, returns an RDD of (K, V) pairs where the values for each key are

Transformation	Description
[numTasks])	aggregated using the given reduce function **func**, which must be of type (V,V) => V. Like in **groupByKey**, the number of Reduce tasks is configurable through an optional second argument.
sortByKey([ascending], [numTasks])	When called on an RDD of (K, V) pairs where K implements **Ordered**, returns an RDD of (K, V) pairs sorted by keys in descending order.
sortBy(func,[ascending], [numTasks])	Similar to **sortByKey**, but more flexible.
join(otherDataset, [numTasks])	When called on RDDs of type (K, V) and (K, W), returns an RDD of (K, (V, W)) pairs with all pairs of elements for each key.
cogroup(otherDataset, [numTasks])	When called on RDDs of type (K, V) and (K, W), returns an RDD of (K, (Iterable<V>, Iterable<W>)) tuples.
union(otherDataset)	Returns a new RDD that contains the union of the elements in the source RDD and the argument.
intersection(otherDataset)	Returns a new RDD that contains the intersection of the elements in the source RDD and the argument.
distinct([numTasks]))	Returns a new RDD after deduplicating the source RDD.
groupByKey([numTasks])	When called on an RDD of (K, V) pairs, returns an RDD of (K, Iterable<V>) pairs.
reduceByKey(func, [numTasks])	When called on an RDD of (K, V) pairs, returns an RDD of (K, V) pairs where the values for each key are aggregated using the given reduce function **func**, which must be of type (V,V) => V. Like in **groupByKey**, the number of Reduce tasks is configurable through an optional second argument.
sortByKey([ascending], [numTasks])	When called on an RDD of (K, V) pairs where K implements **Ordered**, returns an RDD of (K, V) pairs sorted by keys in descending order.
coalesce(numPartitions)	Decreases the number of partitions in the RDD to **numPartitions**.
repartition(numPartitions)	Reshuffles the data in the RDD randomly to create either more or fewer partitions and balance it across them.
repartitionAndSortWithinPartiti	Repartitions the RDD according to the given

Transformation	Description
ons(partitioner)	partitioner and, within each resulting partition, sorts records by their keys.

Common actions:

Action	Description
reduce(func)	Aggregates the elements of the RDD using a function which takes two arguments and returns one.
collect()	Returns all the elements of the RDD as an array at the driver program.
count()	Returns the number of elements in the RDD.
first()	Returns the first element of the RDD, which is similar to **take(1)**.
take(n)	Returns an array consisting of the first *n* elements of a data set.
takeOrdered(n, [ordering])	Return the first *n* elements of the RDD using either their natural order or a custom comparator.
saveAsTextFile(path)	Writes the elements of the RDD as a text file (or set of text files) in a given directory in the local file system, HDFS or any other Hadoop-supported file system. Spark will call **toString** on each element to convert it to a line of text in the file.
saveAsSequenceFile(path)	Writes the elements of the RDD as a Hadoop **SequenceFile** in a given path in the local file system, HDFS or any other Hadoop-supported file system.
saveAsObjectFile(path)	Writes the elements of the RDD in a simple format using Java serialization.
countByKey()	Returns a hashmap of (K, Int) pairs with the count of each key. Only available on RDDs of type (K, V).
foreach(func)	Runs the **func** function on each element of the dataset.
foreachPartition(func)	Runs the **func** function on each partition of a data set.
reduce(func)	Aggregates the elements of the RDD using a function which takes two arguments and returns one.
collect()	Returns all the elements of the RDD as an array at

Action	Description
	the driver program.
count()	Returns the number of elements in the RDD.
first()	Returns the first element of the RDD, which is similar to **take(1)**.
take(n)	Returns an array consisting of the first *n* elements of a data set.

Step 1 Use map and filter.

Generate RDDs in parallel.

```
val rdd1 = sc.parallelize(List(5, 6, 4, 7, 3, 8, 2, 9, 1, 10))
//Multiply each element in rdd1 by 2 and sort the results.
val rdd2 = rdd1.map(_ * 2).sortBy(x => x, true)
//Filter elements greater than or equal to 5.
val rdd3 = rdd2.filter(_ >= 5)
//Display elements on the client in array mode.
rdd3.collect
```

The following figure shows the command output.

```
scala> val rdd1 = sc.parallelize(List(5, 6, 4, 7, 3, 8, 2, 9, 1, 10))
rdd1: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[10] at parallelize at <console>:2
4

scala> val rdd2 = rdd1.map(_ * 2).sortBy(x => x, true)
rdd2: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[16] at sortBy at <console>:25

scala> val rdd3 = rdd2.filter(_ >= 5)
rdd3: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[17] at filter at <console>:25

scala> rdd3.collect
res7: Array[Int] = Array(6, 8, 10, 12, 14, 16, 18, 20)

scala>
```

Figure 6-6

Step 2 Use **flatMap**.

```
val rdd1 = sc.parallelize(Array("a b c", "d e f", "h i j"))
//Divide each element in rdd1 and flatten the elements.
val rdd2 = rdd1.flatMap(_.split(" "))
rdd2.collect
```

The following figure shows the command output.

```
scala> val rdd1 = sc.parallelize(Array("a b c", "d e f", "h i j"))
rdd1: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[18] at parallelize at <console
>:24

scala> val rdd2 = rdd1.flatMap(_.split(" "))
rdd2: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[19] at flatMap at <console>:25

scala> rdd2.collect
res8: Array[String] = Array(a, b, c, d, e, f, h, i, j)

scala>
```

Figure 6-7

Step 3 Use intersection and union.

```
val rdd1 = sc.parallelize(List(5, 6, 4, 3))
val rdd2 = sc.parallelize(List(1, 2, 3, 4))
//Obtain the union set.
val rdd3 = rdd1.union(rdd2)
//Obtain the intersection.
val rdd4 = rdd1.intersection(rdd2)
//Deduplicate data.
rdd3.distinct.collect
rdd4.collect
```

The following figure shows the command output.

```
scala> val rdd1 = sc.parallelize(List(5, 6, 4, 3))
rdd1: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[20] at parallelize at <console>:2
4

scala> val rdd2 = sc.parallelize(List(1, 2, 3, 4))
rdd2: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[21] at parallelize at <console>:2
4

scala> val rdd3 = rdd1.union(rdd2)
rdd3: org.apache.spark.rdd.RDD[Int] = UnionRDD[22] at union at <console>:27

scala> val rdd4 = rdd1.intersection(rdd2)
rdd4: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[28] at intersection at <console>:27

scala> rdd3.distinct.collect
res9: Array[Int] = Array(1, 2, 3, 4, 5, 6)

scala> rdd4.collect
```

Figure 6-8

```
res10: Array[Int] = Array(3, 4)

scala>
```

Figure 6-9

Step 4 Use join and **groupByKey**.

```
val rdd1 = sc.parallelize(List(("tom", 1), ("jerry", 3), ("kitty", 2)))
val rdd2 = sc.parallelize(List(("jerry", 2), ("tom", 1), ("shuke", 2)))
//Obtain the join.
val rdd3 = rdd1.join(rdd2)
rdd3.collect
//Obtain the union set.
val rdd4 = rdd1 union rdd2
```

```
rdd4.collect
//Group by key.
val rdd5=rdd4.groupByKey
rdd5.collect
```

The results are as follows.

```
scala> val rdd1 = sc.parallelize(List(("tom", 1), ("jerry", 3), ("kitty", 2)))
rdd1: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[32] at parallelize at <
console>:24

scala> val rdd2 = sc.parallelize(List(("jerry", 2), ("tom", 1), ("shuke", 2)))
rdd2: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[33] at parallelize at <
console>:24

scala> val rdd3 = rdd1.join(rdd2)
rdd3: org.apache.spark.rdd.RDD[(String, (Int, Int))] = MapPartitionsRDD[36] at join at <conso
le>:27

scala> rdd3.collect
```

```
scala> val rdd4 = rdd1 union rdd2
rdd4: org.apache.spark.rdd.RDD[(String, Int)] = UnionRDD[37] at union at <console>:27

scala> rdd4.collect
res12: Array[(String, Int)] = Array((tom,1), (jerry,3), (kitty,2), (jerry,2), (tom,1), (shuke
,2))

scala> val rdd5=rdd4.groupByKey
rdd5: org.apache.spark.rdd.RDD[(String, Iterable[Int])] = ShuffledRDD[38] at groupByKey at <c
onsole>:25

scala> rdd5.collect
res13: Array[(String, Iterable[Int])] = Array((tom,CompactBuffer(1, 1)), (shuke,CompactBuffer
(2)), (kitty,CompactBuffer(2)), (jerry,CompactBuffer(3, 2)))

scala>
```

Figure 6-10

Step 5 Use **cogroup**.

```
val rdd1 = sc.parallelize(List(("tom", 1), ("tom", 2), ("jerry", 3), ("kitty", 2)))
val rdd2 = sc.parallelize(List(("jerry", 2), ("tom", 1), ("jim", 2)))
//cogroup
val rdd3 = rdd1.cogroup(rdd2)
//Pay attention to the difference between cogroup and groupByKey.
rdd3.collect
```

```
scala> val rdd1 = sc.parallelize(List(("tom", 1), ("tom", 2), ("jerry", 3), ("kitty", 2)))
rdd1: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[39] at parallelize at <
console>:24

scala> val rdd2 = sc.parallelize(List(("jerry", 2), ("tom", 1), ("jim", 2)))
rdd2: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[40] at parallelize at <
console>:24

scala> val rdd3 = rdd1.cogroup(rdd2)
rdd3: org.apache.spark.rdd.RDD[(String, (Iterable[Int], Iterable[Int]))] = MapPartitionsRDD[4
2] at cogroup at <console>:27

scala> rdd3.collect
```

Figure 6-11

```
res14: Array[(String, (Iterable[Int], Iterable[Int]))] = Array((tom,(CompactBuffer(1, 2),CompactBuffer(1))), (kitty,(CompactBuffer(2),CompactBuffer())), (jim,(CompactBuffer(),CompactBuffer(2))), (jerry,(CompactBuffer(3),CompactBuffer(2))))

scala>
```

Figure 6-12

Step 6 Use Reduce.

```
val rdd1 = sc.parallelize(List(1, 2, 3, 4, 5))
//Reduce aggregation.
val rdd2 = rdd1.reduce(_ + _)
rdd2
```

```
scala> val rdd1 = sc.parallelize(List(1, 2, 3, 4, 5))
rdd1: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[43] at parallelize at <console>:24

scala> val rdd2 = rdd1.reduce(_ + _)
rdd2: Int = 15

scala>
```

Figure 6-13

Step 7 Use **reduceByKey** and **sortByKey**.

```
val rdd1 = sc.parallelize(List(("tom", 1), ("jerry", 3), ("kitty", 2),  ("shuke", 1)))
val rdd2 = sc.parallelize(List(("jerry", 2), ("tom", 3), ("shuke", 2), ("kitty", 5)))
val rdd3 = rdd1.union(rdd2)
//Aggregate by key.
val rdd4 = rdd3.reduceByKey(_ + _)
rdd4.collect
//Sort by value in descending order.
val rdd5 = rdd4.map(t => (t._2, t._1)).sortByKey(false).map(t => (t._2, t._1))
rdd5.collect
```

```
scala> val rdd1 = sc.parallelize(List(("tom", 1), ("jerry", 3), ("kitty", 2),  ("shuke", 1)))
rdd1: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[32] at parallelize at <console>:24

scala> val rdd2 = sc.parallelize(List(("jerry", 2), ("tom", 3), ("shuke", 2), ("kitty", 5)))
rdd2: org.apache.spark.rdd.RDD[(String, Int)] = ParallelCollectionRDD[33] at parallelize at <console>:24

scala> val rdd3 = rdd1.union(rdd2)
rdd3: org.apache.spark.rdd.RDD[(String, Int)] = UnionRDD[34] at union at <console>:27

scala> // Aggregate by key.

scala> val rdd4 = rdd3.reduceByKey(_ + _)
rdd4: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[35] at reduceByKey at <console>:25

scala> rdd4.collect
res13: Array[(String, Int)] = Array((tom,4), (shuke,3), (kitty,7), (jerry,5))

scala> // Sort by value in descending order.

scala> val rdd5 = rdd4.map(t => (t._2, t._1)).sortByKey(false).map(t => (t._2, t._1))
rdd5: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[40] at map at <console>:25

scala> rdd5.collect
res14: Array[(String, Int)] = Array((kitty,7), (jerry,5), (tom,4), (shuke,3))

scala>
```

Figure 6-14

Step 8 Understand the lazy mechanism.

The lazy mechanism means that the entire transformation process only records the track of the transformation and does not trigger real calculation. Only when an operation is performed, real calculation is triggered from the beginning to the end.

A simple statement is provided to explain the lazy mechanism of Spark. The **data.txt** file does not exist, but the first two statements are executed successfully. An error occurs only when the third action statement is executed.

```
scala> val  lines = sc.textFile("data.txt")
scala> val  lineLengths = lines.map(s => s.length)
scala> val  totalLength = lineLengths.reduce((a, b) => a + b)
```

```
scala> val  lines = sc.textFile("data.txt")
lines: org.apache.spark.rdd.RDD[String] = data.txt MapPartitionsRDD[42] at textFile at <console>:24

scala> val  lineLengths = lines.map(s => s.length)
lineLengths: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[43] at map at <console>:25

scala> val  totalLength = lineLengths.reduce((a, b) => a + b)
org.apache.hadoop.mapred.InvalidInputException: Input path does not exist: hdfs://hacluster/user/root/data.txt
  at org.apache.hadoop.mapred.FileInputFormat.singleThreadedListStatus(FileInputFormat.java:297)
  at org.apache.hadoop.mapred.FileInputFormat.listStatus(FileInputFormat.java:239)
  at org.apache.hadoop.mapred.FileInputFormat.getSplits(FileInputFormat.java:325)
  at org.apache.spark.rdd.HadoopRDD.getPartitions(HadoopRDD.scala:200)
  at org.apache.spark.rdd.RDD$$anonfun$partitions$2.apply(RDD.scala:253)
  at org.apache.spark.rdd.RDD$$anonfun$partitions$2.apply(RDD.scala:251)
  at scala.Option.getOrElse(Option.scala:121)
  at org.apache.spark.rdd.RDD.partitions(RDD.scala:251)
  at org.apache.spark.rdd.MapPartitionsRDD.getPartitions(MapPartitionsRDD.scala:46)
  at org.apache.spark.rdd.RDD$$anonfun$partitions$2.apply(RDD.scala:253)
  at org.apache.spark.rdd.RDD$$anonfun$partitions$2.apply(RDD.scala:251)
  at scala.Option.getOrElse(Option.scala:121)
  at org.apache.spark.rdd.RDD.partitions(RDD.scala:251)
  at org.apache.spark.rdd.MapPartitionsRDD.getPartitions(MapPartitionsRDD.scala:46)
  at org.apache.spark.rdd.RDD$$anonfun$partitions$2.apply(RDD.scala:253)
  at org.apache.spark.rdd.RDD$$anonfun$partitions$2.apply(RDD.scala:251)
  at scala.Option.getOrElse(Option.scala:121)
  at org.apache.spark.rdd.RDD.partitions(RDD.scala:251)
  at org.apache.spark.SparkContext.runJob(SparkContext.scala:2137)
  at org.apache.spark.rdd.RDD$$anonfun$reduce$1.apply(RDD.scala:1035)
  at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:151)
  at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:112)
  at org.apache.spark.rdd.RDD.withScope(RDD.scala:363)
  at org.apache.spark.rdd.RDD.reduce(RDD.scala:1017)
  ... 49 elided

scala>
```

Figure 6-15

Step 9 Perform persistence operations.

The following is an example of calculating the same RDD for multiple times:

```
scala> val  list = List("Hadoop","Spark","Hive")
list: List[String] = List(Hadoop, Spark, Hive)
scala> val  rdd = sc.parallelize(list)
rdd: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[22] at parallelize at <console>:29
scala> println(rdd.count())
//Action operation, which triggers a real start-to-end calculation.
3
scala> println(rdd.collect().mkString(","))
//Action operation, which triggers a real start-to-end calculation.
Hadoop,Spark,Hive
```

```
scala> val  list = List("Hadoop","Spark","Hive")
list: List[String] = List(Hadoop, Spark, Hive)

scala> val  rdd = sc.parallelize(list)
rdd: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[44] at parallelize at <console>
:26

scala> println(rdd.count())
3

scala>  println(rdd.collect().mkString(","))
Hadoop,Spark,Hive

scala>
```

Figure 6-16

After the preceding instance is added, the execution process after a persistence statement is added is as follows:

```
scala> val  list = List("Hadoop","Spark","Hive")
list: List[String] = List(Hadoop, Spark, Hive)
scala> val  rdd = sc.parallelize(list)
rdd: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[22] at parallelize at <console>:29
scala> rdd.cache()
//persist(MEMORY_ONLY) is called. However, when the statement is executed, the RDD is not cached because the RDD has not been calculated and generated.
scala> println(rdd.count())
//The first action triggers a real start-to-end calculation. In this case, the preceding rdd.cache() is executed and the RDD is stored in the cache.
3
scala> println(rdd.collect().mkString(","))
//The second action does not need to trigger a start-to-end calculation. Only the RDD in the cache needs to be reused.
Hadoop,Spark,Hive
```

```
scala>  val  list = List("Hadoop","Spark","Hive")
list: List[String] = List(Hadoop, Spark, Hive)

scala> val  rdd = sc.parallelize(list)
rdd: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[45] at parallelize at <console>
:26

scala> rdd.cache()
res17: rdd.type = ParallelCollectionRDD[45] at parallelize at <console>:26

scala> println(rdd.count())
3

scala> println(rdd.collect().mkString(","))
Hadoop,Spark,Hive

scala>
```

Figure 6-17

6.2.3 Task 3: RDD Code Programming — Java Programming (Optional)

Step 1 Understand the scenario description.

Same as the MapReduce exercise background, this exercise requires you to calculate the time-on-page.

Develop a Spark application to perform the following operations on logs about the time-on-page of netizens for online shopping on a weekend:

1. Collect statistics on female netizens whose time-on-page for online shopping is more than 2 hours on the weekend.
2. The first column in the log file records names, the second column records gender, and the third column records the time-on-page in the unit of minute. Three columns are separated by comma (,).

log1.txt: logs collected on Saturday.

```
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
FangBo,female,50
LiuYang,female,20
YuanJing,male,10
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60
```

log2.txt: logs collected on Sunday.

```
LiuYang,female,20
YuanJing,male,10
CaiXuyu,female,50
FangBo,female,50
GuoYijun,male,5
CaiXuyu,female,50
Liyuan,male,20
CaiXuyu,female,50
FangBo,female,50
LiuYang,female,20
YuanJing,male,10
FangBo,female,50
GuoYijun,male,50
CaiXuyu,female,50
FangBo,female,60
```

Step 2 Plan data.

Upload two Internet access log files to the **/user/stu01/input** directory of the HDFS file system. (Same as the MapReduce exercise.)

Note: If the files already exist, you do not need to upload them.

Step 3 Start a Spark sample project.

Based on the MRS 3.0.1 sample project imported during environment installation, open the **FemaleInfoCollection** project. The folder in the MRS 3.0.1 sample project package is **SparkJavaExample**.

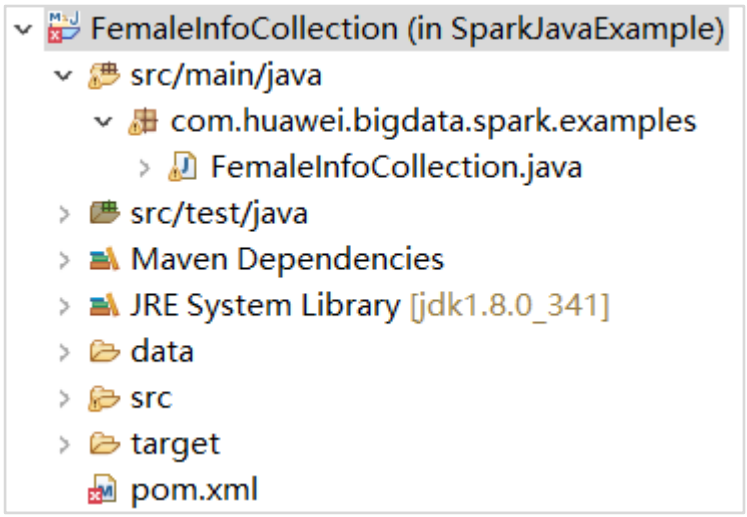


Figure 6-18

Step 4 Package the project.

Open the cmd window, go to the directory where the project is located, and run the **mvn package** command to package the project.

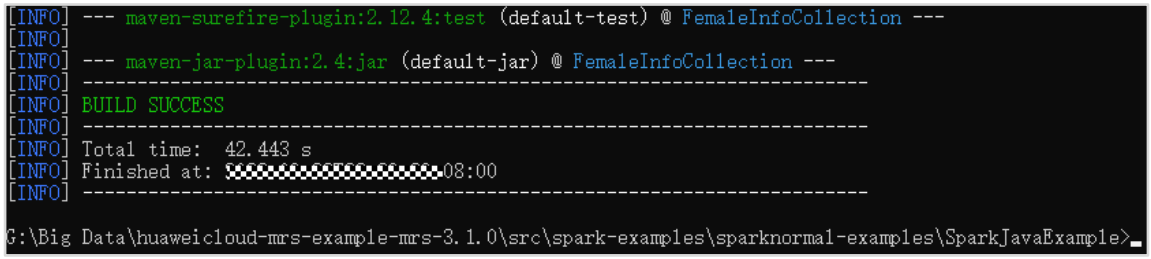


Figure 6-19

Step 5 Use WinSCP to log in to an ECS.

Upload the JAR file to the **/root** directory.

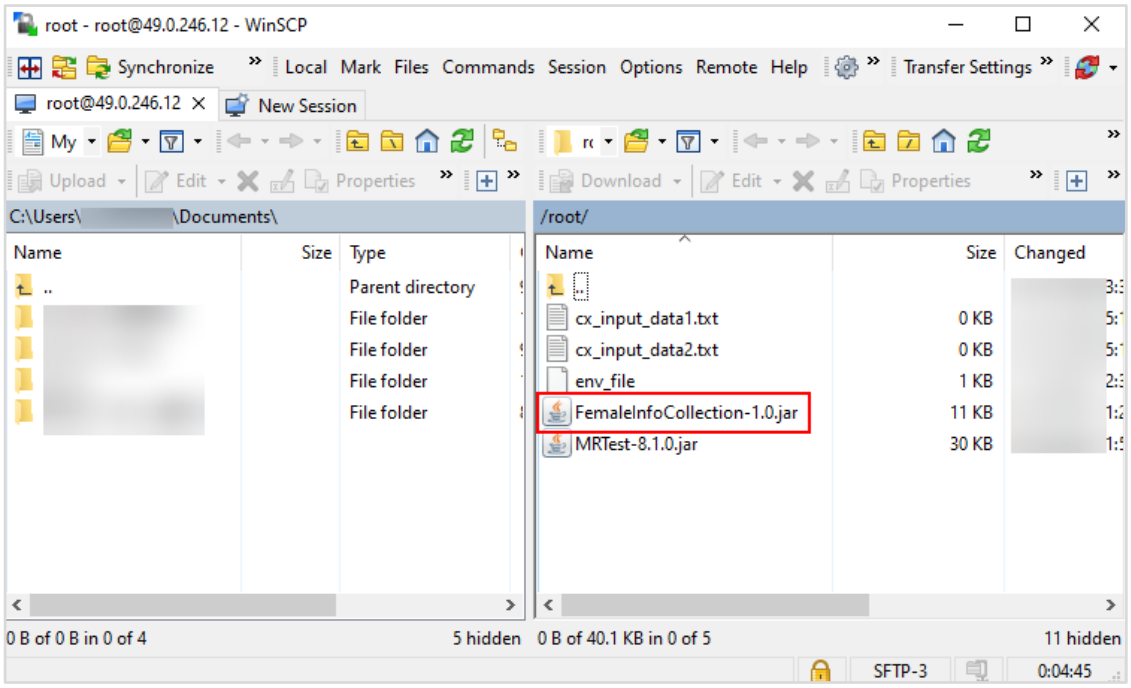


Figure 6-20

Step 6 Use PuTTY to log in to the ECS and run the Spark program.

Use PuTTY to log in to the ECS and set environment variables.

```
source /opt/client/bigdata_env
```

Run the Spark program.

```
/opt/client/Spark2x/spark/bin/spark-submit --class
com.huawei.bigdata.spark.examples.FemaleInfoCollection --master yarn --deploy-mode client
/root/FemaleInfoCollection-1.0.jar /user/stu01/input
```

```
[root@node-master1nxXi ~]# /opt/client/Spark2x/spark/bin/spark-submit  --class com.huawei.bigdata.spa
rk.examples.FemaleInfoCollection  --master  yarn  --deploy-mode  client  /root/FemaleInfoCollection-1
.0.jar  /user/stu01/input
[redacted],945 | WARN  | main | The configuration key 'spark.reducer.maxReqSizeShuffleToMem'
 has been deprecated as of Spark 2.3 and may be removed in the future. Please use the new key 'spark.
maxRemoteBlockSizeFetchToMem' instead. | org.apache.spark.SparkConf.logWarning(Logging.scala:66)
[redacted],514 | WARN  | main | The configuration key 'spark.reducer.maxReqSizeShuffleToMem'
 has been deprecated as of Spark 2.3 and may be removed in the future. Please use the new key 'spark.
maxRemoteBlockSizeFetchToMem' instead. | org.apache.spark.SparkConf.logWarning(Logging.scala:66)
[redacted],540 | WARN  | main | Note that spark.local.dir will be overridden by the value se
t by the cluster manager (via SPARK_LOCAL_DIRS in mesos/standalone/kubernetes and LOCAL_DIRS in YARN)
. | org.apache.spark.SparkConf.logWarning(Logging.scala:66)
[redacted]5,545 | WARN  | main | Detected deprecated memory fraction settings: [spark.shuffle
.memoryFraction, spark.storage.memoryFraction, spark.storage.unrollFraction]. As of Spark 1.6, execut
ion and storage memory management are unified. All memory fractions used in the old model are now dep
recated and no longer read. If you wish to use the old memory management, you may explicitly enable `
spark.memory.useLegacyMode` (not recommended). | org.apache.spark.SparkConf.logWarning(Logging.scala:
66)
[redacted]6,078 | WARN  | main | The configuration key 'spark.reducer.maxReqSizeShuffleToMem'
 has been deprecated as of Spark 2.3 and may be removed in the future. Please use the new key 'spark.
maxRemoteBlockSizeFetchToMem' instead. | org.apache.spark.SparkConf.logWarning(Logging.scala:66)
[redacted]8:56,083 | WARN  | main | The configuration key 'spark.reducer.maxReqSizeShuffleToMem'
 has been deprecated as of Spark 2.3 and may be removed in the future. Please use the new key 'spark.
maxRemoteBlockSizeFetchToMem' instead. | org.apache.spark.SparkConf.logWarning(Logging.scala:66)
[redacted]:56,085 | WARN  | main | The configuration key 'spark.reducer.maxReqSizeShuffleToMem'
 has been deprecated as of Spark 2.3 and may be removed in the future. Please use the new key 'spark.
maxRemoteBlockSizeFetchToMem' instead. | org.apache.spark.SparkConf.logWarning(Logging.scala:66)
[redacted]8:58,175 | WARN  | main | spark.yarn.security.credentials.hbase.enabled is deprecated.
  Please use spark.security.credentials.hbase.enabled instead. | org.apache.spark.deploy.security.Had
oopDelegationTokenManager.logWarning(Logging.scala:66)
[redacted]8:58,180 | WARN  | main | spark.yarn.security.credentials.hbase.enabled is deprecated.
  Please use spark.security.credentials.hbase.enabled instead. | org.apache.spark.deploy.security.Had
oopDelegationTokenManager.logWarning(Logging.scala:66)
Hive Session ID = 4a099abf-d26e-4d30-a551-68af02350f0e
[redacted]9,528 | WARN  | main | load mapred-default.xml, HIVE_CONF_DIR env not found! | org.
apache.hadoop.hive.ql.session.SessionState.loadMapredDefaultXml(SessionState.java:1460)
[redacted]9,612 | WARN  | main | METASTORE_FILTER_HOOK will be ignored, since hive.security.a
uthorization.manager is set to instance of HiveAuthorizerFactory. | org.apache.hadoop.hive.ql.session
.SessionState.setAuthorizerV2Config(SessionState.java:1045)
[redacted]:11,815 | WARN  | main | The enable mv value "null" is invalid. Using the default val
ue "true" | org.apache.carbondata.core.util.CarbonProperties.validateEnableMV(CarbonProperties.java:5
11)
[redacted]9:11,827 | WARN  | main | The value "LOCALLOCK" configured for key carbon.lock.type is
 invalid for current file system. Use the default value HDFSLOCK instead. | org.apache.carbondata.cor
e.util.CarbonProperties.validateAndConfigureLockType(CarbonProperties.java:440)
CaiXuyu,300
FangBo,320
[root@node-master1nxXi ~]#
```

Figure 6-21

The total time-on-page of the two persons exceeds 2 hours.

6.2.4 Task 4: Spark SQL DataFrame Programming

In versions earlier than Spark 2.0, SQLContext in Spark SQL is the entry for creating DataFrames and executing SQL statements. You can use HiveContext to operate Hive table data through Hive SQL statements. HiveContext is compatible with Hive operations and is inherited from SQLContext. In versions later than Spark 2.0, all these functions are integrated into SparkSession. SparkSession encapsulates SparkContext and SQLContext, and can obtain SparkConetxt and SQLContext objects.

```
Spark context Web UI available at http://192.168.0.172:22885
Spark context available as 'sc' (master = local[*], app id = local-1660719984220).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.5-hw-ei-310012
      /_/

Using Scala version 2.11.12 (OpenJDK 64-Bit Server VM, Java 1.8.0_272)
Type in expressions to have them evaluated.
Type :help for more information.
```

Figure 6-22

Step 1 Edit a data file.

Create the **cx_person.txt** file on the local Linux host. The file contains three columns: id, name, and age. The three columns are separated by space. The following figure shows the content of the **cx_person.txt** file.

```
[root@node-masterlnFFO ~]# vi  cx_person.txt
[root@node-masterlnFFO ~]# cat  cx_person.txt
1 zhangsan 20
2 lisi 29
3 wangwu 25
4 zhaoliu 30
5 tianqi 35
6 kobe 40
[root@node-masterlnFFO ~]#
```

Figure 6-23

Step 2 Upload the data file to a directory in HDFS.

```
hdfs dfs -put cx_person.txt   /user/stu01
hdfs dfs -ls /user/stu01
```

```
[root@node-masterlnFFO ~]# hdfs dfs -put cx_person.txt  /user/stu01
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/slf4j-log4j
12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/common/lib/slf
4j-log4j12-1.7.30.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]
[root@node-masterlnFFO ~]# hdfs dfs -ls /user/stu01
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/slf4j-log4j
12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/common/lib/slf
4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 4 items
-rw-r--r--   2 root hadoop         71 [redacted] 16:23 /user/stu01/cx_person.txt
-rw-r--r--   2 root hadoop         49 [redacted] 10:11 /user/stu01/cx_wd.txt
drwxr-xr-x   - root hadoop          0 [redacted] 15:50 /user/stu01/input
drwxr-xr-x   - root hadoop          0 [redacted] 10:15 /user/stu01/output01
```

Figure 6-24

Run the **spark-shell** command to go to Spark. Then run the following command to read data and separate data in each row using column separators:

```
val lineRDD= sc.textFile("/user/stu01/cx_person.txt").map(_.split(" "))
```

```
scala> val lineRDD= sc.textFile("/user/stu01/cx_person.txt").map(_.split(" "))
lineRDD: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[2] at map at
 <console>:24

scala>
```

Figure 6-25

Step 3 Define a case class.

A class is equivalent to a schema of a table.

```
case class Person(id:Int, name:String, age:Int)
```

```
scala> case class Person(id:Int, name:String, age:Int)
defined class Person

scala>
```

Figure 6-26

Step 4 Associate an RDD with the case class.

```
val personRDD = lineRDD.map(x => Person(x(0).toInt, x(1), x(2).toInt))
```

```
scala> val personRDD = lineRDD.map(x => Person(x(0).toInt, x(1), x(2).toInt))
personRDD: org.apache.spark.rdd.RDD[Person] = MapPartitionsRDD[3] at map at <console>:27

scala>
```

Figure 6-27

Step 5 Transform the RDD into DataFrame.

```
val personDF = personRDD.toDF
```

```
scala> val personDF = personRDD.toDF
personDF: org.apache.spark.sql.DataFrame = [id: int, name: string ... 1 more field]

scala>
```

Figure 6-28

Step 6 View information about DataFrame.

```
personDF.show
```

```
scala> personDF.show
[redacted] 16:44:42,726 | WARN  | main | The enable mv value "null" is invalid.
Using the default value "true" | org.apache.carbondata.core.util.CarbonPropertie
s.validateEnableMV(CarbonProperties.java:511)
[redacted] 16:44:42,790 | WARN  | main | The value "LOCALLOCK" configured for ke
y carbon.lock.type is invalid for current file system. Use the default value HDF
SLOCK instead. | org.apache.carbondata.core.util.CarbonProperties.validateAndCon
figureLockType(CarbonProperties.java:440)
+---+--------+---+
| id|    name|age|
+---+--------+---+
|  1|zhangsan| 20|
|  2|    lisi| 29|
|  3|  wangwu| 25|
|  4| zhaoliu| 30|
|  5|  tianqi| 35|
|  6|    kobe| 40|
+---+--------+---+
```

Figure 6-29

```
personDF.printSchema
```

```
scala> personDF.printSchema
root
 |-- id: integer (nullable = false)
 |-- name: string (nullable = true)
 |-- age: integer (nullable = false)


scala>
```

Figure 6-30

Step 7 Use the domain-specific language (DSL).

DataFrame provides the DSL to operate structured data.

View the data of the name field.

```
personDF.select(personDF.col("name")).show
```

```
scala> personDF.select(personDF.col("name")).show
+--------+
|    name|
+--------+
|zhangsan|
|    lisi|
|  wangwu|
| zhaoliu|
|  tianqi|
|    kobe|
+--------+


scala>
```

Figure 6-31

Check another format of the name field.

```
personDF.select("name").show
```

```
scala> personDF.select("name").show
+--------+
|    name|
+--------+
|zhangsan|
|    lisi|
|  wangwu|
| zhaoliu|
|  tianqi|
|    kobe|
+--------+


scala>
```

Figure 6-32

Step 8 Check the data of the **name** and **age** fields.

```
personDF.select(col("name"), col("age")).show
```

```
scala> personDF.select(col("name"), col("age")).show
+--------+---+
|    name|age|
+--------+---+
|zhangsan| 20|
|    lisi| 29|
|  wangwu| 25|
| zhaoliu| 30|
|  tianqi| 35|
|    kobe| 40|
+--------+---+


scala>
```

Figure 6-33

Step 9 Query all names and ages and increase the value of age by 1.

```
personDF.select(col("id"), col("name"), col("age") + 1).show
```

```
scala> personDF.select(col("id"), col("name"), col("age") + 1).show
+---+--------+---------+
| id|    name|(age + 1)|
+---+--------+---------+
|  1|zhangsan|       21|
|  2|    lisi|       30|
|  3|  wangwu|       26|
|  4| zhaoliu|       31|
|  5|  tianqi|       36|
|  6|    kobe|       41|
+---+--------+---------+
```

Figure 6-34

You can also run the following command:

```
personDF.select(personDF("id"), personDF("name"), personDF("age") + 1).show
```

```
scala> personDF.select(personDF("id"), personDF("name"), personDF("age") + 1).sh
ow
+---+--------+---------+
| id|    name|(age + 1)|
+---+--------+---------+
|  1|zhangsan|       21|
|  2|    lisi|       30|
|  3|  wangwu|       26|
|  4| zhaoliu|       31|
|  5|  tianqi|       36|
|  6|    kobe|       41|
+---+--------+---------+
```

Figure 6-35

Step 10 Use the **filter** method to filter the records where age is no less than 25.

```
personDF.filter(col("age") >= 25).show
```

```
scala> personDF.filter(col("age") >= 25).show
+---+-------+---+
| id|   name|age|
+---+-------+---+
|  2|   lisi| 29|
|  3| wangwu| 25|
|  4|zhaoliu| 30|
|  5| tianqi| 35|
|  6|   kobe| 40|
+---+-------+---+
```

Figure 6-36

Step 11 Count the number of people who are older than 30.

```
personDF.filter(col("age")>30).count()
```

```
scala> personDF.filter(col("age")>30).count()
res8: Long = 2

scala>
```

Figure 6-37

Step 12 Group people by age and collect statistics on the number of people of the same age.

```
personDF.groupBy("age").count().show
```

```
scala> personDF.groupBy("age").count().show
+---+-----+
|age|count|
+---+-----+
| 20|    1|
| 40|    1|
| 35|    1|
| 25|    1|
| 29|    1|
| 30|    1|
+---+-----+
```

Figure 6-38

Step 13 Use SQL.

A powerful feature of DataFrame is that it can be regarded as a relational data table. You can use **spark.sql()** in the program to execute SQL statements for query. The result is returned as a DataFrame.

If the SQL is used, you need to register DataFrame as a table in the following way:

```
personDF.registerTempTable("cx_t_person")
```

```
scala> personDF.registerTempTable("cx_t_person")
warning: there was one deprecation warning; re-run with -deprecation for details
```

Figure 6-39

Display the schema information of the table.

```
spark.sql("desc cx_t_person ").show
```

```
scala> spark.sql("desc cx_t_person ").show
+--------+---------+-------+
|col_name|data_type|comment|
+--------+---------+-------+
|      id|      int|   null|
|    name|   string|   null|
|     age|      int|   null|
+--------+---------+-------+
```

Figure 6-40

Step 14 Query the two oldest people.

```
spark.sql("select * from  cx_t_person  order by age desc limit 2").show
```

```
scala> spark.sql("select * from  cx_t_person  order by age desc limit 2").show
+---+------+---+
| id|  name|age|
+---+------+---+
|  6|  kobe| 40|
|  5|tianqi| 35|
+---+------+---+
```

Figure 6-41

Step 15 Query information about people older than 30.

```
spark.sql("select * from cx_t_person where age > 30 ").show
```

```
scala> spark.sql("select * from cx_t_person where age > 30 ").show
+---+------+---+
| id|  name|age|
+---+------+---+
|  5|tianqi| 35|
|  6| sunba| 40|
+---+------+---+
```

Figure 6-42

6.2.5 Task 5: Spark SQL DataSet Programming

Step 1 Create a dataset using **spark.createDataset**.

```
val ds1 = spark.createDataset(1 to 5)
ds1.show
```

```
scala> val ds1 = spark.createDataset(1 to 5)
ds1: org.apache.spark.sql.Dataset[Int] = [value: int]

scala> ds1.show
+-----+
|value|
+-----+
|    1|
|    2|
|    3|
|    4|
|    5|
+-----+
```

Figure 6-43

Step 2 Create a dataset using a file.

```
val ds2 = spark.createDataset(sc.textFile("/user/stu01/cx_person.txt"))
ds2.show
```

```
scala> val ds2 = spark.createDataset(sc.textFile("/user/stu01/cx_person.txt"))
ds2: org.apache.spark.sql.Dataset[String] = [value: string]

scala> ds2.show
+-------------+
|        value|
+-------------+
|1 zhangsan 20|
|    2 lisi 29|
|  3 wangwu 25|
| 4 zhaoliu 30|
|  5 tianqi 35|
|    6 kobe 40|
+-------------+
```

Figure 6-44

Step 3 Create a dataset using the **toDS** method.

```
case class Person2(id:Int, name:String, age:Int)
val data = List(Person2(1001,"liubei",20),Person2(1002,"guanyu",30))
val ds3 = data.toDS
ds3.show
```

```
scala> case class Person2(id:Int, name:String, age:Int)
defined class Person2

scala> val data = List(Person2(1001,"liubei",20),Person2(1002,"guanyu",30))
data: List[Person2] = List(Person2(1001,liubei,20), Person2(1002,guanyu,30))

scala> val ds3 = data.toDS
ds3: org.apache.spark.sql.Dataset[Person2] = [id: int, name: string ... 1 more field]

scala> ds3.show
+----+------+---+
|  id|  name|age|
+----+------+---+
|1001|liubei| 20|
|1002|guanyu| 30|
+----+------+---+
```

Figure 6-45

Step 4 Create a database by using DataFrame and **as[Type]**.

Perform transformation based on the DataFrame of **personDF** in task 1. Note that the **person** object fields in **Person2** and **personDF** must be the same.

```
val ds4= personDF.as[Person2]
ds4.show
```

```
scala> val ds4= personDF.as[Person2]
ds4: org.apache.spark.sql.Dataset[Person2] = [id: int, name: string ... 1 more field]

scala> ds4.show
+---+--------+---+
| id|    name|age|
+---+--------+---+
|  1|zhangsan| 20|
|  2|    lisi| 29|
|  3|  wangwu| 25|
|  4| zhaoliu| 30|
|  5|  tianqi| 35|
|  6|    kobe| 40|
+---+--------+---+
```

Figure 6-46

Step 5 Collect statistics on the number of people older than 30 in the dataset.

```
ds4.filter(col("age") >= 25).show
```

```
scala> ds4.filter(col("age") >= 25).show
+---+-------+---+
| id|   name|age|
+---+-------+---+
|  2|   lisi| 29|
|  3| wangwu| 25|
|  4|zhaoliu| 30|
|  5| tianqi| 35|
|  6|   kobe| 40|
+---+-------+---+
```

Figure 6-47

6.3 Exercise Summary

6.3.1 Quiz

Typically, Spark runs more efficiently than MapReduce. What are the built-in mechanisms of Spark for higher efficiency?

6.3.2 Summary

This exercise introduces RDD-based Spark Core programming and DataFrame- and DataSet-based Spark SQL programming, and enables trainees to understand the basic operations of Spark programming.

7 Flink Real-Time Processing System Practices

7.1 About This Exercise

7.1.1 Overview

Flink is a unified computing framework that supports both batch processing and stream processing. It provides a stream data processing engine that supports data distribution and parallel computing.

Flink provides high-concurrency pipeline data processing, millisecond-level latency, and high reliability, making it suitable for low-latency data processing.

7.1.2 Objectives

The asynchronous checkpointing mechanism and real-time hot-selling product statistics of Flink help you understand the core idea of Flink and how to use Flink to solve problems.

7.2 Tasks

7.2.1 Task 1: Importing a Flink Sample Project

Step 1 Download Flink sample code.

Visit https://support.huaweicloud.com/intl/en-us/devg3-mrs/mrs_07_010002.html.

Click the sample project of MRS 3.1.0 to download it.

Figure 7-1

Step 2 Import the sample project.

For details about how to import the MRS sample project, navigate to the Appendix to refer to the instructions on how to import an MRS sample project in Eclipse. After the import, the project automatically downloads related dependency packages.

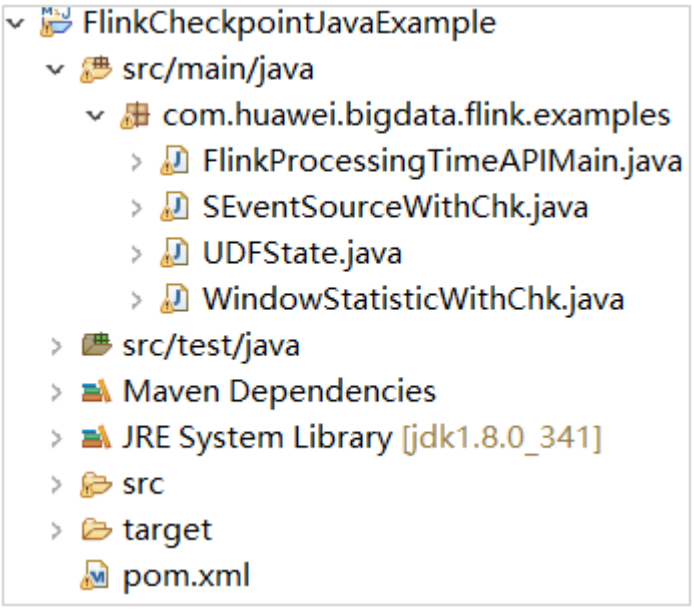


Figure 7-2

The preceding figure shows the project code structure.

7.2.2 Task 2: Exercising the Asynchronous Checkpointing Mechanism

Assume that you want to collect data volume in a 4-second time window every other second and the status of operators must be strictly consistent. That is, if an application recovers from a failure, the status of all operators must the same.

7.2.2.1 Data Planning

1. A custom operator generates about 10,000 pieces of data per second.
2. The generated data is a quadruple (Long, String, String, Integer).
3. After the statistics are collected, the statistical result is displayed on the terminal.
4. The output data is of the Long type.

7.2.2.2 Procedure

1. The source operator sends 10,000 pieces of data every second and injects the data into the window operator.
2. The window operator calculates the data generated in the last 4 seconds every second.
3. The statistical result is displayed on the terminal every second.
4. A checkpoint is triggered every 6 seconds and saved to HDFS.

7.2.2.3 Procedure

Step 1 Write the snapshot data code.

The snapshot data is used to store the number of data pieces recorded by operators during snapshot creation.

Create a class named **UDFState** in the **com.huawei.flink.example.common** package of the sample project. The code is as follows:

```
import java.io.Seriablizale;
// As a part of the snapshot, this class saves the user-defined status.
public class UDFState implements Serializable {
    private long count;
    // Initialize the user-defined status.
    public UDFState() {
        count = 0L;
    }
    // Set the user-defined status.
    public void setState(long count) {
        this.count = count;
    }
    // Obtain the user-defined status.
    public long geState() {
        return this.count;
    }
}
```

Step 2 Write a data source with a checkpoint.

The code snippet of a source operator pauses 1 second every time after sending 10,000 pieces of data. When a snapshot is created, the code saves the total number of sent data pieces in UDFState. When the snapshot is used for restoration, the number of sent data pieces saved in UDFState is read and assigned to the count variable.

Create the **SimpleSourceWithCheckPoint** class in the common package. The code is as follows:

```
import org.apache.flink.api.java.tuple.Tuple4;
import org.apache.flink.streaming.api.checkpoint.ListCheckpointed;
import org.apache.flink.streaming.api.functions.source.SourceFunction;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

public class SimpleSourceWithCheckPoint implements SourceFunction<Tuple4<Long, String, String,
Integer>>, ListCheckpointed<UDFState> {
    private long count = 0;
    private boolean isRunning = true;
    private String alphabet = "justtest";
    @Override
    public List<UDFState> snapshotState(long l, long l1) throws Exception
    {
```

```
            UDFState udfState = new UDFState();
            List<UDFState> udfStateList = new ArrayList<UDFState>();
            udfState.setState(count);
            udfStateList.add(udfState);
            return udfStateList;
}
@Override
public void restoreState(List<UDFState> list) throws Exception
{
      UDFState udfState = list.get(0);
      count = udfState.geState();
}
@Override
public void run(SourceContext<Tuple4<Long, String, String, Integer>> sourceContext) throws
Exception
{
       Random random = new Random();
       while (isRunning) {
            for (int i = 0; i < 10000; i++) {
                 sourceContext.collect(Tuple4.of(random.nextLong(), "hello" + count, alphabet, 1));
                       count ++;
                 }
                 Thread.sleep(1000);
            }
      }
      @Override
      public void cancel()
      {
            isRunning = false;
      }
}
```

Step 3 Define a window with a checkpoint.

This code snippet is about a window operator and is used to calculate the number or tuples in a window.

Create the **WindowStatisticWithChk** class in the common package. The code is as follows:

```
import org.apache.flink.api.java.tuple.Tuple;
import org.apache.flink.api.java.tuple.Tuple4;
import org.apache.flink.streaming.api.checkpoint.ListCheckpointed;
import org.apache.flink.streaming.api.functions.windowing.WindowFunction;
import org.apache.flink.streaming.api.windowing.windows.TimeWindow;
import org.apache.flink.util.Collector;

import java.util.ArrayList;
import java.util.List;

public class WindowStatisticWithChk implements WindowFunction<Tuple4<Long, String, String,
Integer>, Long, Tuple, TimeWindow>, ListCheckpointed<UDFState> {
      private long total = 0;
      @Override
      public List<UDFState> snapshotState(long l, long l1) throws Exception
      {
```

```
            UDFState udfState = new UDFState();
            List<UDFState> list = new ArrayList<UDFState>();
            udfState.setState(total);
            list.add(udfState);
            return list;
}
@Override
      public void restoreState(List<UDFState> list) throws Exception
      {
            UDFState udfState = list.get(0);
            total = udfState.geState();
      }
@Override
      public void apply(Tuple tuple, TimeWindow timeWindow, Iterable<Tuple4<Long, String, String,
Integer>> iterable, Collector<Long> collector) throws Exception
      {
            long count = 0L;
            for (Tuple4<Long, String, String, Integer> tuple4 : iterable) {
                  count ++;
            }
            total += count;
            collector.collect(total);
      }
}
```

Step 4 Develop application code.

The code is about the definition of StreamGraph and is used to implement services. The processing time is used as the time for triggering the window.

Create the **FlinkProcessingTimeAPIChkMain** class in the common package. The code is as follows:

```
import org.apache.flink.api.java.utils.ParameterTool;
import org.apache.flink.runtime.state.StateBackend;
import org.apache.flink.runtime.state.filesystem.FsStateBackend;
import org.apache.flink.streaming.api.CheckpointingMode;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.api.windowing.assigners.SlidingProcessingTimeWindows;
import org.apache.flink.streaming.api.windowing.time.Time;
public class FlinkProcessingTimeAPIMain {
      public static void main(String[] args) throws Exception
      {
            String chkPath = ParameterTool.fromArgs(args).get("chkPath",
"hdfs://hacluster/flink/checkpoints/");
            StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
            env.setStateBackend((StateBackend) new FsStateBackend((chkPath)));
            env.enableCheckpointing(6000, CheckpointingMode.EXACTLY_ONCE);
            env.addSource(new SimpleSourceWithCheckPoint())
                        .keyBy(0)
                        .window(SlidingProcessingTimeWindows.of(Time.seconds(4), Time.seconds(1)))
                        .apply(new WindowStatisticWithChk())
                        .print();
```

```
            env.execute();
        }
}
```

The code is written.

Step 5 Package the program.

Open the cmd window, go to the directory where the project is located, and run the **mvn package** command to package the project.

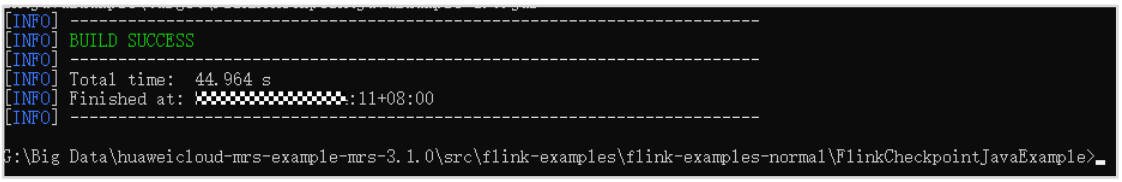


Figure 7-3

Run the **mvn package** command to generate a JAR file, for example, **FlinkCheckpointJavaExample-1.0.jar**, and obtain it from the **target** directory in the project directory.

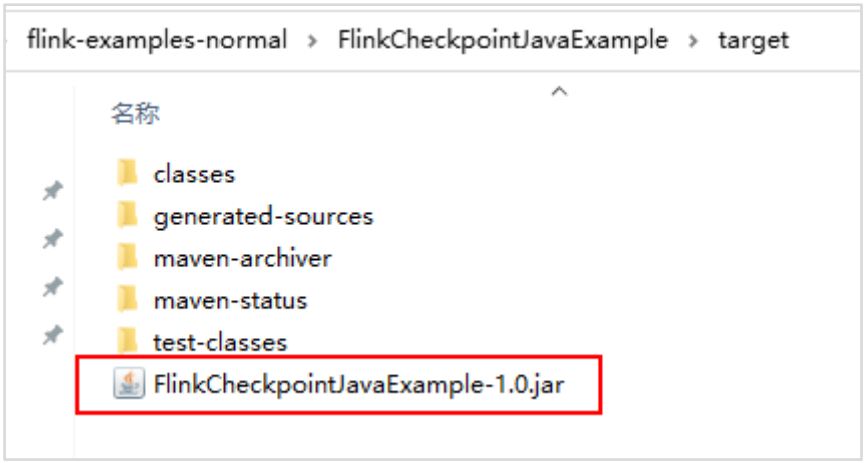


Figure 7-4

Step 6 Use WinSCP to log in to an ECS.

Upload the JAR file to the **/root** directory.

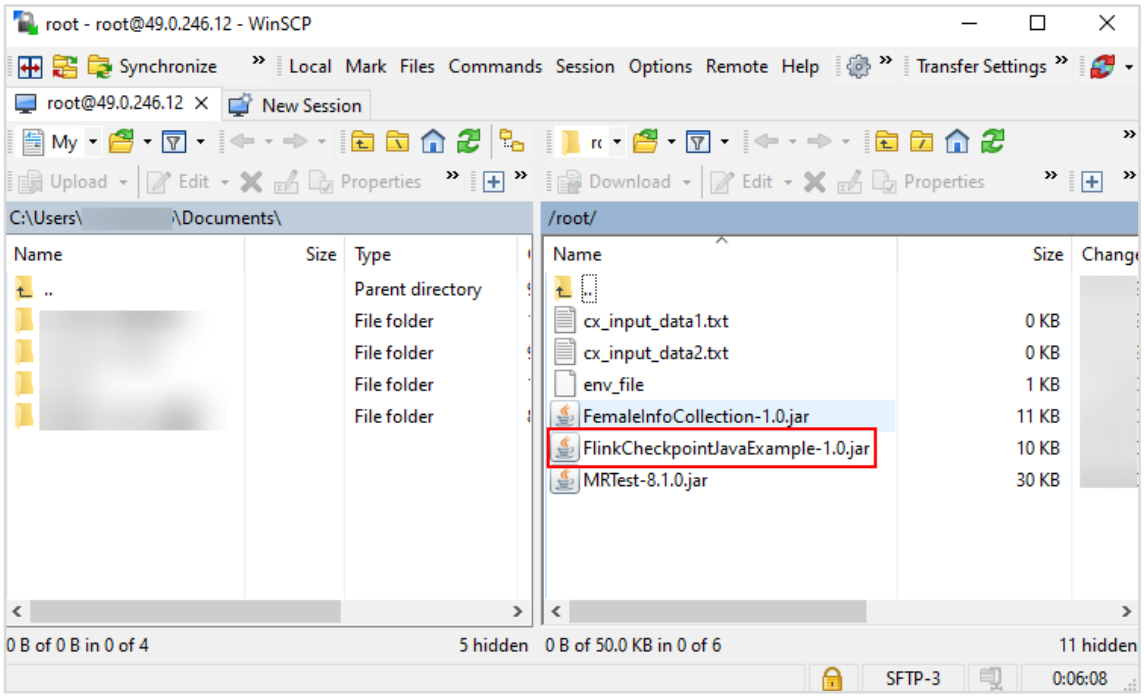


Figure 7-5

Step 7 Start the Flink cluster.

Use PuTTY to log in to the ECS, add the permission, and run the following commands to set the source environment variable:

```
su - omm
source /opt/Bigdata/client/bigdata_env
hdfs dfs -chmod -R 777 /flink
cd /opt/client
source /opt/client/bigdata_env
```

```
[root@node-masterlnxXi ~]# su - omm
Last login: [redacted] 17:06:14 CST [redacted]
[omm@node-masterlnxXi ~]$ source /opt/Bigdata/client/bigdata_env
[omm@node-masterlnxXi ~]$ hdfs dfs -chmod -R 777 /flink
[omm@node-masterlnxXi ~]$ su - root
Password:
Last login: [redacted] 17:05:47 CST [redacted] from 119.3.119.21 on pts/0
[root@node-masterlnxXi ~]# cd /opt/client
[root@node-masterlnxXi client]# source /opt/client/bigdata_env
[root@node-masterlnxXi client]#
```

Figure 7-6

Start the Flink cluster before running the Flink applications on Linux. Run the **yarn session** command on the Flink client to start the Flink cluster. The following is a command example:

```
/opt/client/Flink/flink/bin/yarn-session.sh  -n 3  -jm  1024  -tm  1024
```

```
[root@node-master1nxXi client]# /opt/client/Flink/flink/bin/yarn-session.sh  -n 3  -jm  1024  -tm  1024
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/Flink/flink/lib/log4j-slf4j-impl-2.12.1.jar!/org/slf4j/impl/
StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.30.jar
!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.
7.30.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.apache.logging.slf4j.Log4jLoggerFactory]
2[illegible] 17:14:32,562 INFO  org.apache.flink.configuration.GlobalConfiguration           [] - Loading co
nfiguration property: akka.ask.timeout, 10 s
2[illegible] 17:14:32,568 INFO  org.apache.flink.configuration.GlobalConfiguration           [] - Loading co
nfiguration property: akka.client-socket-worker-pool.pool-size-factor, 1.0
[illegible] 17:14:32,568 INFO  org.apache.flink.configuration.GlobalConfiguration           [] - Loading co
nfiguration property: akka.client-socket-worker-pool.pool-size-max, 2
[illegible] 17:14:32,568 INFO  org.apache.flink.configuration.GlobalConfiguration           [] - Loading co
nfiguration property: akka.client-socket-worker-pool.pool-size-min, 1
```

Figure 7-7

Note: **yarn-session** starts a running Flink cluster on Yarn. Once the session is successfully created, you can use the bin/flink tool to submit tasks to the cluster. The system uses the **conf/flink-conf.yaml** configuration file by default.

Parameters in the **yarn-session** command:

Mandatory:

-n,--container <arg>: number of allocated Yarn containers (= number of TaskManagers)

Optional:

-D <arg>: dynamic attribute.

-d,--detached: independently running.

-jm,--jobManagerMemory <arg>: JobManager memory, in MB.

-nm,--name: sets a name for a user-defined application on Yarn.

-q,--query: displays available resources (memory and number of CPU cores) on Yarn.

-qu,--queue <arg>: specifies the Yarn queue.

-s,--slots <arg>: number of slots used by each TaskManager.

-tm,--taskManagerMemory <arg>: memory of each TaskManager, in MB.

-z,--zookeeperNamespace <arg>: creates a namespace in ZooKeeper in HA mode.

After the command is executed, the following result is displayed:

```
[illegible] 17:14:45,041 INFO  org.apache.flink.shaded.curator4.org.apache.curator.fra
ived: {server.20=node-master1nxXi.mrs-ukrl.com:2888:3888:participant;node-master1nxXi
str-corekfge0002.mrs-ukrl.com:2888:3888:participant;node-str-corekfge0002.mrs-ukrl.co
888:participant;node-master2tefz.mrs-ukrl.com:2181}
[illegible] 17:14:45,046 INFO  org.apache.flink.shaded.curator4.org.apache.curator.fra
ived: {server.20=node-master1nxXi.mrs-ukrl.com:2888:3888:participant;node-master1nxXi
str-corekfge0002.mrs-ukrl.com:2888:3888:participant;node-str-corekfge0002.mrs-ukrl.co
888:participant;node-master2tefz.mrs-ukrl.com:2181}
[illegible]2 17:14:45,223 INFO  org.apache.flink.runtime.leaderretrieval.DefaultLeaderR
rvice with ZookeeperLeaderRetrievalDriver{retrievalPath='/leader/rest_server_lock'}.
Cluster started: Yarn cluster with application id application_1663836345431_0001
JobManager Web Interface: http://192.168.0.221:32261
```

Figure 7-8

The IP address of the JobManager web page is an intranet IP address. Log in to FusionInsight Manager, find the server, and bind an EIP to it. For details about how to bind an EIP, see the related operations in the MRS documents. Use the EIP to change the intranet IP address and access the server. For example, if the bound IP address is **122.112.193.167**, the access address is **http://122.112.193.167:32261**.

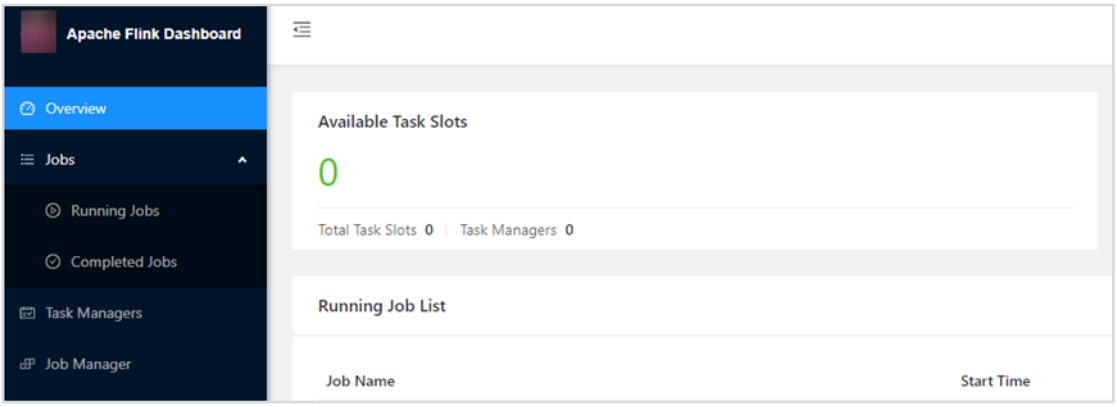


Figure 7-9

Step 8 Run the JAR file of Flink.

Save the checkpoint snapshot information to HDFS.

Press **Ctrl+C** to close the cluster command window (the cluster is still running in the background), or start PuTTY and run the following command:

```
/opt/client/Flink/flink/bin/flink run --class
com.huawei.bigdata.flink.examples.FlinkProcessingTimeAPIMain /root/FlinkCheckpointJavaExample-
1.0.jar --chkPath hdfs://hacluster/flink/checkpoint/
```

Parameter description: **class** is followed by the full path of the main program entry class and then the JAR package of the program. **chkPath** is the path for storing the checkpoint file. In cluster mode, Flink stores the checkpoint file in HDFS.

The **run** parameter can be used to edit and run a program.

Syntax: run [OPTIONS] <jar-file> <arguments>

run parameters:

-c,--class <classname>: If the entry class is not specified in the JAR package, this parameter is used to specify the entry class.

-m,--jobmanager <host:port>: specifies the address of the JobManager (active node) to be connected. This parameter can be used to specify a JobManager that is different from that in the configuration file.

-p,--parallelism <parallelism>: specifies the degree of parallelism of a program. The default value in the configuration file can be overwritten.

The following figure shows the command output.

Figure 7-10

On the Flink management panel, one more running Flink job is displayed.

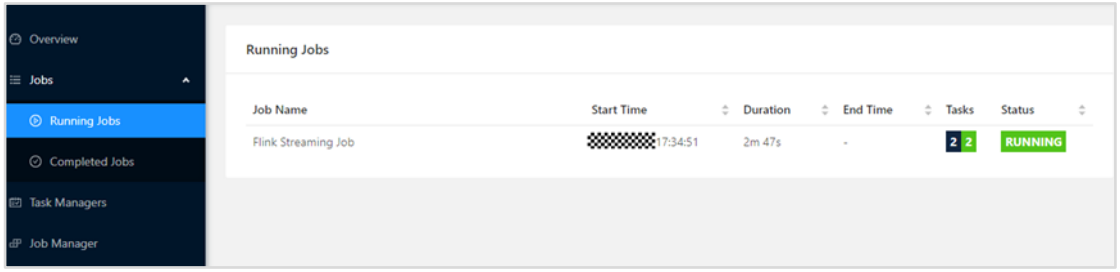


Figure 7-11

Step 9 View the output.

On the **Task Managers** page of the Flink management panel, click **Stdout** to view the output.

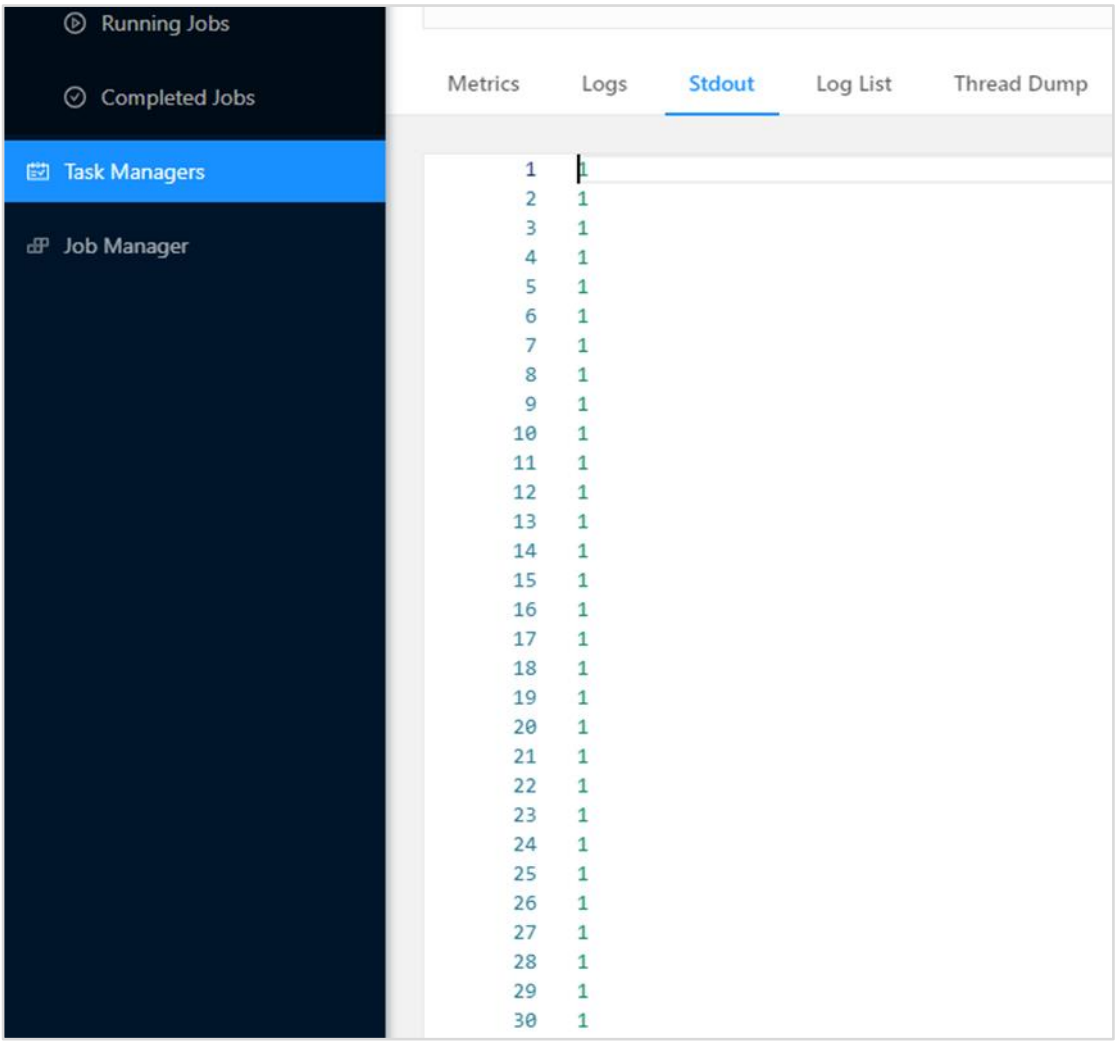


Figure 7-12

Step 10 View checkpoints.

Start PuTTY and run the HDFS command to view the **/flink/checkpoint** directory.

```
[root@node-master1nxXi ~]# hdfs dfs -ls /flink/checkpoint
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl
/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.30.jar!/org/sl
f4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 1 items
drwxr-xr-x   - root hadoop          0 2020-08-08 17:53 /flink/checkpoint/424a8761a46b3dc17b24efbfa6dfdf66
[root@node-master1nxXi ~]#
```

Figure 7-13

Step 11 Kill a Flink job.

Run the following command to view the Flink job list:

```
/opt/client/Flink/flink/bin/flink  list
```

```
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
[redacted]3,520 INFO  org.apache.flink.yarn.cli.FlinkYarnSessionCli                [] - Foun
d Yarn properties file under /tmp/.yarn-properties-root.
[redacted],520 INFO  org.apache.flink.yarn.cli.FlinkYarnSessionCli                [] - Foun
d Yarn properties file under /tmp/.yarn-properties-root.
[redacted]3,986 INFO  org.apache.flink.yarn.YarnClusterDescriptor                 [] - No p
ath for the flink jar passed. Using the location of class org.apache.flink.yarn.YarnClusterDescripto
r to locate the jar
[redacted],068 INFO  org.apache.flink.yarn.YarnClusterDescriptor                  [] - Foun
d Web Interface node-ana-corezzoh0002.mrs-ukrl.com:32261 of application 'application_1663836345431_0
008'.
Waiting for response...
------------------ Running/Restarting Jobs -------------------
[redacted] : 0acf4bcda244d0c6b57883d07ecaabcb : Flink Streaming Job (RUNNING)
--------------------------------------------------------------
No scheduled jobs.
[root@node-masterlnxXi ~]#
```

Figure 7-14

Specify the obtained job ID and run the following command:

```
/opt/client/Flink/flink/bin/flink  cancel  0acf4bcda244d0c6b57883d07ecaabcb
```

The following information is displayed.

```
[redacted]3,776 INFO  org.apache.flink.yarn.cli.FlinkYarnSessionCli                [] - Foun
d Yarn properties file under /tmp/.yarn-properties-root.
Cancelling job 0acf4bcda244d0c6b57883d07ecaabcb.
[redacted],242 INFO  org.apache.flink.yarn.YarnClusterDescriptor                  [] - No p
ath for the flink jar passed. Using the location of class org.apache.flink.yarn.YarnClusterDescripto
r to locate the jar
[redacted],325 INFO  org.apache.flink.yarn.YarnClusterDescriptor                  [] - Foun
d Web Interface node-ana-corezzoh0002.mrs-ukrl.com:32261 of application 'application_1663836345431_0
008'.
Cancelled job 0acf4bcda244d0c6b57883d07ecaabcb.
[root@node-masterlnxXi ~]#
```

Figure 7-15

7.3 Exercise Summary

7.3.1 Quiz

What is the time semantics in Flink?

7.3.2 Summary

This exercise describes two cases of implementing the asynchronous checkpointing mechanism and real-time hot-selling offerings, and helps trainees learn multiple core concepts and API usage of Flink, including how to use EventTime, Watermark, State, Window API, and TopN. It is expected that this exercise can deepen your understanding of Flink and help you resolve practical problems.

8 Flume Data Collection Practices

8.1 About This Exercise

8.1.1 Overview

Flume is an important data collection tool in the big data components, which is used to collect data from various data sources for other components to analyze data. In the log analysis service, server logs are collected to analyze whether the server is running properly. Flume is an important application in big data services. In real-time services, data is usually collected to Kafka for analysis and processing by real-time components such as Streaming and Spark.

8.1.2 Objectives

Upon the completion of this practice, you will be able to use Flume to migrate data in service scenarios.

8.2 Tasks

8.2.1 Task 1: Installing the Flume Client

Step 1 Download a client.

Log in to FusionInsight Manager, access the **mrs_hcia** cluster, and choose **Flume**. The Flume service page is displayed.

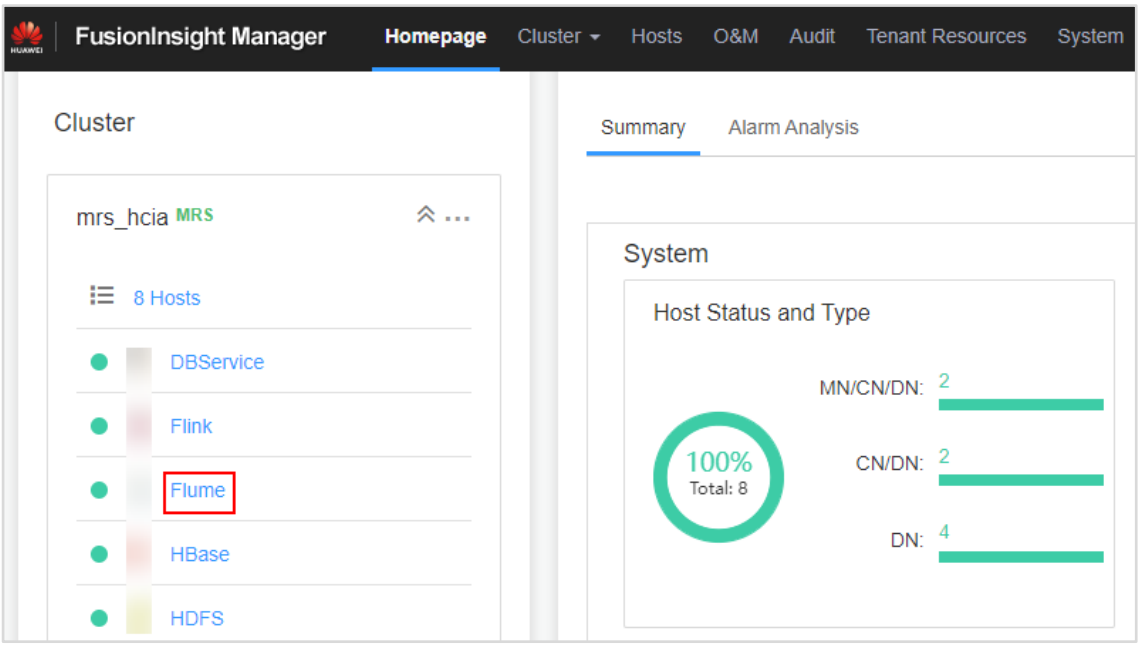


Figure 8-1

Go to the Flume page, click **More**, and select **Download Client**.

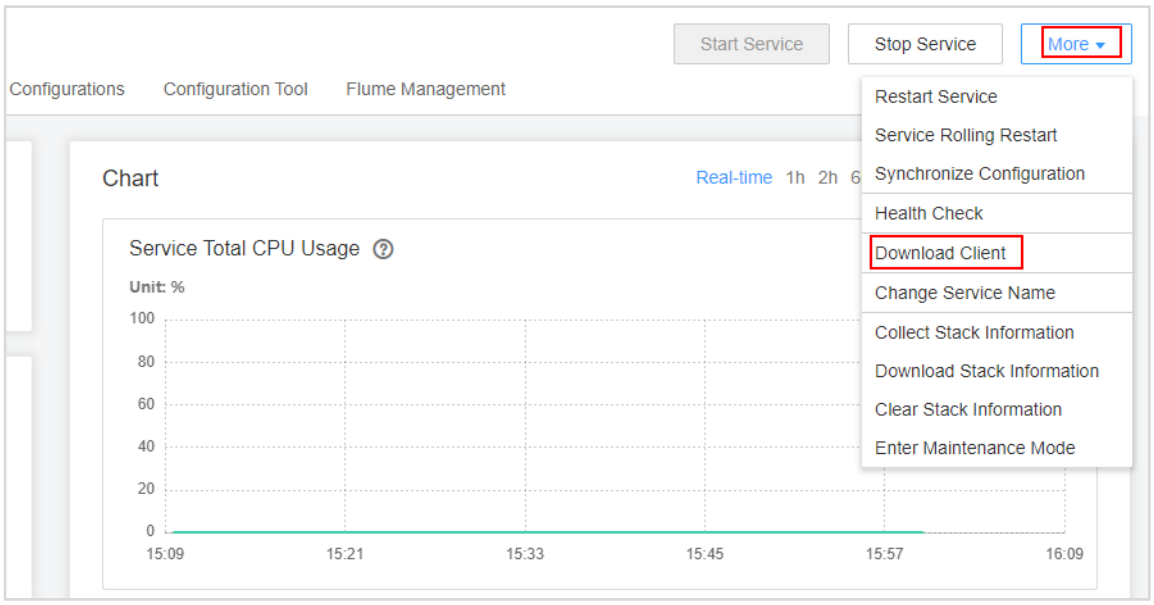


Figure 8-2

In the displayed dialog box, set **Select Client Type** to **Complete Client**. Set **Select Platform Type** to **x86_64**, select **Save to Path**, and click **OK** to generate the client file.

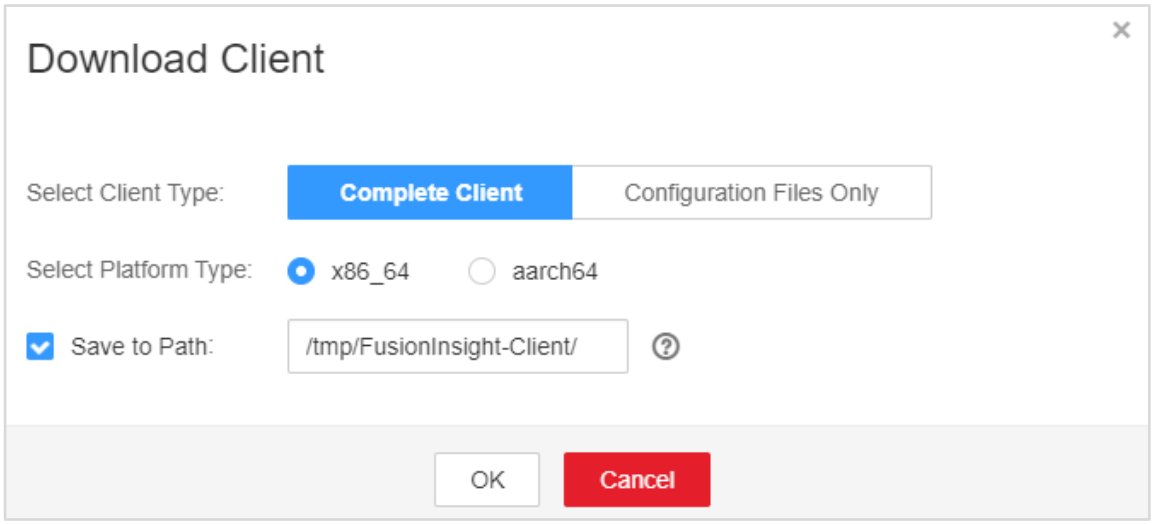


Figure 8-3

After the download is complete, a dialog box is displayed, indicating the server to which the file is downloaded and displaying the save path.

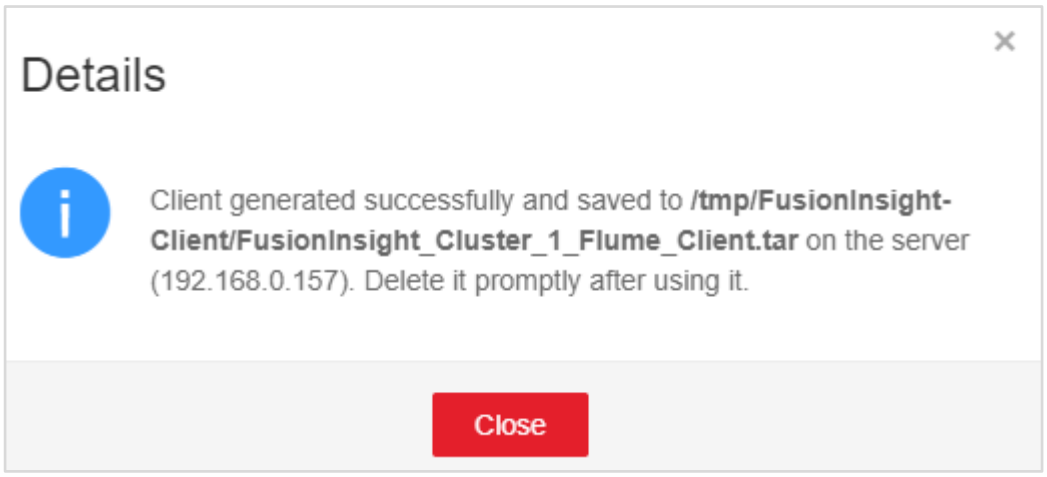


Figure 8-4

Step 2 Upload software packages.

Upload the software package to the **/opt/client** directory on the node where the Flume service client is to be installed.

Step 3 Decompress the software packages.

Log in to the server, go to the **/opt/client** directory, and run the following commands to decompress the installation package to the current directory:

```
cd /opt/client
tar -xvf FusionInsight_Cluster_1_Flume_Client.tar
```

```
[root@node-masterlknep client]# cd /opt/client
[root@node-masterlknep client]# tar -xvf FusionInsight_Cluster_1_Flume_Client.tar
FusionInsight_Cluster_1_Flume_ClientConfig.tar.sha256
FusionInsight_Cluster_1_Flume_ClientConfig.tar
```

Figure 8-5

Step 4 Verify the software package.

```
sha256sum -c FusionInsight_Cluster_1_Flume_ClientConfig.tar.sha256
```

If "FusionInsight_Cluster_1_Flume_ClientConfig.tar: OK" is displayed, the file package verification is successful.

```
[root@node-master1knep client]# sha256sum -c FusionInsight_Cluster_1_Flume_ClientConfig.tar.sha256
FusionInsight_Cluster_1_Flume_ClientConfig.tar: OK
```

Figure 8-6

Step 5 Decompress the **FusionInsight_Cluster_1_Flume_ClientConfig.tar** package.

```
tar -xvf FusionInsight_Cluster_1_Flume_ClientConfig.tar
```

```
[root@node-master1knep client]# tar -xvf FusionInsight_Cluster_1_Flume_ClientConfig.tar
FusionInsight_Cluster_1_Flume_ClientConfig
FusionInsight_Cluster_1_Flume_ClientConfig/refreshConfig.sh
FusionInsight_Cluster_1_Flume_ClientConfig/uninstall.sh
FusionInsight_Cluster_1_Flume_ClientConfig/ca.crt
FusionInsight_Cluster_1_Flume_ClientConfig/application.properties
FusionInsight_Cluster_1_Flume_ClientConfig/register-client.sh
FusionInsight_Cluster_1_Flume_ClientConfig/log4j.properties
FusionInsight_Cluster_1_Flume_ClientConfig/client-registry-1.0.0.jar
FusionInsight_Cluster_1_Flume_ClientConfig/autoRefreshConfig.sh
FusionInsight_Cluster_1_Flume_ClientConfig/clientregister.keytab
```

Figure 8-7

Step 6 Install the Flume client.

Run the following command in the Flume client installation directory to install the client to a specified directory (for example, **/opt/Flume**): After the client is installed successfully, the installation is complete.

```
./install.sh  -d /opt/Flume
```

If the following information is displayed, the client running environment is successfully installed.

```
[root@node-master1knep client]# cd /opt/client/FusionInsight_Cluster_1_Flume_ClientConfig/Flume/FlumeClient
[root@node-master1knep FlumeClient]# ./install.sh -d /opt/Flume
CST [illegible] 10:42:23 [flume-client install]: install flume client successfully.
[root@node-master1knep FlumeClient]#
```

Figure 8-8

8.2.2 Task 2: Using SpoolDir to Collect and Upload Data to HDFS

The SpoolDir data source uses Flume to monitor folders in a specified path and upload files to HDFS. Check that the HDFS and HBase clients are installed. Flume is mainly used for data ingestion. Therefore, you need to configure parameters based on service requirements.

Step 1 Access the **Configuration Tool** page of the Flume service.

On the **Configuration Tool** tab page, set **Agent Name** to **client**.

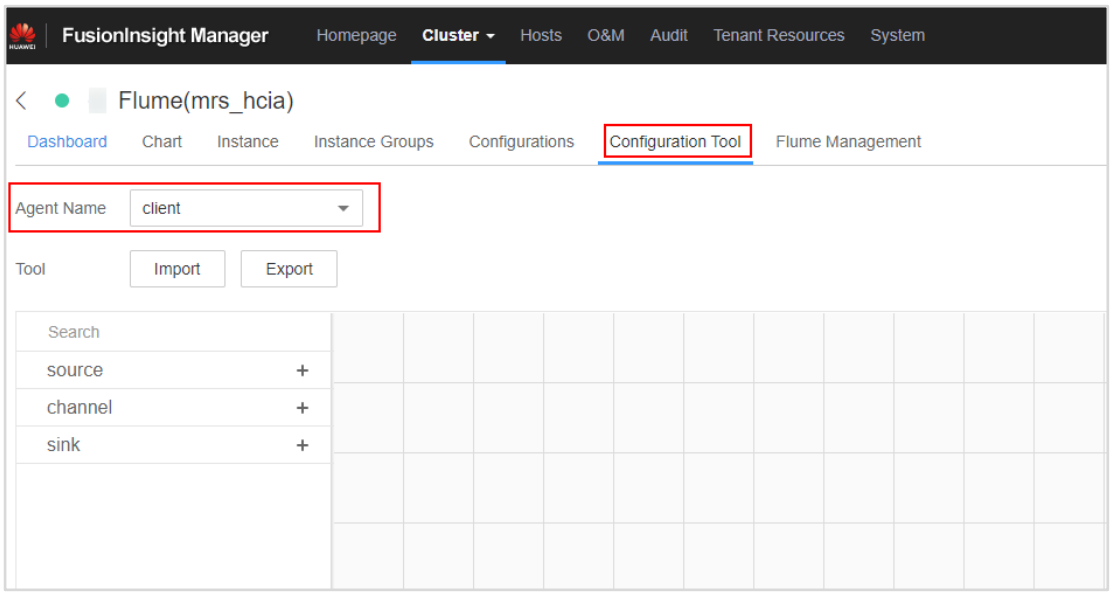


Figure 8-9

Step 2 Configure a source.

Click **+** to add a source, select **Spooldir Source**, and set the parameters as follows:

SourceName: **s1**. The value cannot be empty.

spoolDir: **/tmp/ spooldir**

For other configurations, see the following figure.

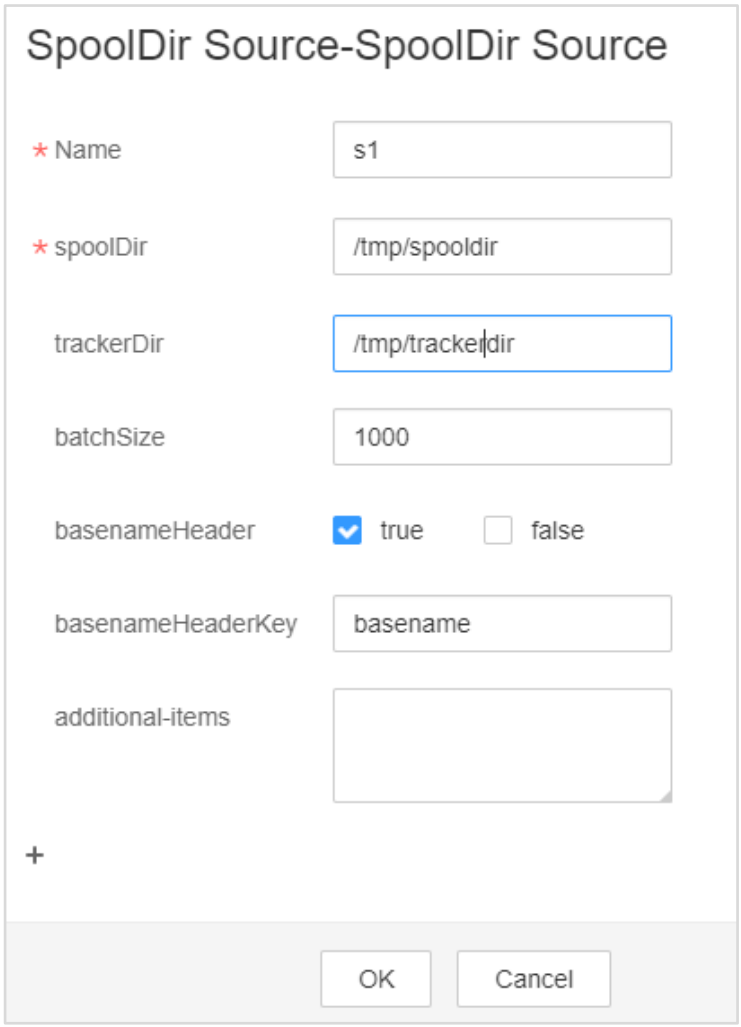


Figure 8-10

Step 3 Configure a channel.

Click **+** to add a channel, select **Memory Channel**, set **ChannelName** to **ch1**, and set other parameters by referring to the following figure.

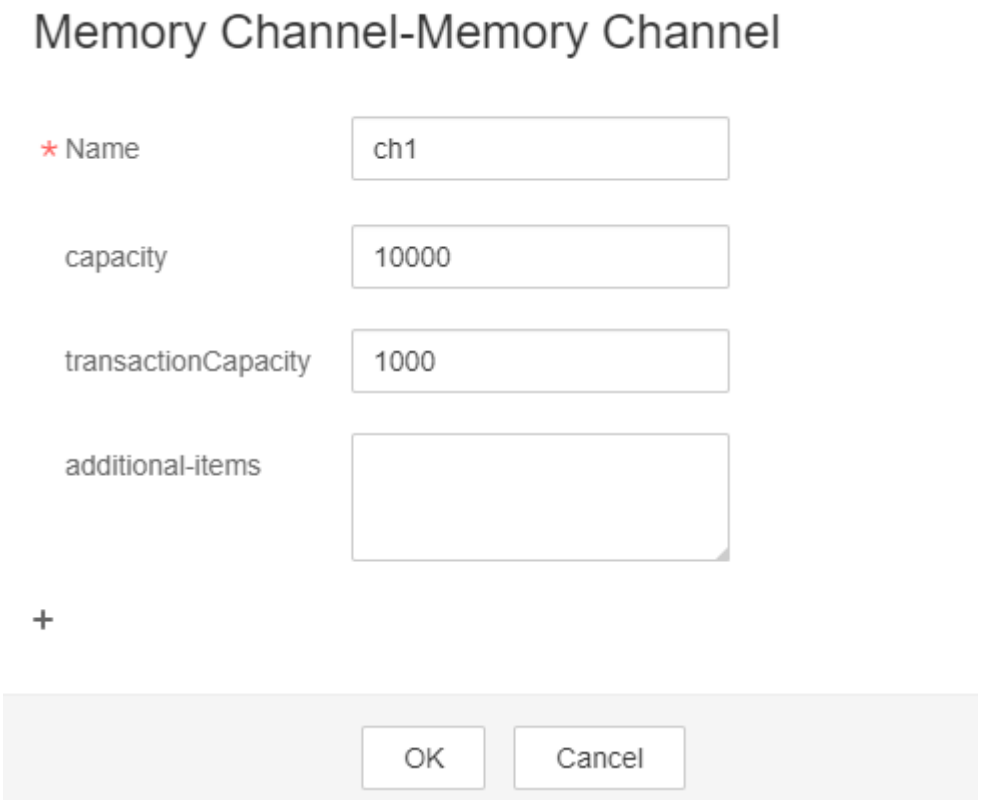


Figure 8-11

Step 4 Configure a sink.

Click **+** to add a sink., select **HDFS Sink**, and set the parameters as follows:

Name: **sh1**

hdfs.path: **hdfs://hacluster/user/cx_stu02/**. The **cx_stu02** directory is automatically created by the system.

Retain the default values for other parameters. See the following figure.

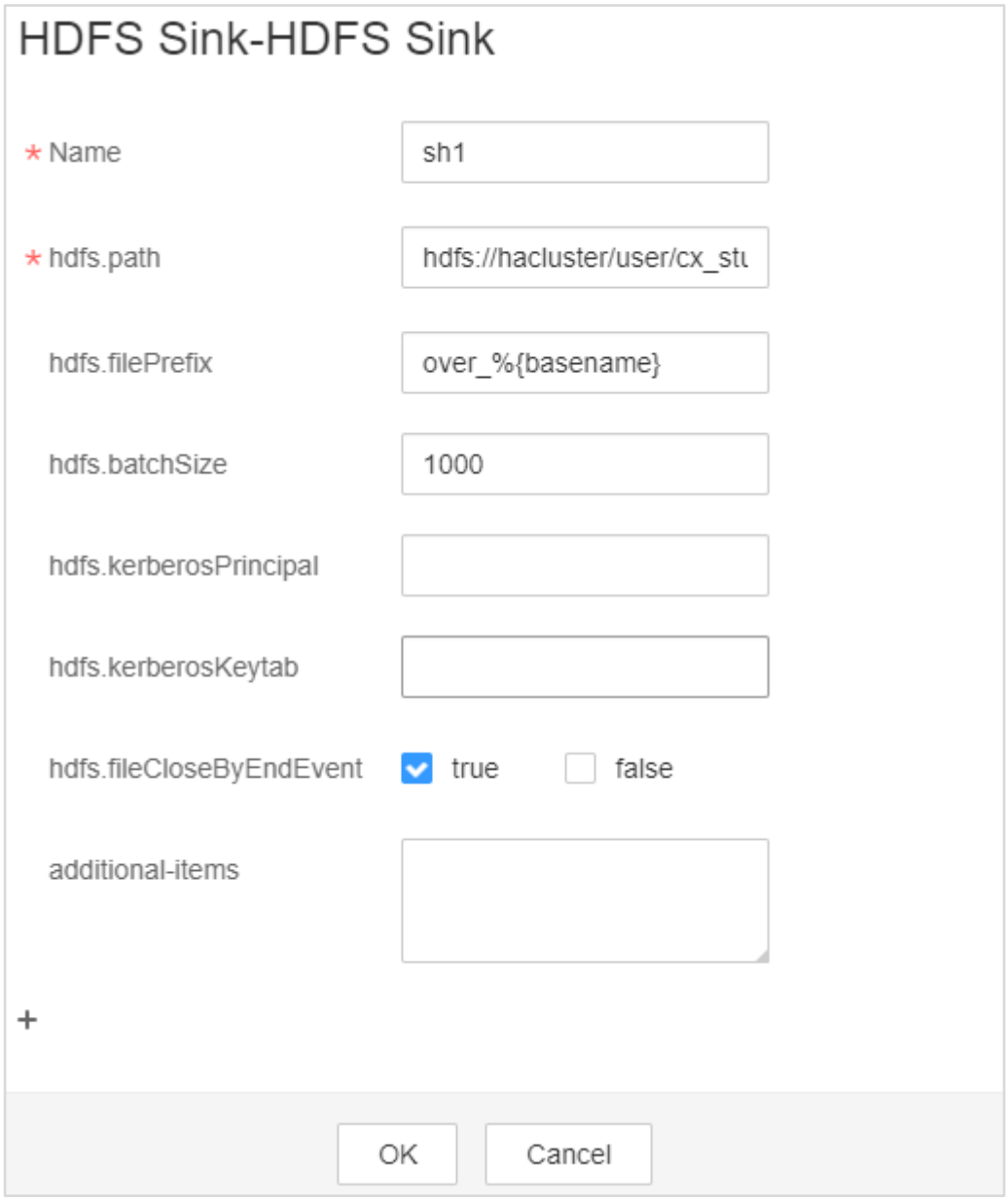


Figure 8-12

Note: The MRS cluster is in non-security mode. Therefore, you do not need to configure Kerberos in the sink.

Step 5 Generate a configuration file.

Click **Export**. The **properties.properties** configuration file is automatically generated.

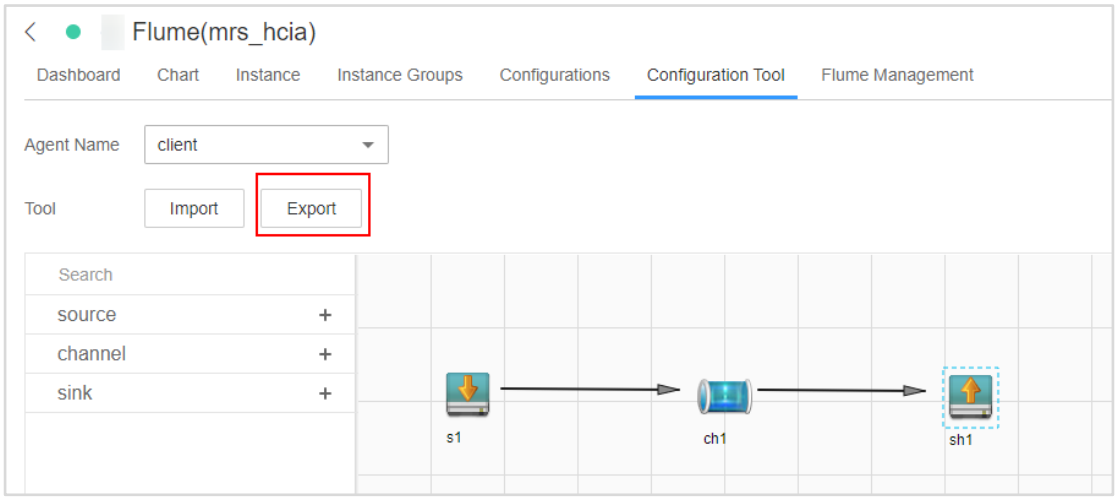


Figure 8-13

Step 6 Create **/tmp/spooldir** in Linux.

Step 7 Upload the Flume configuration file.

Use WinSCP to upload the **properties.properties** file to the **conf** directory of Flume, that is, **/opt/Flume/fusioninsight-flume-1.9.0/conf/**.

Note: The Flume client automatically loads the **properties.properties** file.

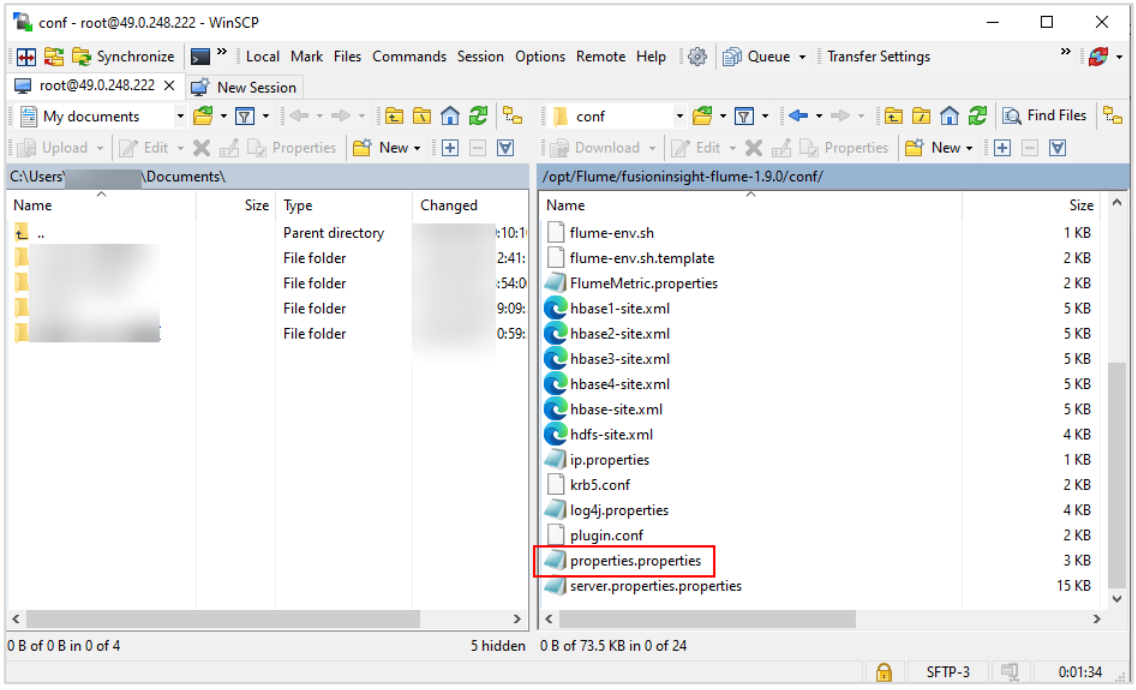


Figure 8-14

Step 8 Write a file for testing.

Go to the **/tmp/flume_spooldir** directory, run the **vi** command to create the **test1.txt** file, and enter any content.

```
[root@node-master1knep ~]# cd /tmp/spooldir
[root@node-master1knep spooldir]# vi test1.txt
```

Figure 8-15

Step 9 View the result.

Run the following commands to view the data collection result:

```
hdfs dfs -ls /user/cx_stu02
hdfs dfs -cat /user/cx_stu02/over_test1.txt
```

```
[root@node-master1knep spooldir]# hdfs dfs -ls /user/cx_stu02
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/slf4
ar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/common/
1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Found 1 items
-rw-r--r--   3 root hadoop         37 [redacted] 11:39 /user/cx_stu02/over_test1.txt
[root@node-master1knep spooldir]# hdfs dfs -cat /user/cx_stu02/over_test1.txt
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/slf4
ar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/common/
1.7.30.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]
hello world
hello hadoop
hello Flume
[root@node-master1knep spooldir]#
```

Figure 8-16

The data is successfully collected and uploaded to HDFS. You can continue to create a data file for testing.

8.2.3 Task 3: Using SpoolDir to Collect and Upload Data to Kafka

Flume uses SpoolDir to monitor folders in a specified path and then collects and uploads the data in the folders to Kafka. The consumers can read the data on the console.

Step 1 Access the **Configuration Tool** page of the Flume service.

On the Flume service page, click **Import** to import the configuration file in Task 2.

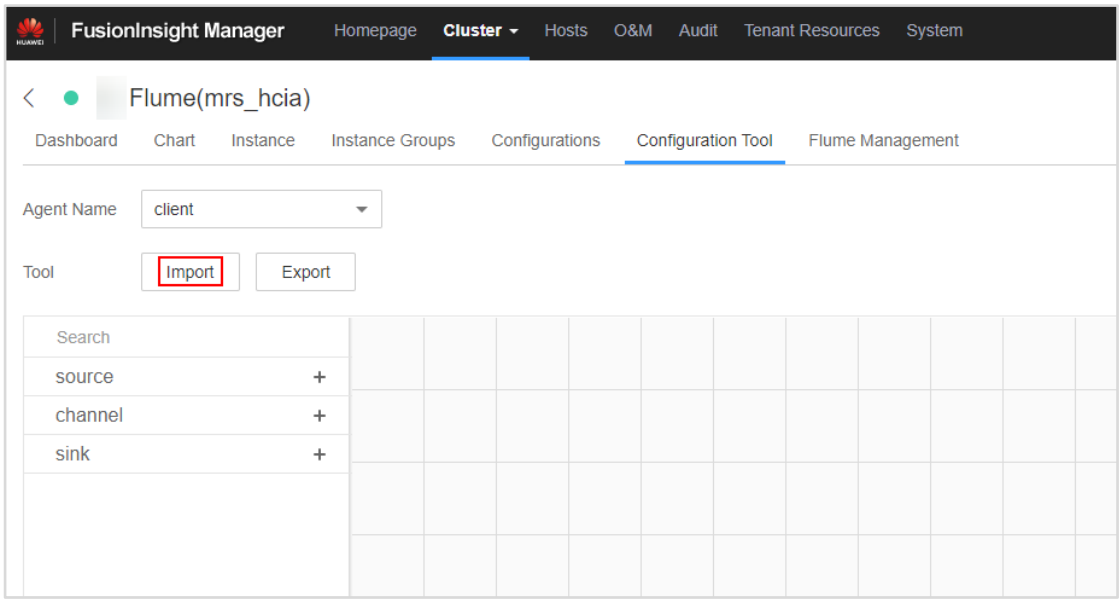


Figure 8-17

Step 2 Modify Sink configurations.

On the **Configuration Tool** page, select and delete the original **sink sh1**, select and add Kafka Sink again, and set **kafka.bootstrap.servers**, as shown in the following figure.

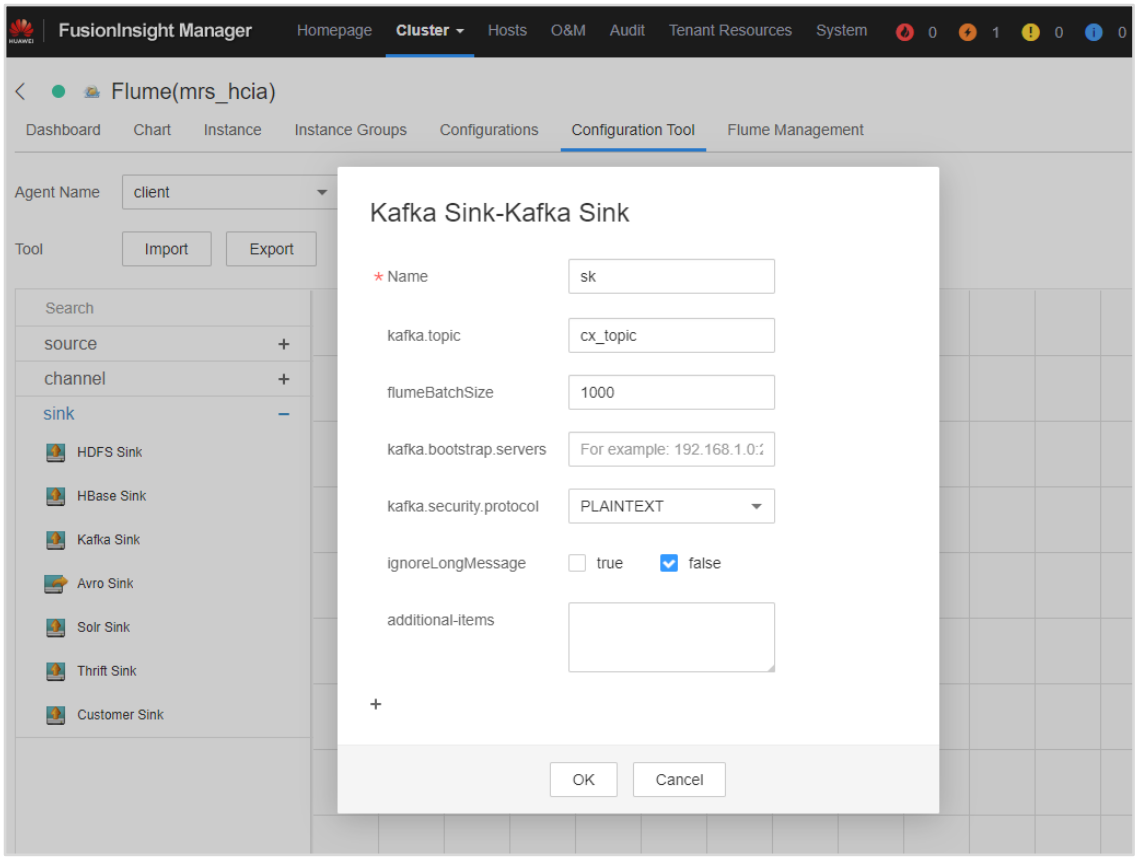


Figure 8-18

Note: If there are multiple Kafka instances in the cluster, you need to configure all of them. The port number is 9092 for all of them.

Step 3 Generate a configuration file.

After the configuration is complete, click **Export** to export the automatically generated **properties.properties** configuration file.

Step 4 Create a Kafka topic.

Go to the Kafka directory (**cd /opt/client/Kafka/kafka**) and run the following commands:

```
source /opt/client/bigdata_env
cd /opt/client/Kafka/kafka/bin
sh kafka-topics.sh --create --zookeeper 192.168.0.198:2181/kafka --partitions 1 --replication-factor 1 --topic cx_topic1
```

```
[root@node-masterlknep bin]# sh kafka-topics.sh --create --zookeeper 192.168.0
.198:2181/kafka --partitions 1 --replication-factor 1 --topic cx_topic1
WARNING: Due to limitations in metric names, topics with a period ('.') or under
score ('_') could collide. To avoid issues it is best to use either, but not bot
h.
Created topic cx_topic1.
[root@node-masterlknep bin]#
```

Figure 8-19

Note: You can obtain the IP address of the ZooKeeper by referring to the related content in the Appendix.

Step 5 Upload the Flume configuration file.

Use WinSCP to upload the **properties.properties** file to the **/opt/FlumeClient/fusioninsight-flume-1.9.0/conf/** directory.

Note: The Flume client automatically loads the **properties.properties** file.

Step 6 Create a console consumer.

Run the following command in the Kafka directory:

```
sh kafka-console-consumer.sh --topic cx_topic1 --bootstrap-server 192.168.0.105:9092 --consumer.config /opt/client/Kafka/kafka/config/consumer.properties
```

```
[root@node-masterlknep bin]# sh kafka-console-consumer.sh --topic cx_topic1 --bo
otstrap-server 192.168.0.105:9092 --consumer.config /opt/client/Kafka/kafka/conf
ig/consumer.properties
```

Figure 8-20

Note: The IP address of **bootstrap-server** corresponds to the IP address of the Kafka instance. You can obtain the IP address by referring to the related content in Appendix.

After this command is executed, the **cx_topic1** data is consumed. Do not perform other operations in this window or close it.

Step 7 Test data.

In PuTTY, open a shell connection (do not close the consumer window that is just started) and go to the **/tmp/spooldir** directory. Run the **vi** command to edit the **testkafka.txt** file, enter any content, save the file, and exit.

```
[root@node-master1knep ~]# cd /tmp/spooldir
[root@node-master1knep spooldir]# vi testkafka.txt
[root@node-master1knep spooldir]# ls
test1.txt.COMPLETED  testkafka.txt.COMPLETED
[root@node-master1knep spooldir]# cat testkafka.txt.COMPLETED
hello Flume and Kafka
[root@node-master1knep spooldir]#
```

Figure 8-21

Step 8 View the result.

Switch back to the shell window of the consumer. The data output is displayed.

```
[root@node-master1knep bin]# sh kafka-console-consumer.sh --topic cx_topic1 --bo
otstrap-server 192.168.0.105:9092 --consumer.config /opt/client/Kafka/kafka/conf
ig/consumer.properties
hello Flume and Kafka
```

Figure 8-22

8.3 Exercise Summary

8.3.1 Quiz

Is it possible that data collected by Flume will be lost?

8.3.2 Summary

This exercise describes how to use Flume to collect and upload data to HDFS and Kafka. This exercise aims to help trainees better understand Flume by learning common offline and real-time data collection methods.

9 Kafka Message Subscription Practices

9.1 About This Exercise

9.1.1 Overview

The Kafka message subscription system plays an important role in big data services, especially in real-time services. The typical Taobao You May Like service uses Kafka to store page clickstream data. After the streaming analysis, the analysis result is pushed to users.

9.1.2 Objectives

Understand how to use Kafka shell producers and consumers to generate and consume data in real time.

9.2 Tasks

9.2.1 Task 1: Producing and Consuming Kafka Messages on the Shell Side

Step 1 Log in to Kafka.

Use PuTTY to log in to a server and run the **source** command to set environment variables.

```
source /opt/client/bigdata_env
```

```
[root@node-masterlnFFO ~]# source /opt/client/bigdata_env
[root@node-masterlnFFO ~]#
```

Figure 9-1

Go to the **Kafka** directory.

```
cd /opt/client/Kafka/kafka/bin
```

```
[root@node-masterlnFFO kafka]# cd  /opt/client/Kafka/kafka/bin
[root@node-masterlnFFO bin]# ls
connect-distributed.sh          kafka-delete-records.sh
connect-standalone.sh           kafka-log-dirs.sh
kafka-acls.sh                   kafka-mirror-maker.sh
kafka-balancer.sh               kafka-preferred-replica-election.sh
kafka-broker-api-versions.sh    kafka-producer-perf-test.sh
kafka-broker-info.sh            kafka-reassign-partitions.sh
kafka-configs.sh                kafka-replica-verification.sh
kafka-console-consumer.sh       kafka-run-class.sh
kafka-console-producer.sh       kafka-streams-application-reset.sh
kafka-consumer-groups.sh        kafka-topics.sh
kafka-consumer-perf-test.sh     kafka-verifiable-consumer.sh
kafka-delegation-tokens.sh      kafka-verifiable-producer.sh
[root@node-masterlnFFO bin]#
```

Figure 9-2

Step 2 Create a Kafka topic.

Run the following command to create a Kafka topic:

```
sh kafka-topics.sh --create --topic cx_topic2 --partitions 1 --replication-factor 1 --zookeeper 192.168.0.172:2181/kafka
```

```
[root@node-masterlnFFO bin]# sh kafka-topics.sh --create --topic cx_topic2 --partitions 1  --replication-factor 1 --zookeeper 192.168.0.172:2181/kafka
WARNING: Due to limitations in metric names, topics with a period ('.') or underscore ('_') could collide. To avoid issues it is best to use either, but not both.
Created topic cx_topic2.
[root@node-masterlnFFO bin]#
```

Figure 9-3

Note: For details about how to obtain the ZooKeeper IP address, see the related content in the Appendix.

Step 3 View topics.

```
sh kafka-topics.sh  --list --zookeeper 192.168.0.172:2181/kafka
```

```
[root@node-masterlnFFO bin]# sh kafka-topics.sh  --list --zookeeper 192.168.0.172:2181/kafka
__KafkaMetricReport
__consumer_offsets
__default_metrics
cx_topic2
[root@node-masterlnFFO bin]#
```

Figure 9-4

Step 4 Create a console consumer.

```
sh kafka-console-consumer.sh --topic cx_topic2 --bootstrap-server 192.168.0.119:9092 --consumer.config /opt/client/Kafka/kafka/config/consumer.properties
```

```
[root@node-masterlnFFO bin]# sh kafka-console-consumer.sh --topic cx_topic2 --bootstrap-server 192.168.0.119:9092 --consumer.config /opt/client/Kafka/kafka/config/consumer.properties
```

Figure 9-5

Note: The IP address of **bootstrap-server** is the IP address of the Kafka broker. You can obtain the IP address by referring to the related content in Appendix.

After this command is executed, the **cx_topic2** data is consumed. Do not perform other operations in this window or close the window.

Step 5 Create a console producer.

Log in to PuTTY again, run the **source** command to obtain the environment variables, and go to the Kafka directory.

```
[root@node-masterlnFFO ~]# source /opt/client/bigdata_env
[root@node-masterlnFFO ~]# cd  /opt/client/Kafka/kafka/bin
[root@node-masterlnFFO bin]#
```

Figure 9-6

Run the following command to create a producer:

```
sh kafka-console-producer.sh --broker-list 192.168.0.119:9092 --topic cx_topic2 --producer.config /opt/client/Kafka/kafka/config/producer.properties
```

After the command is executed, enter any data.

```
[root@node-masterlnFFO bin]# sh kafka-console-producer.sh --broker-list 192.168.
0.119:9092 --topic cx_topic2 --producer.config /opt/client/Kafka/kafka/config/pr
oducer.properties
[[redacted] 11:48:09,881] WARN The configuration 'producer.type' was supplied bu
t isn't a known config. (org.apache.kafka.clients.producer.ProducerConfig)
[[redacted] 11:48:09,881] WARN The configuration 'serializer.class' was supplied
 but isn't a known config. (org.apache.kafka.clients.producer.ProducerConfig)
>hello
>test
>hello
>kafka
>hello
>hadoop
>
```

Figure 9-7

Note: The IP address of **broker-list** is the IP address of the Kafka broker. You can obtain the IP address by referring to the related content in Appendix.

Step 6 Test the producer and consumer.

Switch to the shell of the consumer. The console data output is displayed.

```
[root@node-masterlnFFO bin]# sh kafka-console-consumer.sh --topic cx_topic2 --bootstrap-
server 192.168.0.119:9092 --consumer.config /opt/client/Kafka/kafka/config/consumer.prop
erties

hello
test
hello
kafka
hello
hadoop
```

Figure 9-8

You can continue to enter data on the producer for testing.

9.2.2 Task 2: Using Kafka Consumer Groups

The consumer group is a very interesting design of Kafka. In terms of high concurrency, multiple servers can be placed in the same consumer group to ensure that all consumers do not pull the same message and the message is complete, thereby improving execution efficiency of the consumers.

Step 1 Create a topic.

Run the following commands to create a topic named **cx_topic3**. For details, see Task 1.

```
sh kafka-topics.sh --create --topic cx_topic3 --partitions 3 --replication-factor 1 --zookeeper 192.168.0.172:2181/kafka
sh kafka-topics.sh  --list --zookeeper 192.168.0.172:2181/kafka
```

```
[root@node-masterlnFFO bin]# sh kafka-topics.sh --create --topic cx_topic3 --partitions
 3 --replication-factor 1 --zookeeper 192.168.0.172:2181/kafka
WARNING: Due to limitations in metric names, topics with a period ('.') or underscore ('
_') could collide. To avoid issues it is best to use either, but not both.
Created topic cx_topic3.
[root@node-masterlnFFO bin]# sh kafka-topics.sh  --list --zookeeper 192.168.0.172:2181/k
afka
__KafkaMetricReport
__consumer_offsets
__default_metrics
cx_topic2
cx_topic3
[root@node-masterlnFFO bin]#
```

Figure 9-9

Note that the topic partition is set to **3**. Different value settings will lead to different effects.

Run the following command to delete a topic:

```
sh kafka-topics.sh --delete --topic Topic name --IP address of the node where the ZooKeeper instance resides:clientPort/kafka
```

Step 2 Create a producer and consumer.

Start the producer:

```
sh kafka-console-producer.sh --broker-list 192.168.0.119:9092 --topic cx_topic3 --producer.config /opt/client/Kafka/kafka/config/producer.properties
```

Open three PuTTY windows, set environment variables, go to the Kafka directory, and run the following command to start three consumers:

```
sh kafka-console-consumer.sh --topic cx_topic3 --bootstrap-server 192.168.0.119:9092  --consumer-property group1 --consumer.config /opt/client/Kafka/kafka/config/consumer.properties
```

Note that you can add **--consumer-property group1** to specify consumer group **group1**.

```
sh kafka-console-producer.sh --broker-list 192.168.0.70:9092 --topic cx_topic1 --producer.config
/opt/client/Kafka/kafka/config/producer.properties
```

Step 3 Configure three consumers.

Send the following six messages in sequence in the producer window.

```
[root@node-masterlnFFO bin]# sh kafka-console-producer.sh --broker-list 192.168
0.119:9092 --topic cx_topic3 --producer.config /opt/client/Kafka/kafka/config/p
oducer.properties
[[redacted] 13:50:44,093] WARN The configuration 'producer.type' was supplied b
t isn't a known config. (org.apache.kafka.clients.producer.ProducerConfig)
[[redacted] 13:50:44,093] WARN The configuration 'serializer.class' was supplie
 but isn't a known config. (org.apache.kafka.clients.producer.ProducerConfig)
>1
>2
>3
>4
>5
>6
>
```

Figure 9-10

Switch to the three consumer windows. It is found that each window consumes two messages evenly.

```
[root@node-masterlnFFO bin]# sh kafka-console-consumer.sh --topic cx_topic3 --bo
otstrap-server 192.168.0.119:9092  --consumer-property group1 --consumer.config
/opt/client/Kafka/kafka/config/consumer.properties
[[redacted] 13:52:33,836] WARN The configuration 'group1' was supplied but isn't
 a known config. (org.apache.kafka.clients.consumer.ConsumerConfig)
3
6
```

the first consumer

Figure 9-11

```
[root@node-masterlnFFO ~]# source /opt/client/bigdata_env
[root@node-masterlnFFO ~]# cd  /opt/client/Kafka/kafka/bin
[root@node-masterlnFFO bin]# sh kafka-console-consumer.sh --topic cx_topic3 --bo
otstrap-server 192.168.0.119:9092  --consumer-property group1 --consumer.config
/opt/client/Kafka/kafka/config/consumer.properties
[[redacted] 13:52:10,551] WARN The configuration 'group1' was supplied but isn't
 a known config. (org.apache.kafka.clients.consumer.ConsumerConfig)
1
4
```

the second consumer

Figure 9-12

```
[root@node-masterlnFFO bin]# sh kafka-console-consumer.sh --topic cx_topic3 --bo
otstrap-server 192.168.0.119:9092  --consumer-property group1 --consumer.config
/opt/client/Kafka/kafka/config/consumer.properties
[[redacted] 13:52:39,697] WARN The configuration 'group1' was supplied but isn't
 a known config. (org.apache.kafka.clients.consumer.ConsumerConfig)
2
5
```

the third consumer

Figure 9-13

The three consumers evenly consume six messages. Each consumer processes two messages. This ensures data integrity.

Step 4 Start the fourth consumer.

Open another PuTTY window, set environment variables, go to the Kafka directory, and run the following command to start the fourth consumer.

```
sh kafka-console-consumer.sh --topic cx_topic3 --bootstrap-server 192.168.0.119:9092  --consumer-property group1 --consumer.config /opt/client/Kafka/kafka/config/consumer.properties
```

Specify the same consumer group **group1**.

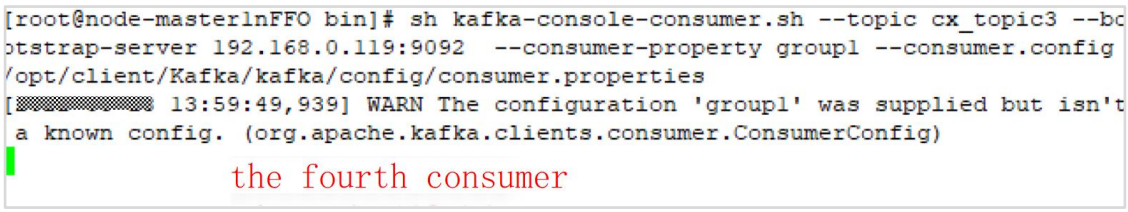


Figure 9-14

Step 5 Configure four consumers.

Continue to send six messages in sequence in the producer window.

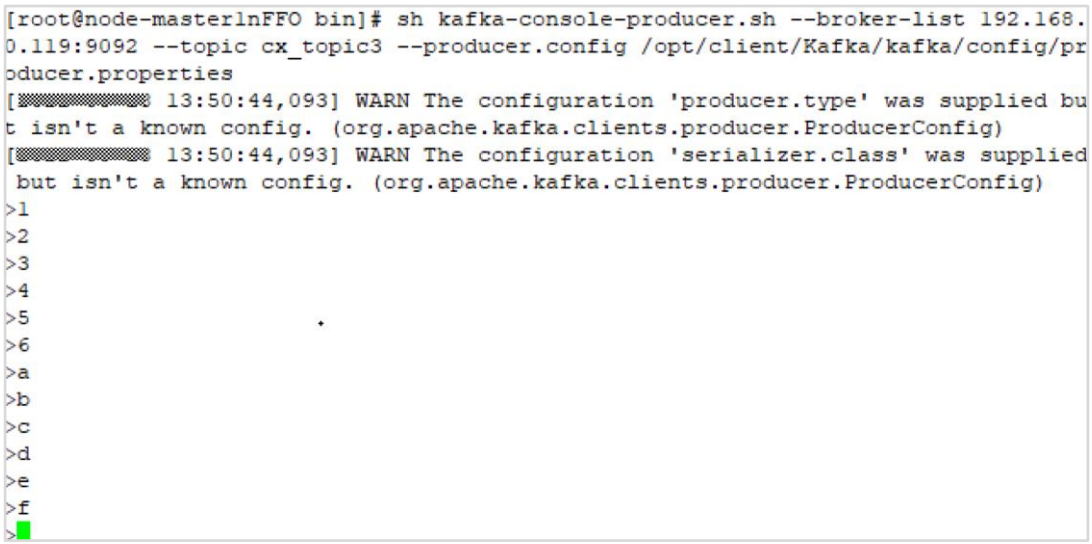


Figure 9-15

Switch to the four consumer windows.

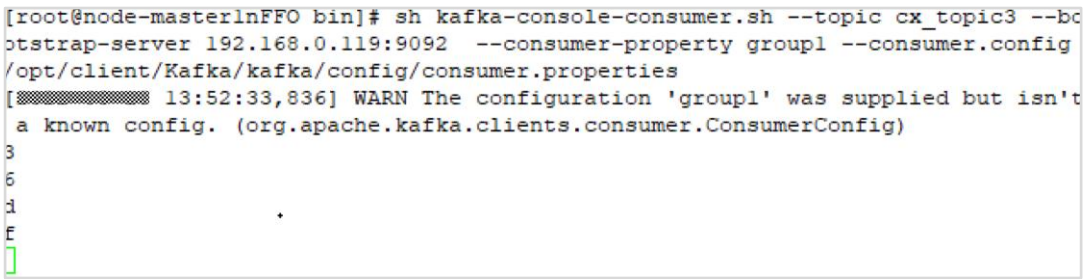


Figure 9-16

```
[root@node-masterlnFFO bin]# sh kafka-console-consumer.sh --topic cx_topic3 --bo
otstrap-server 192.168.0.119:9092  --consumer-property group1 --consumer.config
/opt/client/Kafka/kafka/config/consumer.properties
[[redacted] 13:52:10,551] WARN The configuration 'group1' was supplied but isn't
 a known config. (org.apache.kafka.clients.consumer.ConsumerConfig)
1
4
a
c
```

Figure 9-17

```
[root@node-masterlnFFO bin]# sh kafka-console-consumer.sh --topic cx_topic3 --bo
otstrap-server 192.168.0.119:9092  --consumer-property group1 --consumer.config
/opt/client/Kafka/kafka/config/consumer.properties
[[redacted] 13:52:39,697] WARN The configuration 'group1' was supplied but isn't
 a known config. (org.apache.kafka.clients.consumer.ConsumerConfig)
2
5
b
e
```

Figure 9-18

```
[root@node-masterlnFFO bin]# sh kafka-console-consumer.sh --topic cx_topic3 --bo
otstrap-server 192.168.0.119:9092  --consumer-property group1 --consumer.config
/opt/client/Kafka/kafka/config/consumer.properties
[[redacted] 13:59:49,939] WARN The configuration 'group1' was supplied but isn't
 a known config. (org.apache.kafka.clients.consumer.ConsumerConfig)
```

Figure 9-19

As shown in the preceding figure, a consumer does not have a corresponding partition, and the consumer cannot obtain messages. Therefore, when creating topics, you can create more partitions to ensure that multiple consumers can correspond to the partitions, preventing consumers from being wasted. To add partitions, you can run the **kafka-reassign-partitions.sh** command.

Step 6 Configure two consumers.

Disable two consumers by pressing **Ctrl+C** and retain the remaining two consumers.

```
[root@node-masterlnFFO bin]# sh kafka-console-consumer.sh --topic cx_topic3 --bo
otstrap-server 192.168.0.119:9092  --consumer-property group1 --consumer.config
/opt/client/Kafka/kafka/config/consumer.properties
[[redacted] 13:52:33,836] WARN The configuration 'group1' was supplied but isn't
 a known config. (org.apache.kafka.clients.consumer.ConsumerConfig)
3
6
d
f
^CProcessed a total of 4 messages
[root@node-masterlnFFO bin]#
```

Figure 9-20

```
[root@node-masterlnFFO bin]# sh kafka-console-consumer.sh --topic cx_topic3 --bo
otstrap-server 192.168.0.119:9092  --consumer-property group1 --consumer.config
/opt/client/Kafka/kafka/config/consumer.properties
[[redacted] 13:52:10,551] WARN The configuration 'group1' was supplied but isn't
 a known config. (org.apache.kafka.clients.consumer.ConsumerConfig)
1
4
a
c
^CProcessed a total of 4 messages
[root@node-masterlnFFO bin]#
```

Figure 9-21

Enter the following 10 messages in sequence in the producer window:

```
[root@node-masterlnFFO bin]# sh kafka-console-producer.sh --broker-list 192.168
0.119:9092 --topic cx_topic3 --producer.config /opt/client/Kafka/kafka/config/p
oducer.properties
[[redacted] 13:50:44,093] WARN The configuration 'producer.type' was supplied b
t isn't a known config. (org.apache.kafka.clients.producer.ProducerConfig)
[[redacted] 13:50:44,093] WARN The configuration 'serializer.class' was supplie
 but isn't a known config. (org.apache.kafka.clients.producer.ProducerConfig)
>1
>2
>3
>4
>5
>6
>a
>b
>c
>d
>e
>f
>111
>222
>333
>444
>555
>666
>777
>888
>999
>1010
>
```

Figure 9-22

Check the status of the two consumer windows.

```
[root@node-masterlnFFO bin]# sh kafka-console-consumer.sh --topic cx_topic3 --bo
otstrap-server 192.168.0.119:9092  --consumer-property group1 --consumer.config
/opt/client/Kafka/kafka/config/consumer.properties
[[redacted] 13:59:49,939] WARN The configuration 'group1' was supplied but isn't
 a known config. (org.apache.kafka.clients.consumer.ConsumerConfig)
222
444
666
888
```

Figure 9-23

```
[root@node-master1nFFO bin]# sh kafka-console-consumer.sh --topic cx_topic3 --bo
otstrap-server 192.168.0.119:9092  --consumer-property group1 --consumer.config
/opt/client/Kafka/kafka/config/consumer.properties
[[redacted] 13:52:39,697] WARN The configuration 'group1' was supplied but isn't
 a known config. (org.apache.kafka.clients.consumer.ConsumerConfig)
2
5
b
e
111
333
555
777
999
1010
```

Figure 9-24

When two consumers are used, the unexpected phenomenon also occurs. That is, messages are not evenly distributed. Instead, one consumer is allocated six messages and the other four messages. The reason is that one consumer corresponds to two partitions, and the other corresponds to one partition. Messages are consumed based on partitions.

Check the partitions of the **cx_topic3** topic.

View the consumer group list.

```
./kafka-consumer-groups.sh  --bootstrap-server 192.168.0.119:9092  --list
```

```
[root@node-master1nFFO bin]# sh kafka-consumer-groups.sh  --bootstrap-server 19
2.168.0.119:9092  --list
__KafkaMetricReportGroup
example-group1
[root@node-master1nFFO bin]#
```

Figure 9-25

View the details about the consumer group.

```
./kafka-consumer-groups.sh  --bootstrap-server 192.168.0.119:9092  --describe  --group example-group1
```

```
[root@node-master1nFFO bin]# ./kafka-consumer-groups.sh  --bootstrap-server 192.168.0.119:9092  --describe  --group example-group1

GROUP           TOPIC           PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID
example-group1  cx_topic3       0          16              16              0               consumer-example-group1-1-6b6a2d37-5ac7-4822-a55f-4845dd082d2a
example-group1  cx_topic3       1          12              12              0               consumer-example-group1-1-6b6a2d37-5ac7-4822-a55f-4845dd082d2a
example-group1  cx_topic3       2          19              19              0               consumer-example-group1-1-cc836f24-825d-43e2-aeea-1261eec86564
example-group1  cx_topic2       0          6               6               0               -
```

Figure 9-26

As the figure shows, the **consumer_id** values of **partition0** and **partition1** are both **consumer-example-group1-1-6b6a2d37-5ac7-4822-a55f-4845dd082d2a**.

Sometimes, the data consumption sequences of different partitions are different. This is because Kafka messages are stored by partition, and only messages in the same partition are pulled in sequence.

9.3 Exercise Summary

9.3.1 Quiz

What are the functions of offset in Kafka?

9.3.2 Summary

This exercise describes how to generate and consume data in real time on the shell end and enables trainees to have a deeper understanding of Kafka. Multiple consumers in the same consumer group are equivalent to one consumer, which improves consumption efficiency.

10 Comprehensive Exercise: Hive Data Warehouse

10.1 About This Exercise

10.1.1 Overview

In the big data services, multiple components form a service system. The following two exercises involve these components.

The first exercise is typical data analysis. It uses Sqoop to migrate MySQL database data to Hive, and uses the big data processing capability of Hive to analyze related results.

The second exercise uses Sqoop to migrate MySQL database data to HBase, and then uses HBase to query and analyze data.

10.1.2 Objectives

Use big data components to convert and query data in real time.

10.2 Tasks

Use Sqoop, Hive, and HBase big data to import and query data.

Data is imported from the MySQL database to Hive, and then imported from MySQL to HBase for data analysis.

10.2.1 Task 1: Applying for the MySQL Service

Step 1 Apply for the MySQL service.

Log in to the Huawei Cloud website, choose **Products** > **Databases**, and find **RDS for MySQL**.

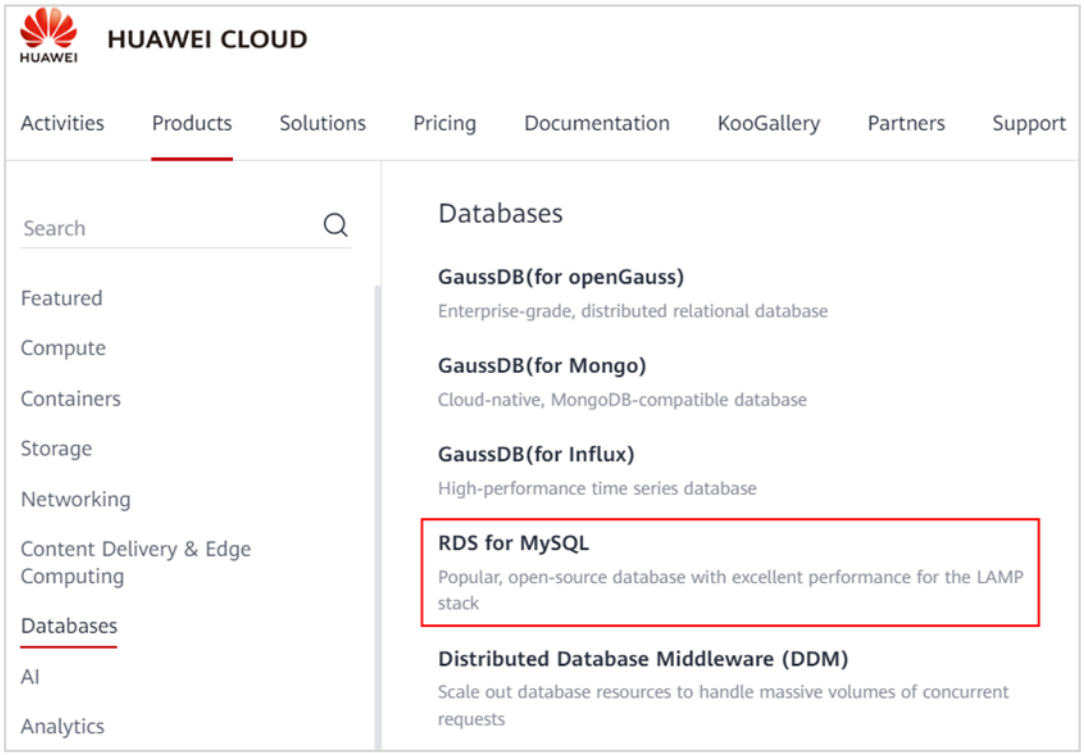


Figure 10-1

Click **Buy Now** and configure the database instance information as follows:

Billing Mode: Click **Pay-per-use**.

Region: Select **CN-Hong Kong** (the same region as MRS)

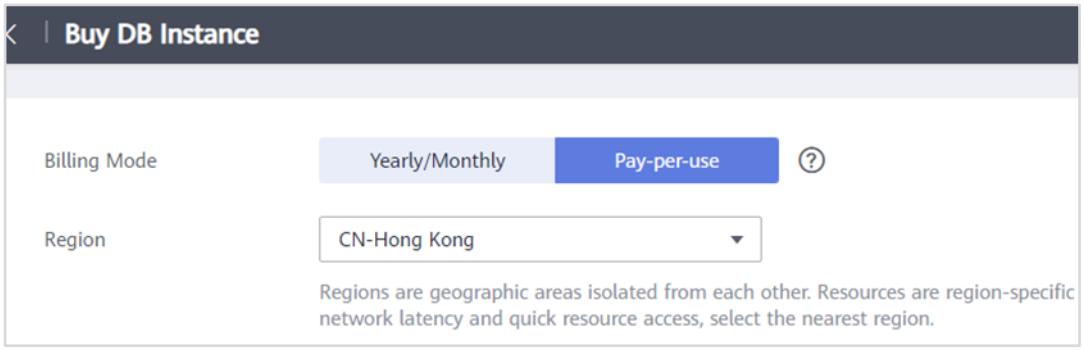


Figure 10-2

DB Instance Name: Enter a custom name. In this exercise, **rds_hcia** is used as an example. The instance name must start with a letter and contain 4 to 64 characters. Only letters, digits, hyphens (-), and underscores (_) are allowed.

DB Engine: Select **MySQL**.

DB Engine Version: Select **5.6**.

DB Instance Type: Select **Single**.

Storage Type: Select **Cloud SSD**.

AZ: Retain the default value.

Time Zone: Retain the default value.

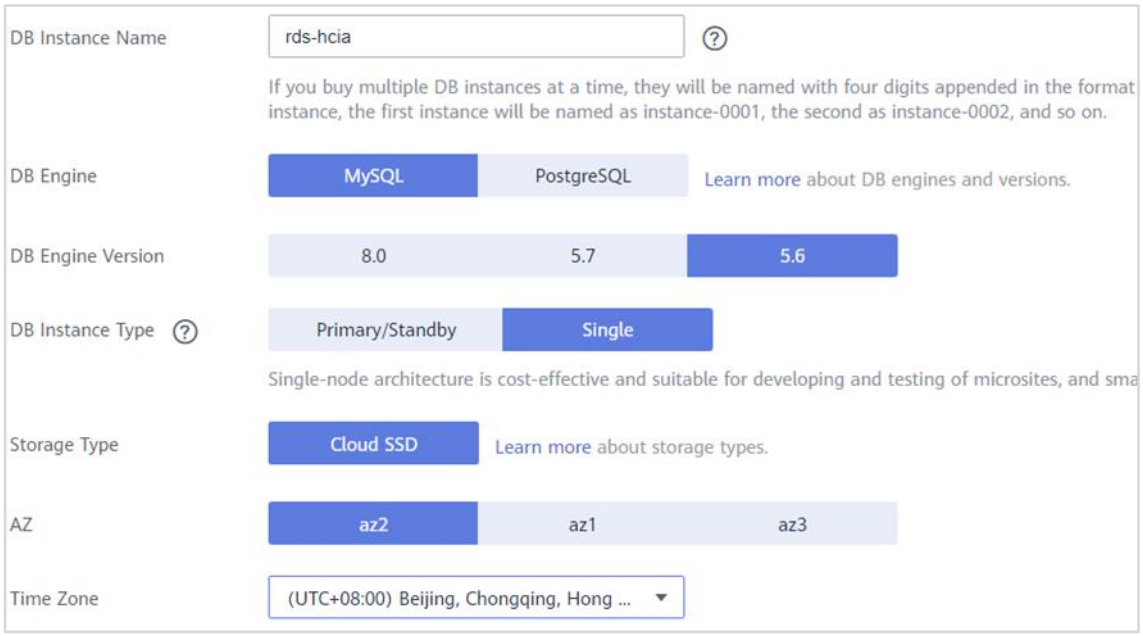


Figure 10-3

Instance Class: The default value is **2 vCPUs | 4 GB**.

Storage Space (GB): The default value is **40** GB.

Disk Encryption: Select **Disable**.

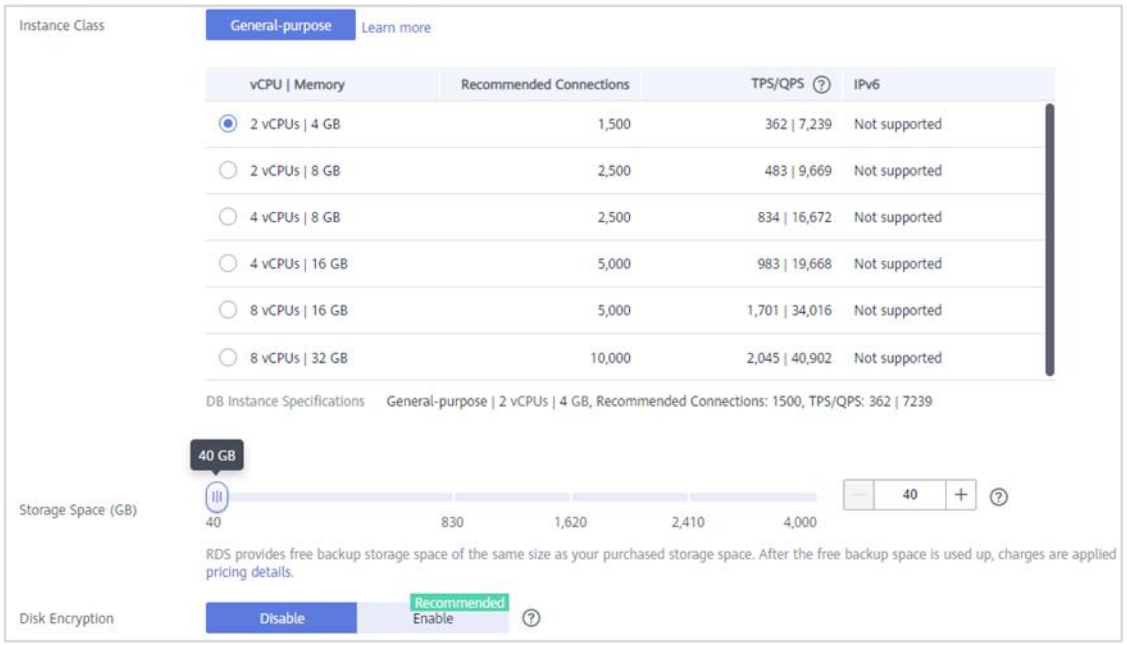


Figure 10-4

VPC: Retain the default value, which is the same network as MRS.

Database Port: Retain the default value.

Security Group: Retain the default value, which is the same security group as MRS.

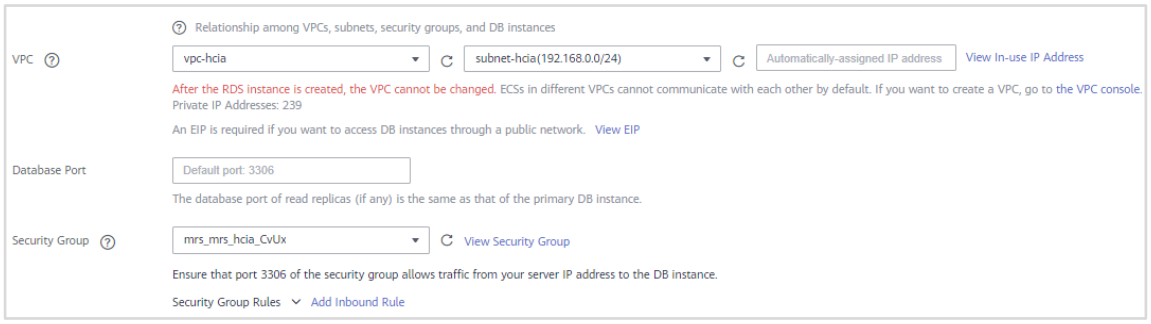


Figure 10-5

Password: Select **Configure**.

Administrator: The default value is **root**.

Administrator Password: Set the password as required.

Retain the default values for other parameters.

Quantity: The default value is **1**.

Confirm the configurations and click **Next**.

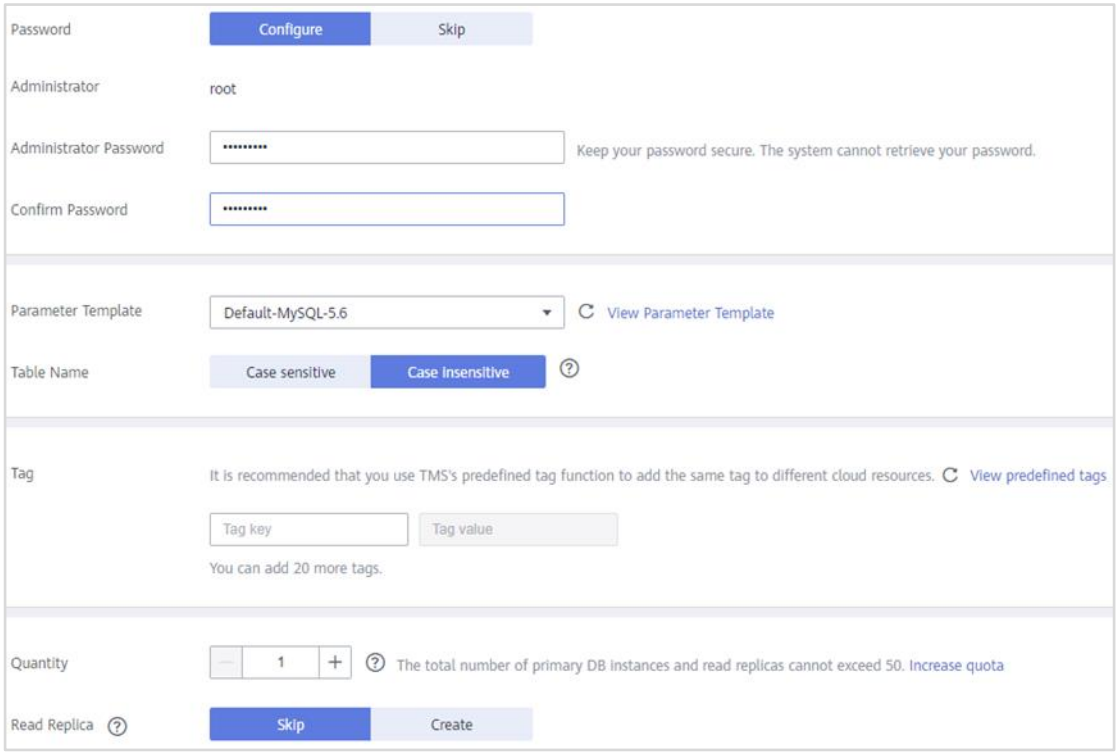


Figure 10-6

Step 2 Log in to MySQL.

After the MySQL database is created, click **Log In**.

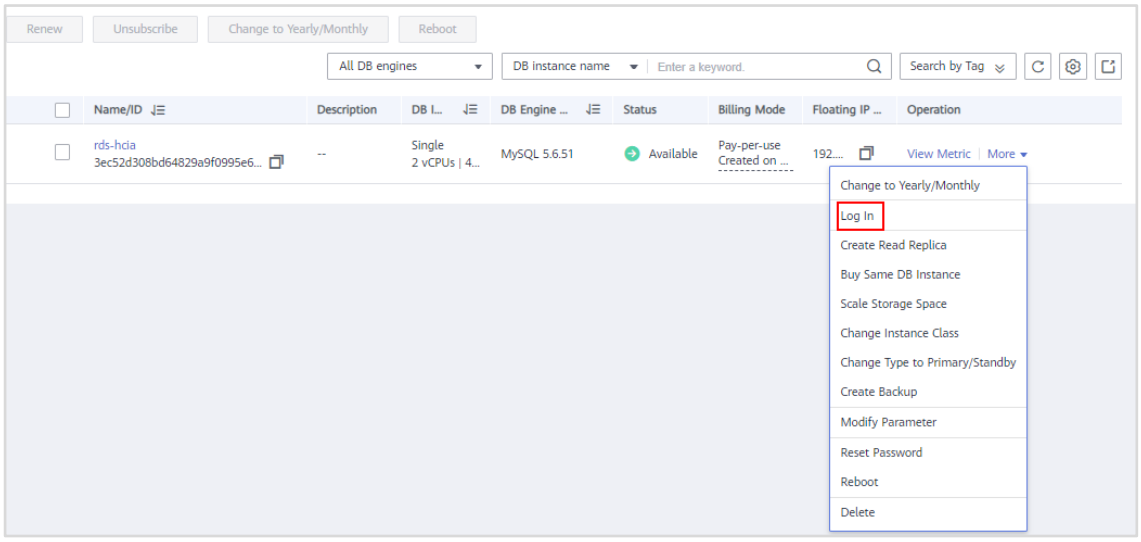


Figure 10-7

Enter username **root** and its password.

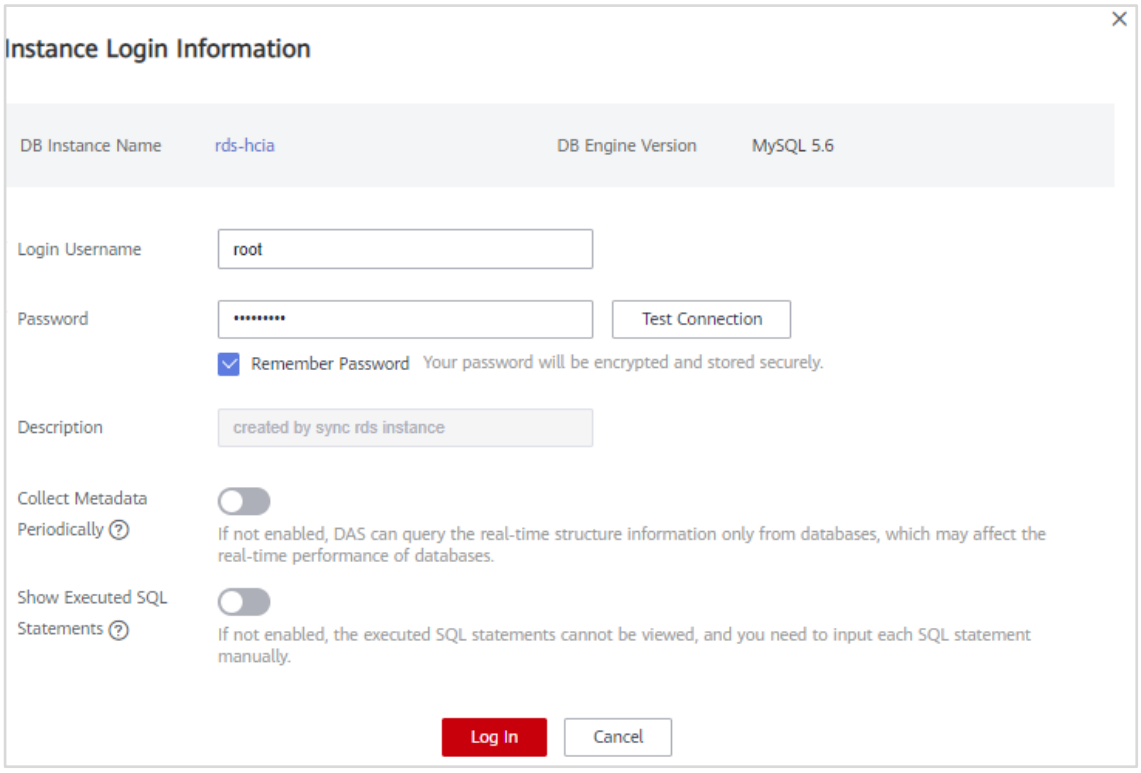


Figure 10-8

Click **Test Connection**. After the test is successful, click **Log In**. The MySQL data service management page is displayed.

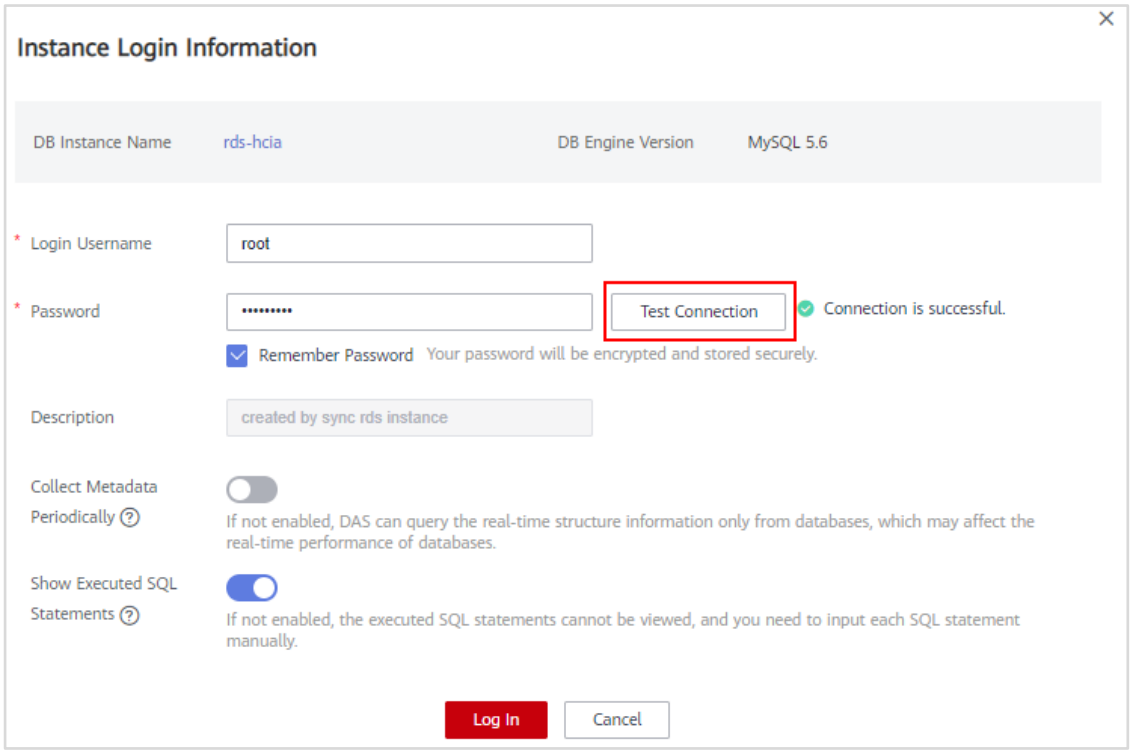


Figure 10-9

Step 3 Create a database.

Click **Create Database**, enter a database name, for example, **rdsdb**, set **Character Set** to **utf8**, and click **OK**.

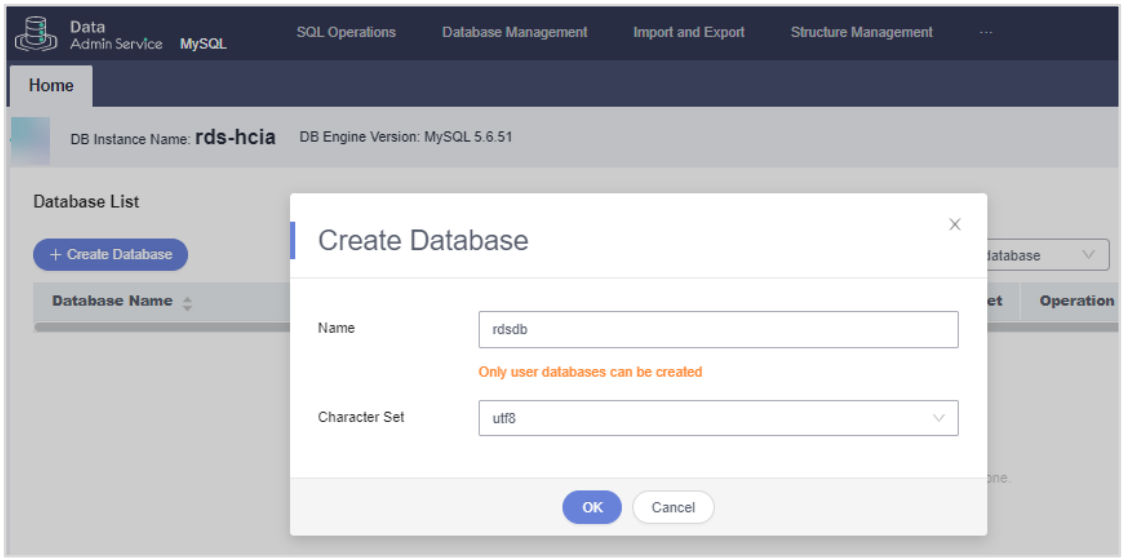


Figure 10-10

10.2.2 Task 2: Preparing MySQL Data

Step 1 Log in to the MySQL database.

Open the MySQL instance page and click **Log In**.

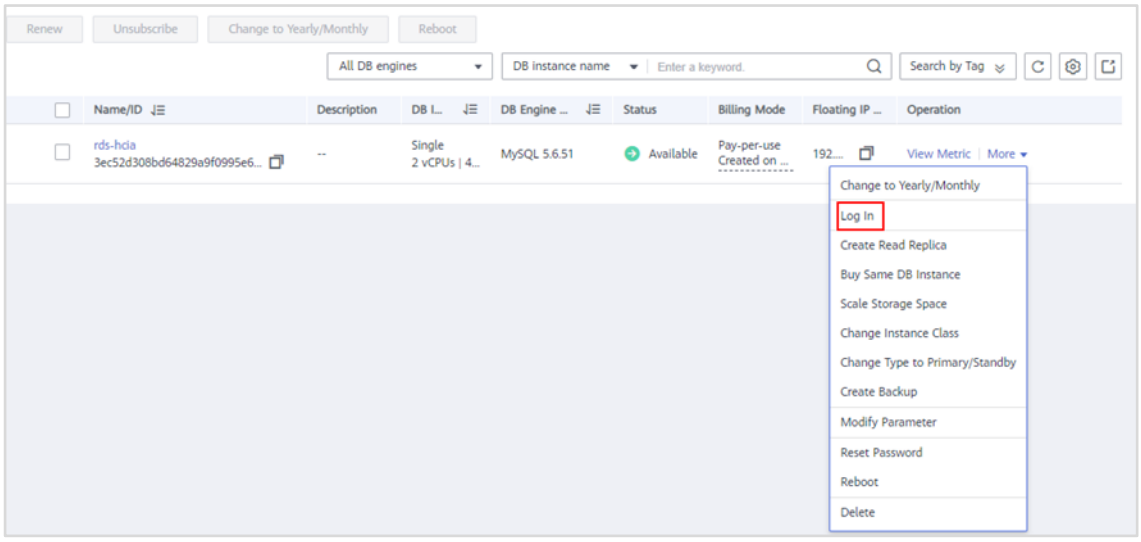


Figure 10-11

Step 2 Create the **cx_socker** table and set **timestr** as the primary key.

Click the **rdsdb** database to access it, and click **create table** to create the **cx_socker** table.

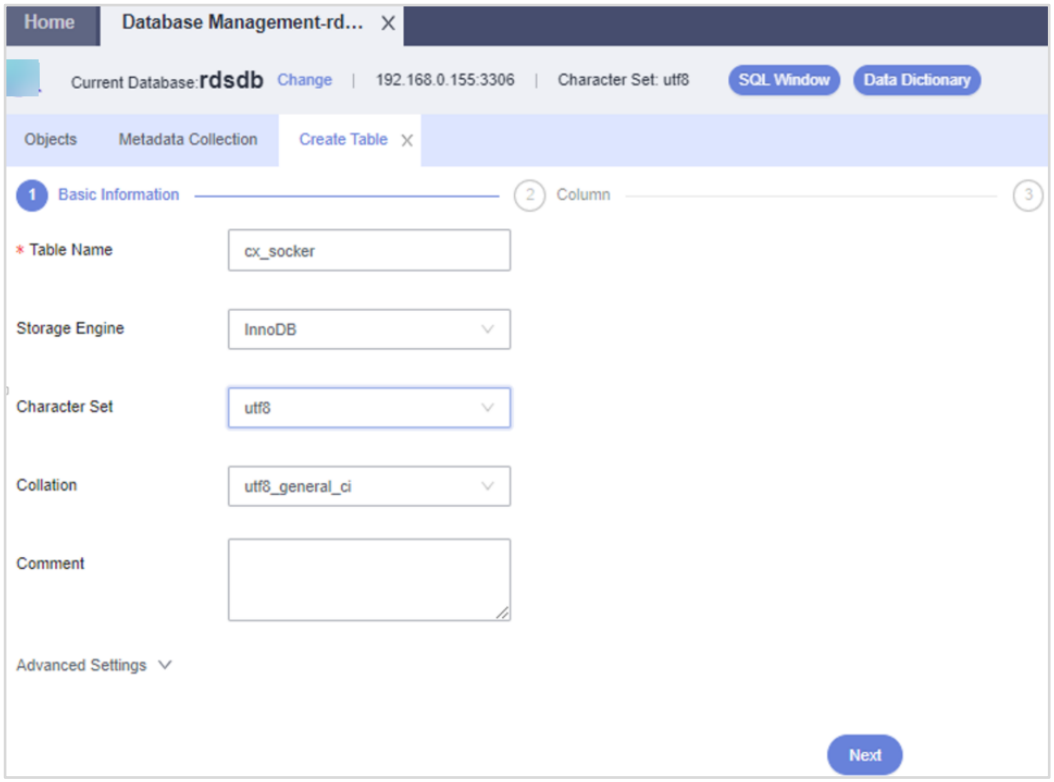


Figure 10-12

The fields to be created are as follows.

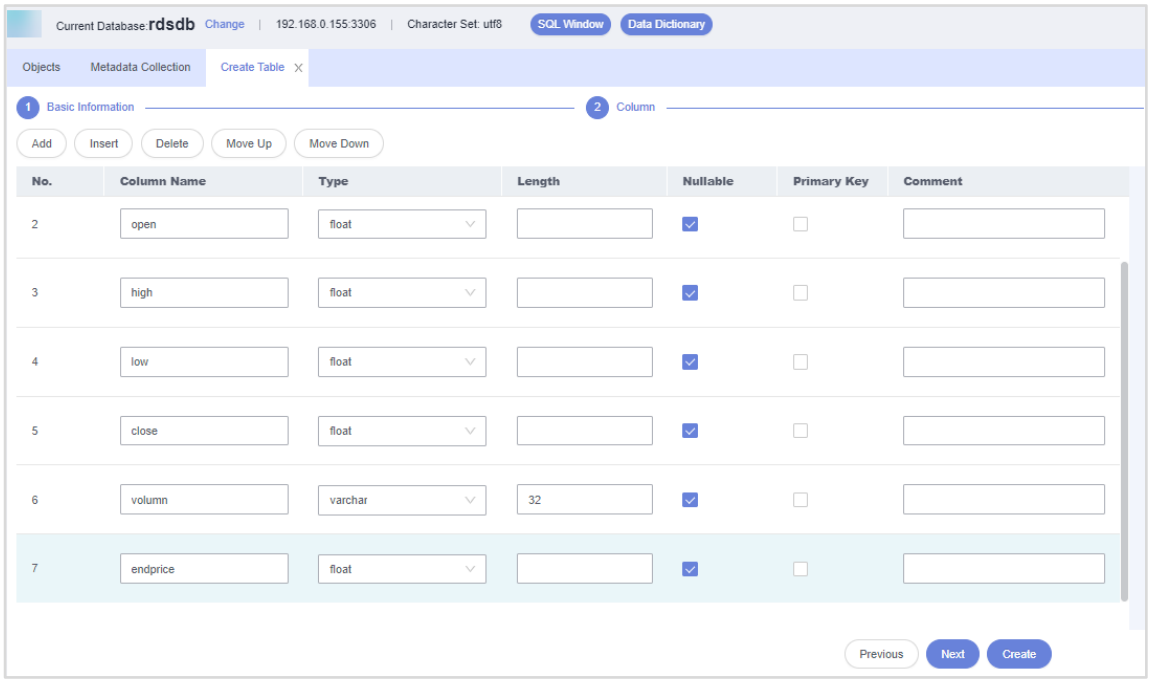


Figure 10-13

Click **Create Now** and execute the script.

Step 3 Import data to **cx_socker**.

On the MySQL management page, choose **Import and Export** > **Import**.

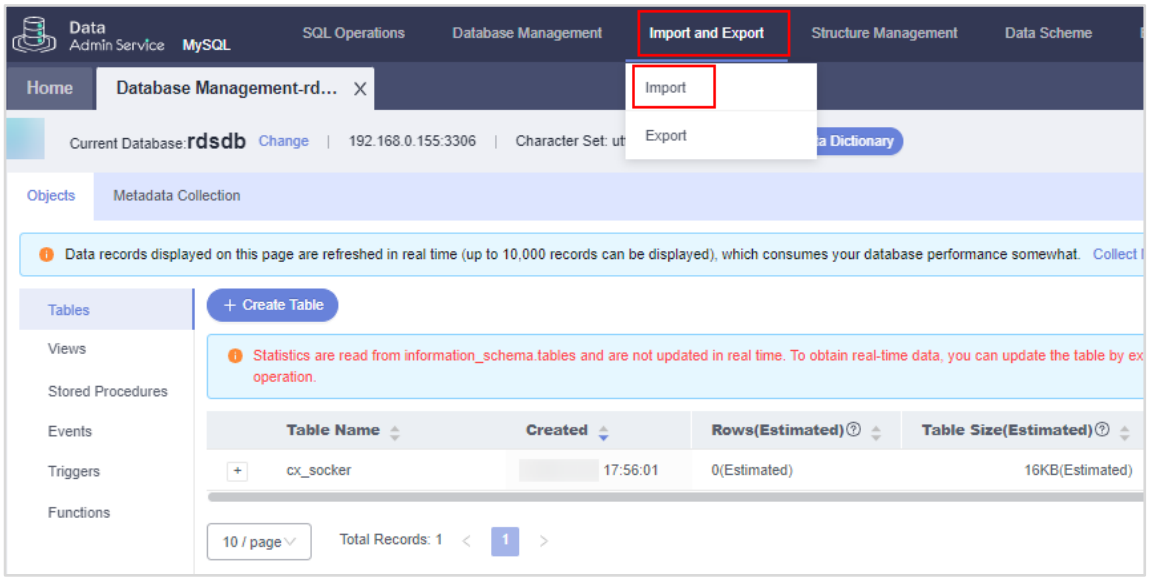


Figure 10-14

Click **Create Task**. By default, no bucket is available. Click **Create OBS Bucket**.

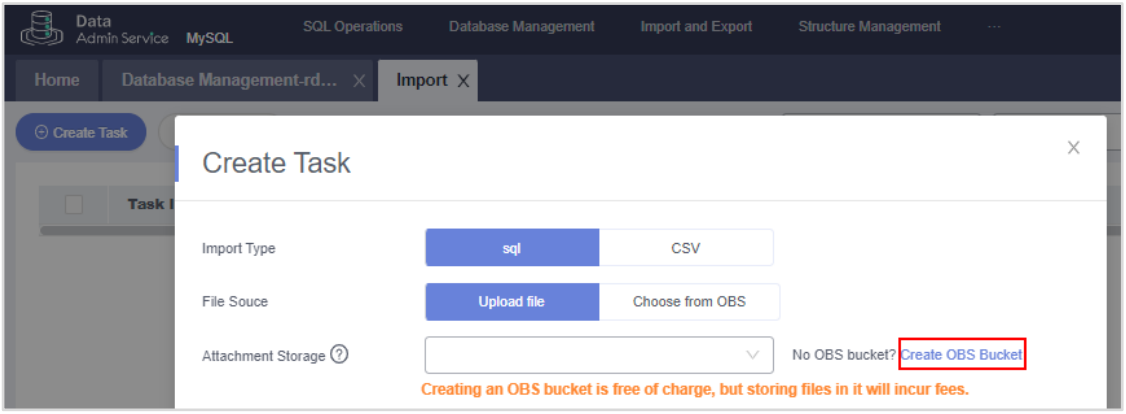


Figure 10-15

Click **OK** to return to the import page. Select data file **sp500.csv** and upload it. The configuration is as follows:

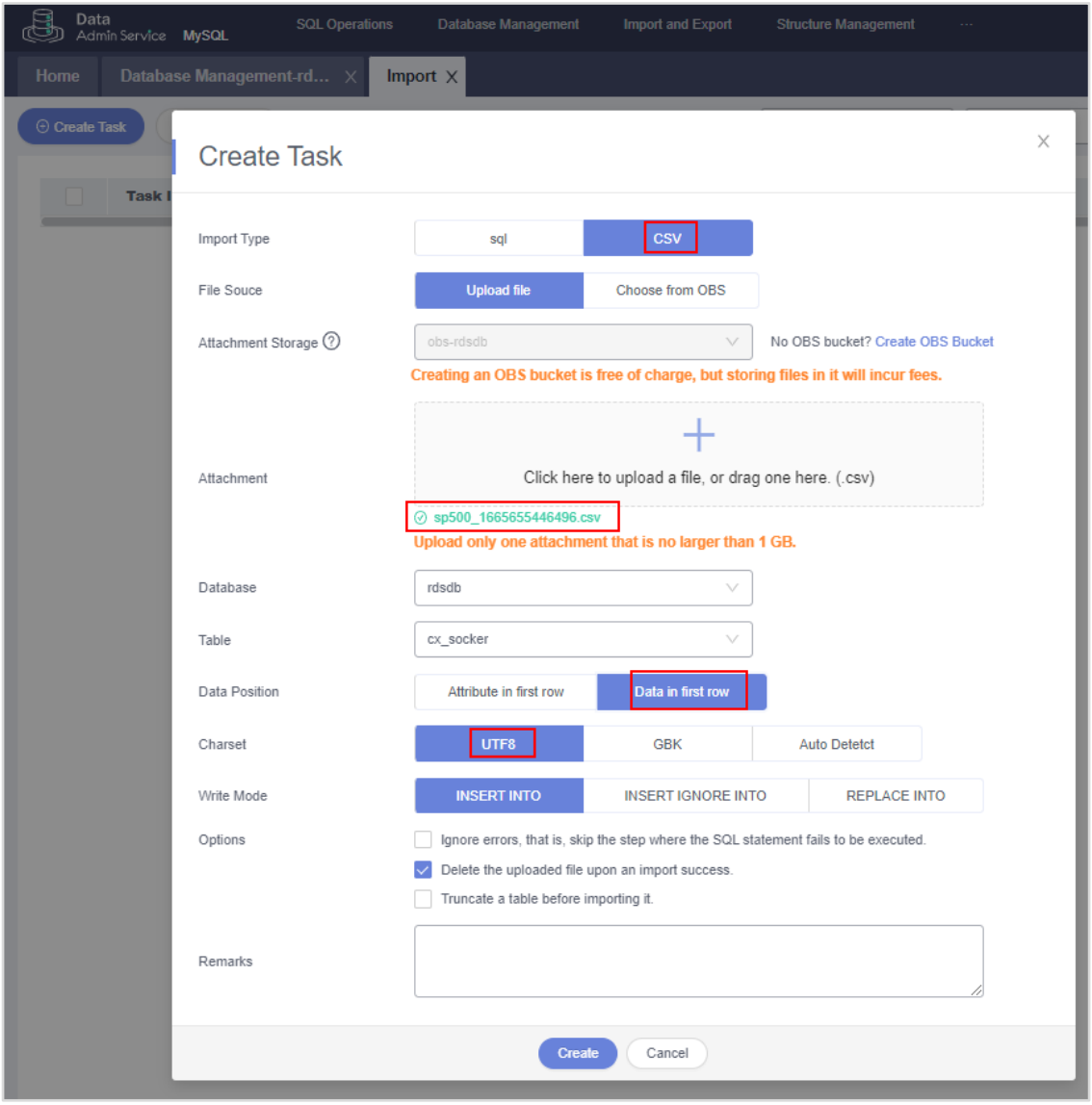


Figure 10-16

Click **Create** and wait for the task to be executed. If the task is successfully executed, the following information is displayed.

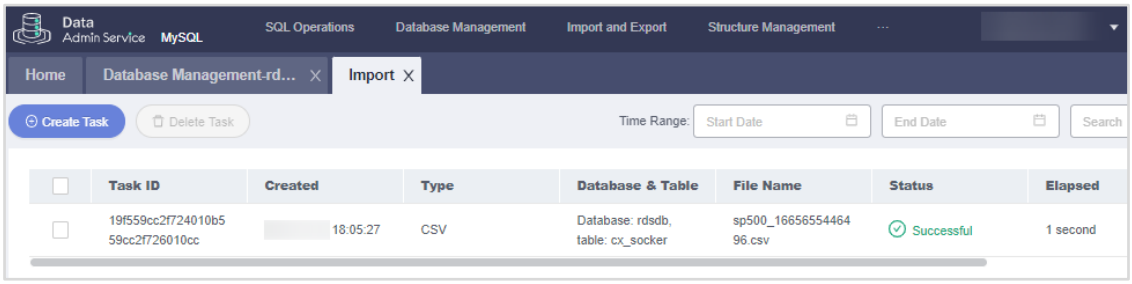


Figure 10-17

Step 4 View data in **cx_socker**.

On the **Database Management** page, click **Query SQL Statements**.

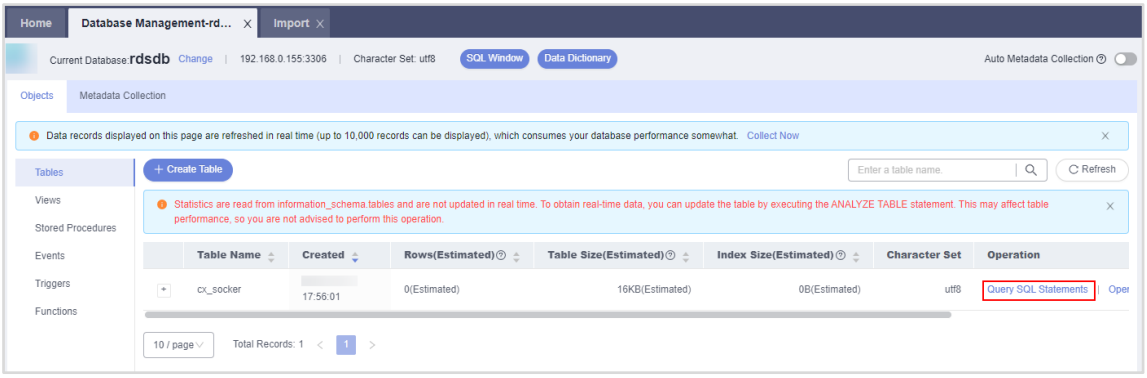


Figure 10-18

Detailed data is shown as follows.

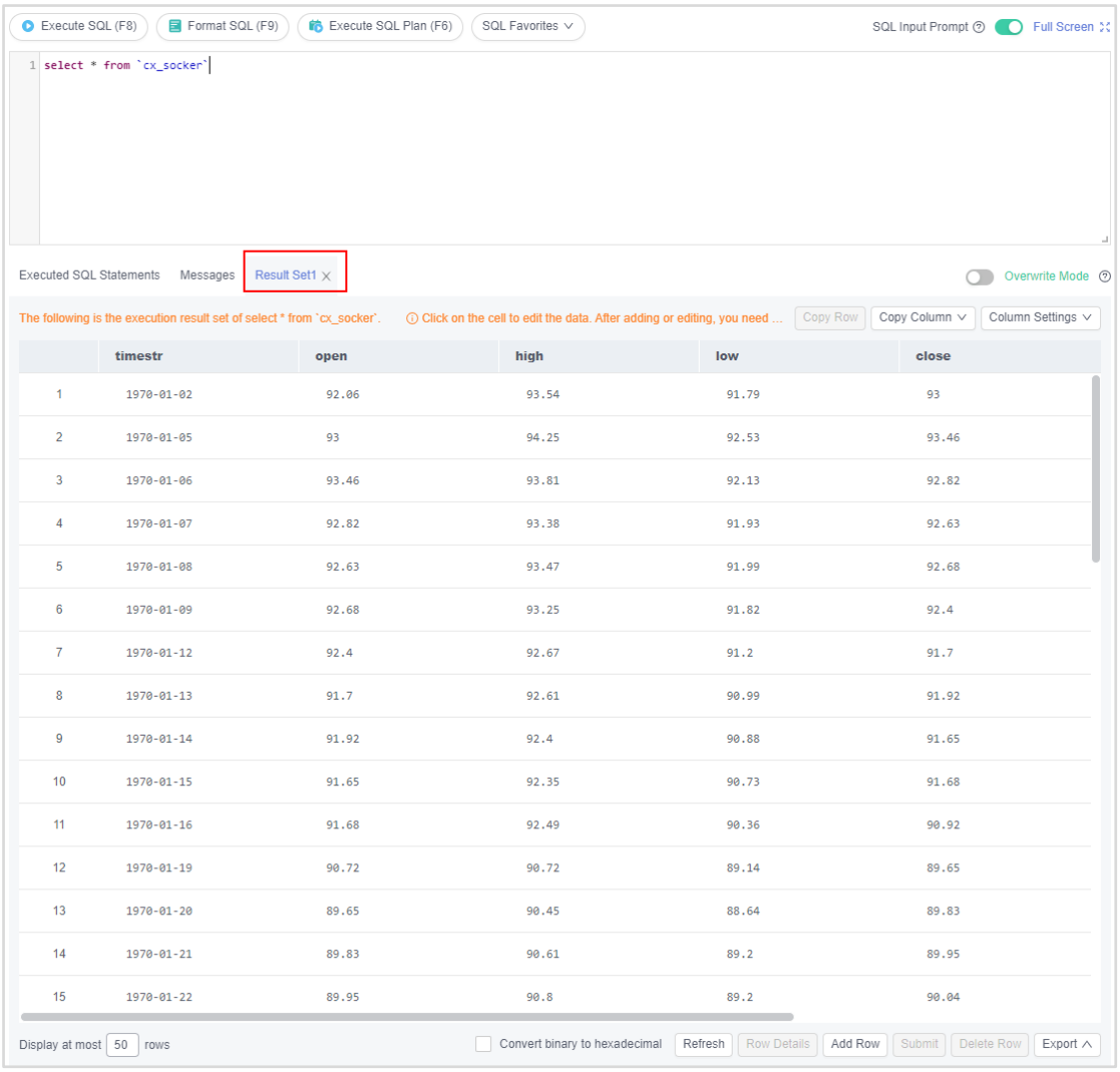


Figure 10-19

10.2.3 Task3: Importing MySQL Data to Hive

Step 1 Create a table in Hive.

Log in to Hive and run the following command to create the **cx_hive_socker** table:

```
create table cx_hive_socker (timestr string,open float,high float,low float,close float,volume string,endprice float) row format delimited fields terminated by ',' stored as textfile;
```

```
0: jdbc:hive2://192.168.0.70:10000/> create table cx_hive_socker (timestr string,open
 float,high float,low float,close float,volume string,endprice float) row format deli
mited fields terminated by ',' stored as textfile;
No rows affected (1.782 seconds)
0: jdbc:hive2://192.168.0.70:10000/>
```

Figure 10-20

The **cx_hive_socker** table does not contain data.

Step 2 Import the data to the Hive table.

Run the following command to initialize environment variables:

```
source /opt/client/bigdata_env
```

Run the **sqoop** command to operate the sqoop client. Run the following command to connect to the MySQL database and list the database names.

```
sqoop list-database --connect jdbc:mysql://116.63.197.140:3306/    --username root --password xxx
```

Note: Replace **xxx** in the command with the password of MySQL database user **root**.

```
[root@node-master1GZfa ~]# source /opt/client/bigdata_env
[root@node-master1GZfa ~]# sqoop list-databases --connect jdbc:mysql://116.63.197.140
:3306/ --username root --password [redacted]
Warning: /opt/client/Sqoop/sqoop/../accumulo does not exist! Accumulo imports will fa
il.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/slf
4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/common
/lib/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/client/Hive/HCatalog/lib/slf4j-log4j12-1.7.30.
jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/client/HBase/hbase/geomesa/lib/slf4j-log4j12-1
.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/client/HBase/hbase/ranger-2.0.0-hbase-plugin/i
nstall/lib/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/client/HBase/hbase/lib/client-facing-thirdpart
y/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/client/HBase/hbase/lib/jdbc/slf4j-log4j12-1.7.
30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/client/HBase/hbase/tools/hbase-hbck2-2.2.3-hw-
ei-310012.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/client/HBase/hbase/tools/hbase-tools-2.2.3-hw-
ei-310012.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]
[redacted] 14:04:11,835 INFO sqoop.Sqoop: Running Sqoop version: 1.4.7
[redacted] 14:04:11,873 WARN tool.BaseSqoopTool: Setting your password on the command
-line is insecure. Consider using -P instead.
[redacted] 14:04:12,029 INFO manager.MySQLManager: Preparing to use a MySQL streaming
 resultset.
information_schema
mysql
performance_schema
rdsdb
```

Figure 10-21

Run the following command to import the data table from the MySQL database to the Hive table:

```
sqoop import --connect jdbc:mysql://116.63.197.140:3306/rdsdb --username root --password xxx --
table cx_socker --hive-import --hive-table cx_hive_socker --delete-target-dir --fields-terminated-by "," 
-m 1 --as-textfile
```

```
[root@node-master1GZfa ~]# sqoop import --connect jdbc:mysql://116.63.197.140:3306/rdsdb --usernam
e root --password Xxal@0814 --table cx_socker --hive-import --hive-table cx_hive_socker --delete-t
arget-dir --fields-terminated-by ','  --m 1 --as-textfile
```

Figure 10-22

The command is successfully executed if the following information is displayed.

```
[redacted] 17:20:24,567 INFO ql.Driver: OK
[redacted] 17:20:24,568 INFO ql.Driver: Concurrency mode is disabled, not creating a lock manager
Time taken: 0.608 seconds
[redacted] 17:20:24,568 INFO CliDriver: Time taken: 0.608 seconds
[redacted] 17:20:24,574 INFO session.SessionState: Deleted directory: /tmp/hive/root/d46976eb-57cd
-4ef1-bd62-58269fa2lbe6 on fs with scheme hdfs
[redacted] 17:20:24,575 INFO session.SessionState: Deleted directory: /opt/client/HDFS/hadoop/root
/d46976eb-57cd-4ef1-bd62-58269fa2lbe6 on fs with scheme file
[redacted] 17:20:24,575 INFO metastore.HiveMetaStoreClient: Closed a connection to metastore, curr
ent connections: 1
[redacted] 17:20:24,575 INFO hive.HiveImport: Hive import complete.
[redacted] 17:20:24,576 INFO hive.HiveImport: Export directory is contains the _SUCCESS file only,
 removing the directory.
[root@node-master1GZfa ~]#
```

Figure 10-23

Step 3 View data in Hive.

Run the following command in Hive to check whether the data is successfully imported:

```
select * from cx_hive_socker limit 10;
```

```
0: jdbc:hive2://192.168.0.58:10000/> select * from cx_hive_socker limit 10;
+--------------------------+----------------------+----------------------+----------------------+-
----------------------+-------------------------+--------------------------+
| cx_hive_socker.timestr  | cx_hive_socker.open  | cx_hive_socker.high  | cx_hive_socker.low  |
cx_hive_socker.close  | cx_hive_socker.volume  | cx_hive_socker.endprice  |
+--------------------------+----------------------+----------------------+----------------------+-
----------------------+-------------------------+--------------------------+
| 1970-01-02               | 92.06                | 93.54                | 91.79               |
93.0                  | 8050000                | 93.0                     |
| 1970-01-05               | 93.0                 | 94.25                | 92.53               |
93.46                 | 11490000               | 93.46                    |
| 1970-01-06               | 93.46                | 93.81                | 92.13               |
92.82                 | 11460000               | 92.82                    |
| 1970-01-07               | 92.82                | 93.38                | 91.93               |
92.63                 | 10010000               | 92.63                    |
| 1970-01-08               | 92.63                | 93.47                | 91.99               |
92.68                 | 10670000               | 92.68                    |
| 1970-01-09               | 92.68                | 93.25                | 91.82               |
92.4                  | 9380000                | 92.4                     |
| 1970-01-12               | 92.4                 | 92.67                | 91.2                |
91.7                  | 8900000                | 91.7                     |
| 1970-01-13               | 91.7                 | 92.61                | 90.99               |
91.92                 | 9870000                | 91.92                    |
| 1970-01-14               | 91.92                | 92.4                 | 90.88               |
91.65                 | 10380000               | 91.65                    |
| 1970-01-15               | 91.65                | 92.35                | 90.73               |
91.68                 | 11120000               | 91.68                    |
+--------------------------+----------------------+----------------------+----------------------+-
----------------------+-------------------------+--------------------------+
10 rows selected (0.209 seconds)
0: jdbc:hive2://192.168.0.58:10000/>
```

Figure 10-24

10.2.4 Task 4: Processing Hive Data

Obtain the latest stock growth data and save the result to a new Hive table.

Step 1 Create a table in Hive.

Run the following command in the Hive shell to create a table:

```
create table cx_up_hive_socker like cx_hive_socker;
```

```
0: jdbc:hive2://192.168.0.58:10000/> create table cx_up_hive_socker like cx_hive_socker;
No rows affected (0.218 seconds)
0: jdbc:hive2://192.168.0.58:10000/> show tables;
+--------------------+
|      tab_name      |
+--------------------+
| cx_hive_socker     |
| cx_up_hive_socker  |
+--------------------+
2 rows selected (0.031 seconds)
0: jdbc:hive2://192.168.0.58:10000/>
```

Figure 10-25

Step 2 Insert data.

```
insert into cx_up_hive_socker select * from cx_hive_socker where cx_hive_socker.endprice>
cx_hive_socker.open sort by cx_hive_socker.endprice desc;
```

```
0: jdbc:hive2://192.168.0.58:10000/> select * from cx_up_hive_socker limit 10;
+---------------------------+--------------------------+--------------------------+--------------
---------+---------------------------+--------------------------+----------------------------+
| cx_up_hive_socker.timestr  | cx_up_hive_socker.open  | cx_up_hive_socker.high  | cx_up_hive_so
cker.low  | cx_up_hive_socker.close  | cx_up_hive_socker.volume  | cx_up_hive_socker.endprice  |
+---------------------------+--------------------------+--------------------------+--------------
---------+---------------------------+--------------------------+----------------------------+
| 2007-10-09                | 1553.18                  | 1565.26                  | 1551.82
         | 1565.15                  | 2932040000               | 1565.15                    |
| 2007-10-12                | 1555.41                  | 1563.03                  | 1554.09
         | 1561.8                   | 2788690000               | 1561.8                     |
| 2007-10-05                | 1543.84                  | 1561.91                  | 1543.84
         | 1557.59                  | 2919030000               | 1557.59                    |
| 2007-07-19                | 1546.13                  | 1555.2                   | 1546.13
         | 1553.08                  | 3251450000               | 1553.08                    |
| 2007-07-13                | 1547.68                  | 1555.1                   | 1544.85
         | 1552.5                   | 2801120000               | 1552.5                     |
| 2007-10-31                | 1532.15                  | 1552.76                  | 1529.4
         | 1549.38                  | 3953070000               | 1549.38                    |
| 2007-07-12                | 1518.74                  | 1547.92                  | 1518.74
         | 1547.7                   | 3489600000               | 1547.7                     |
| 2007-10-01                | 1527.29                  | 1549.02                  | 1527.25
         | 1547.04                  | 3281990000               | 1547.04                    |
| 2007-10-04                | 1539.91                  | 1544.02                  | 1537.63
         | 1542.84                  | 2690430000               | 1542.84                    |
| 2007-07-23                | 1534.06                  | 1547.23                  | 1534.06
         | 1541.57                  | 3102700000               | 1541.57                    |
+---------------------------+--------------------------+--------------------------+--------------
---------+---------------------------+--------------------------+----------------------------+
10 rows selected (0.065 seconds)
0: jdbc:hive2://192.168.0.58:10000/>
0: jdbc:hive2://192.168.0.58:10000/> insert into cx_up_hive_socker select * from cx_hive_socker
where cx_hive_socker.endprice> cx_hive_socker.open sort by cx_hive_socker.endprice desc;
No rows affected (28.16 seconds)
```

Figure 10-26

Step 3 Run the following command to obtain the total number of the stocks that grow:

```
select count(*) from cx_hive_socker where cx_hive_socker.endprice> cx_hive_socker.open;
```

```
0: jdbc:hive2://192.168.0.58:10000/> select count(*) from cx_hive_socker where cx_hive_socker.en
dprice> cx_hive_socker.open;
+-------+
|  _c0  |
+-------+
| 5228  |
+-------+
1 row selected (25.585 seconds)
0: jdbc:hive2://192.168.0.58:10000/>
```

Figure 10-27

10.2.5 Task 5: Importing HDFS Data to HBase

Step 1 Create a table in HBase.

Access the HBase shell and run the following statement to create a table:

```
create 'cx_hbase_socker','cf1'
```

```
hbase:001:0> create 'cx_hbase_socker','cf1'
2022-08-21 17:41:47,036 INFO  [main] client.HBaseAdmin: Operation: CREATE, Table Name: default:c
x_hbase_socker, procId: 18 completed
Created table cx_hbase_socker
Took 1.8999 seconds
=> Hbase::Table - cx_hbase_socker
hbase:002:0>
```

Figure 10-28

Step 2 Import the data to the HBase table.

After the Sqoop client is installed, the HBase dependency JAR package is not directly imported. You need to import the HBase dependency JAR package of a specified earlier version. Perform the following steps:

Log in to the node where the Sqoop client is installed as user **root**, download the JAR packages of HBase 1.6.0, and use WinSCP to upload the packages to the **lib** directory on the Sqoop client.

- hbase-client-1.6.0.jar
- hbase-common-1.6.0.jar
- hbase-protocol-1.6.0.jar
- hbase-server-1.6.0.jar

After the upload is successful, run the **ls** command to check whether the JAR packages exist. If the following information is displayed, the JAR packages are successfully uploaded.

```
-rwxr-xr-x. 1 root root   96221 May 11  2021 commons-pool-1.5.4.jar
-rwxr-xr-x. 1 root root  415578 May 11  2021 commons-vfs2-2.0.jar
-rwxr-xr-x. 1 root root 3203385 May 11  2021 hadoop-huaweicloud-2.8.3-hw-39.jar
-rw-r-----. 1 root root 1418160 Aug 21 18:10 hbase-client-1.6.0.jar
-rw-r-----. 1 root root  643964 Aug 21 18:10 hbase-common-1.6.0.jar
-rw-r-----. 1 root root 5005133 Aug 21 18:10 hbase-protocol-1.6.0.jar
-rw-r-----. 1 root root 4749054 Aug 21 18:10 hbase-server-1.6.0.jar
-rwxr-xr-x. 1 root root  706710 May 11  2021 hsqldb-1.8.0.10.jar
-rwxr-xr-x. 1 root root 1222059 May 11  2021 ivy-2.3.0.jar
-rwxr-xr-x. 1 root root   46989 May 11  2021 jackson-annotations-2.6.3.jar
-rwxr-xr-x. 1 root root  258876 May 11  2021 jackson-core-2.6.5.jar
```

Figure 10-29

After the packages are uploaded, run the **chmod 755** *Package name* command to change the permission on these packages to **755**.

```
[root@node-master1GZfa lib]# cd /opt/client/Sqoop/sqoop/lib
[root@node-master1GZfa lib]# chmod 755 hbase-client-1.6.0.jar
[root@node-master1GZfa lib]# chmod 755 hbase-common-1.6.0.jar
[root@node-master1GZfa lib]# chmod 755 hbase-protocol-1.6.0.jar
[root@node-master1GZfa lib]# chmod 755 hbase-server-1.6.0.jar
[root@node-master1GZfa lib]#
```

Figure 10-30

Check the JAR packages. The following figure shows that the permission on the JAR packages is successfully changed.

```
-rwxr-xr-x. 1 root root 3203385 May 11  2021 hadoop-huaweicloud-2.8.3-hw-39.jar
-rwxr-xr-x. 1 root root 1418160 Aug 21 18:10 hbase-client-1.6.0.jar
-rwxr-xr-x. 1 root root  643964 Aug 21 18:10 hbase-common-1.6.0.jar
-rwxr-xr-x. 1 root root 5005133 Aug 21 18:10 hbase-protocol-1.6.0.jar
-rwxr-xr-x. 1 root root 4749054 Aug 21 18:10 hbase-server-1.6.0.jar
-rwxr-xr-x. 1 root root  706710 May 11  2021 hsqldb-1.8.0.10.jar
```

Figure 10-31

Run the following command to import the data table from the MySQL database to the HBase table:

```
sqoop import --connect jdbc:mysql://116.63.197.140:3306/rdsdb --username root --password xxx --table 'cx_ socker', --hbase-table 'cx_hbase_socker' --column-family info --hbase-row-key timestr --hbase-create-table --m 1
```

Step 3 View the result.

Go to HBase and run the **scan** command to check the table data:

```
scan 'cx_hbase_socker'
```

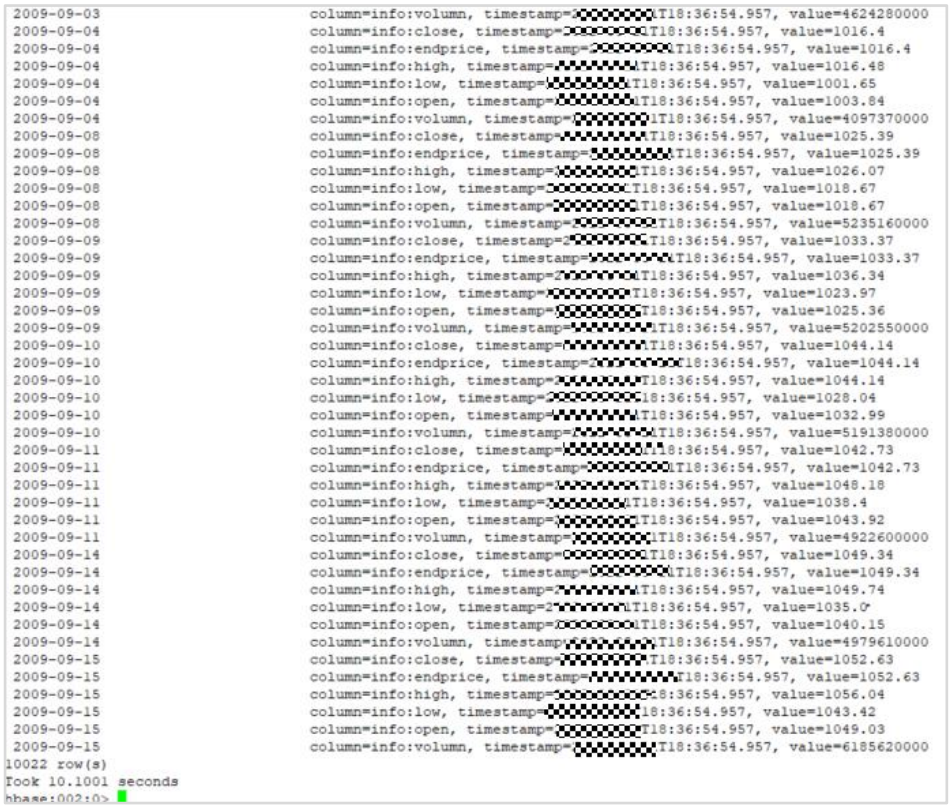


Figure 10-32

10.2.6 Task 6: Querying Data in HBase in Real Time

Step 1 Query specified records.

Run the following command in HBase:

```
get 'cx_hbase_socker','2009-09-15'
```

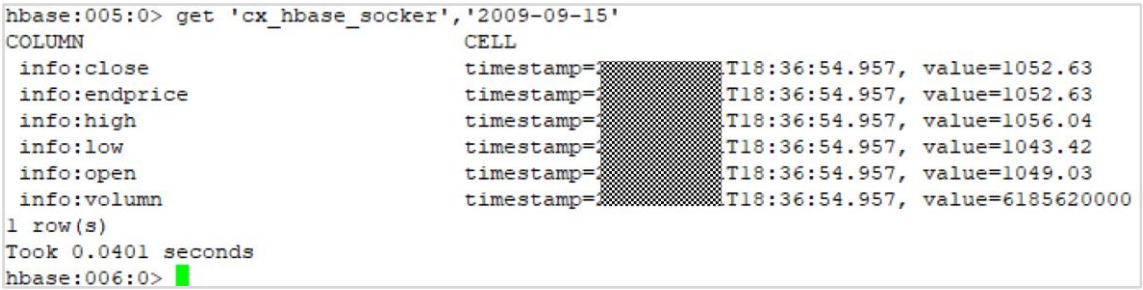


Figure 10-33

Step 2 Query the number of records in a specified period.

Run the following command in HBase:

```
scan 'cx_hbase_socker',{COLUMNS=>'info:endprice',STARTROW=>'2009-08-15',STOPROW=>'2009-09-15'}
```

```
hbase:012:0> scan 'cx_hbase_socker',{COLUMNS=>'info:endprice',STARTROW=>'2009-08-15',STOPROW=>'2009-09-15'}
ROW                           COLUMN+CELL
 2009-08-17                   column=info:endprice, timestamp=[redacted]T18:36:54.957, value=979.73
 2009-08-18                   column=info:endprice, timestamp=[redacted]T18:36:54.957, value=989.67
 2009-08-19                   column=info:endprice, timestamp=[redacted]T18:36:54.957, value=996.46
 2009-08-20                   column=info:endprice, timestamp=[redacted]T18:36:54.957, value=1007.37
 2009-08-21                   column=info:endprice, timestamp=[redacted]T18:36:54.957, value=1026.13
 2009-08-24                   column=info:endprice, timestamp=[redacted]T18:36:54.957, value=1025.57
 2009-08-25                   column=info:endprice, timestamp=[redacted]T18:36:54.957, value=1028.0
 2009-08-26                   column=info:endprice, timestamp=[redacted]T18:36:54.957, value=1028.12
 2009-08-27                   column=info:endprice, timestamp=[redacted]T18:36:54.957, value=1030.98
 2009-08-28                   column=info:endprice, timestamp=[redacted]T18:36:54.957, value=1028.93
 2009-08-31                   column=info:endprice, timestamp=[redacted]T18:36:54.957, value=1020.62
 2009-09-01                   column=info:endprice, timestamp=[redacted]T18:36:54.957, value=998.04
 2009-09-02                   column=info:endprice, timestamp=[redacted]T18:36:54.957, value=994.75
 2009-09-03                   column=info:endprice, timestamp=[redacted]T18:36:54.957, value=1003.24
 2009-09-04                   column=info:endprice, timestamp=[redacted]T18:36:54.957, value=1016.4
 2009-09-08                   column=info:endprice, timestamp=[redacted]T18:36:54.957, value=1025.39
 2009-09-09                   column=info:endprice, timestamp=[redacted]T18:36:54.957, value=1033.37
 2009-09-10                   column=info:endprice, timestamp=[redacted]T18:36:54.957, value=1044.14
 2009-09-11                   column=info:endprice, timestamp=[redacted]T18:36:54.957, value=1042.73
 2009-09-14                   column=info:endprice, timestamp=[redacted]T18:36:54.957, value=1049.34
20 row(s)
Took 0.0073 seconds
hbase:013:0>
```

Figure 10-34

Step 3 Query all columns whose values are greater than a specified value. Values are compared as strings.

Run the following command in HBase:

```
scan 'cx_hbase_socker',{FILTER => "ValueFilter(>,'binary:999')"}
```

```
hbase:013:0> scan 'cx_hbase_socker',{FILTER => "ValueFilter(>,'binary:999')"}
ROW                           COLUMN+CELL
 1970-10-22                   column=info:volumn, timestamp=[redacted]T18:36:54.351, value=9e+06
 1998-02-04                   column=info:low, timestamp=[redacted]T18:36:54.811, value=999.43
 2000-04-26                   column=info:volumn, timestamp=[redacted]T18:36:54.811, value=999600000
 2003-08-18                   column=info:close, timestamp=[redacted]T18:36:54.863, value=999.74
 2003-08-18                   column=info:endprice, timestamp=[redacted]T18:36:54.863, value=999.74
 2003-08-19                   column=info:open, timestamp=[redacted]T18:36:54.863, value=999.74
 2003-08-21                   column=info:low, timestamp=[redacted]T18:36:54.863, value=999.33
 2003-08-29                   column=info:low, timestamp=[redacted]T18:36:54.863, value=999.52
 2009-08-07                   column=info:low, timestamp=[redacted]T18:36:54.957, value=999.83
 2009-08-07                   column=info:open, timestamp=[redacted]T18:36:54.957, value=999.83
 2009-08-19                   column=info:high, timestamp=[redacted]T18:36:54.957, value=999.61
9 row(s)
Took 0.2336 seconds
hbase:014:0>
```

Figure 10-35

Step 4 Query all information ending with **endprice**. The value of the string must be greater than 999.

Run the following command in HBase:

```
scan 'cx_hbase_socker',{FILTER=>"ValueFilter(>,'binary:999') AND ColumnPrefixFilter('endprice')"}
```

```
hbase:014:0> scan 'cx_hbase_socker',{FILTER=>"ValueFilter(>,'binary:999') AND ColumnPrefixFilter('endprice')"
}
ROW                           COLUMN+CELL
 2003-08-18                   column=info:endprice, timestamp=[redacted]T18:36:54.863, value=999.74
1 row(s)
Took 0.2442 seconds
hbase:015:0>
```

Figure 10-36

10.3 Exercise Summary

10.3.1 Quiz

What are the functions of Sqoop on the big data platform?

10.3.2 Summary

These exercises integrate the applications of each component, helping trainees better understand and use big data components.

11 Appendix: Environment Preparation and Commands

11.1 (Optional) Preparing the Java Environment

11.1.1 Installing the JDK

Step 1 Download the JDK.

Visit https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html.

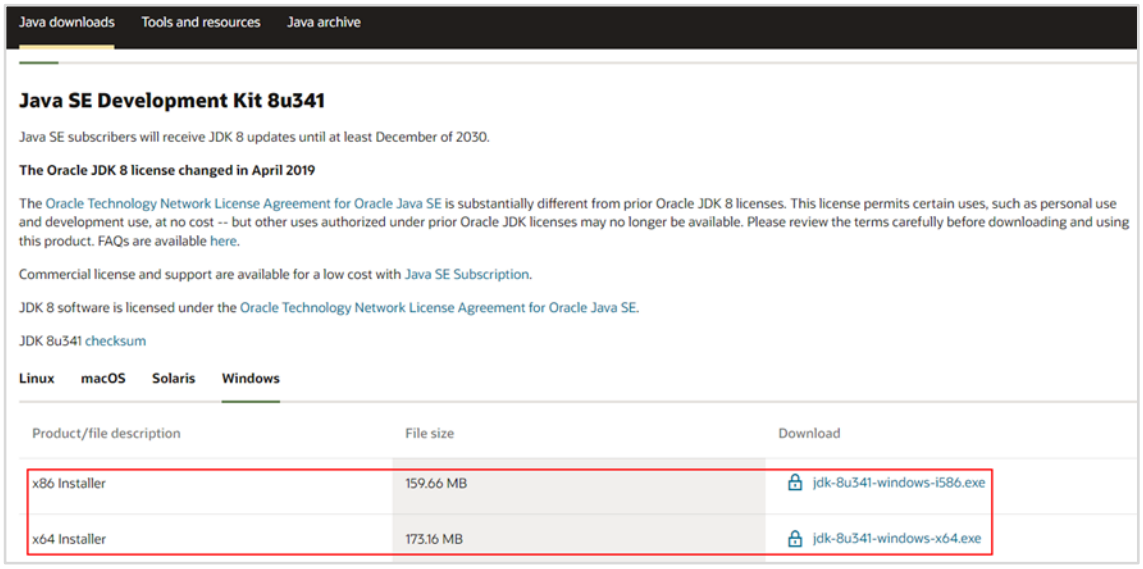


Figure 11-1

Select **I reviewed and accept...**, and download the JDK of the Windows x64 version. If the OS is 32-bit, select the x86 version. Note that you need to log in to the Oracle database before downloading the package.

Figure 11-2

Step 2 Install the JDK.

Double-click the downloaded **.exe** file to install the JDK and click **Next**.

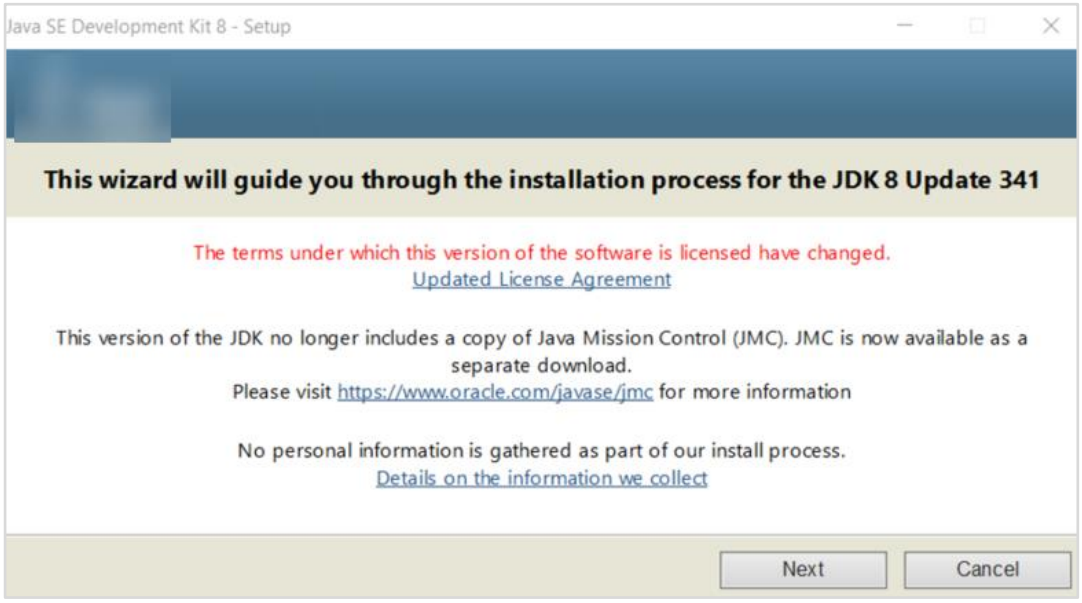


Figure 11-3

During the installation, you need to select the JDK and JRE installation paths. In this example, you can select the default path or a proper path on the local host. Then, click **Next** and wait until the installation is complete.

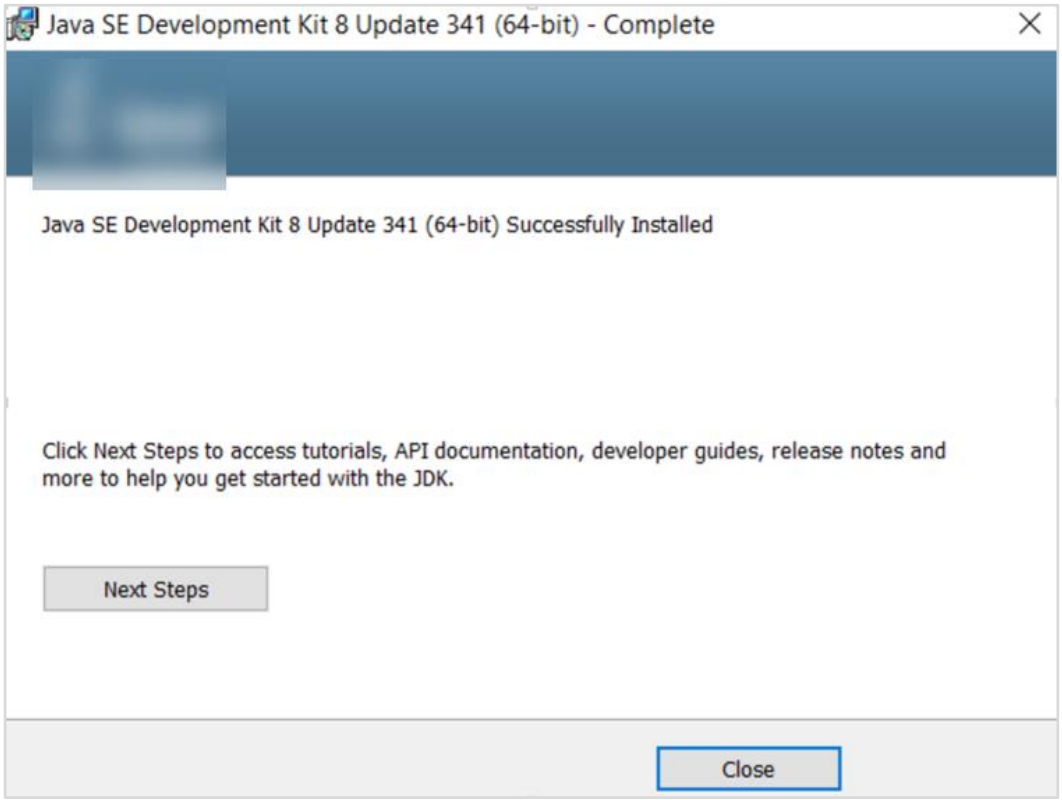


Figure 11-4

After the installation is complete, click **Close**.

Step 3 Configure JDK environment variables.

Configure JDK environment variables on your computer as follows: Open **This PC** on the desktop, click **Computer**, and select **System properties**. In the displayed window, choose **Advanced system settings**. In the dialog box that is displayed, click **Environment Variables**.

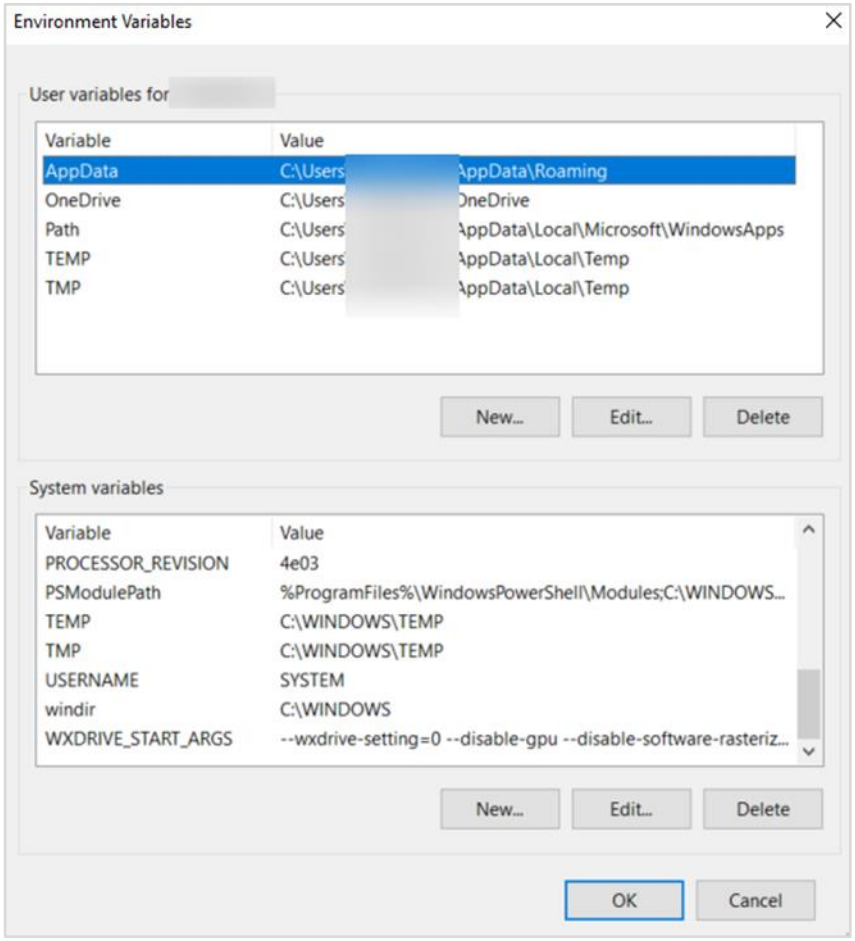


Figure 11-5

Click **New** in the **System variables** area. Set **Variable name** to **JAVA_HOME** (all uppercase letters) and **Variable value** to the JDK installation path. Click **OK**.

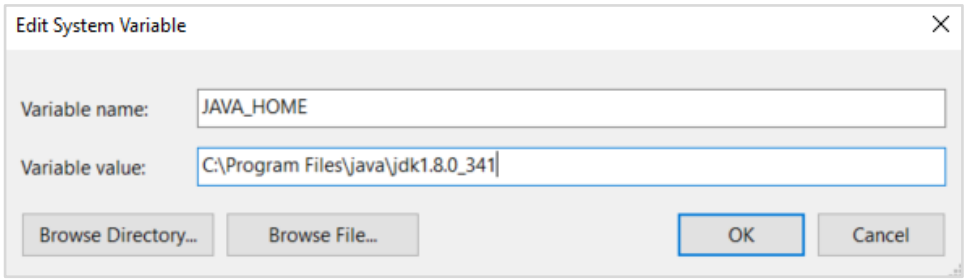


Figure 11-6

Double-click the **Path** variable in the **System variables** area, add **%JAVA_HOME%\bin** to the end of the original value, and click **OK**.

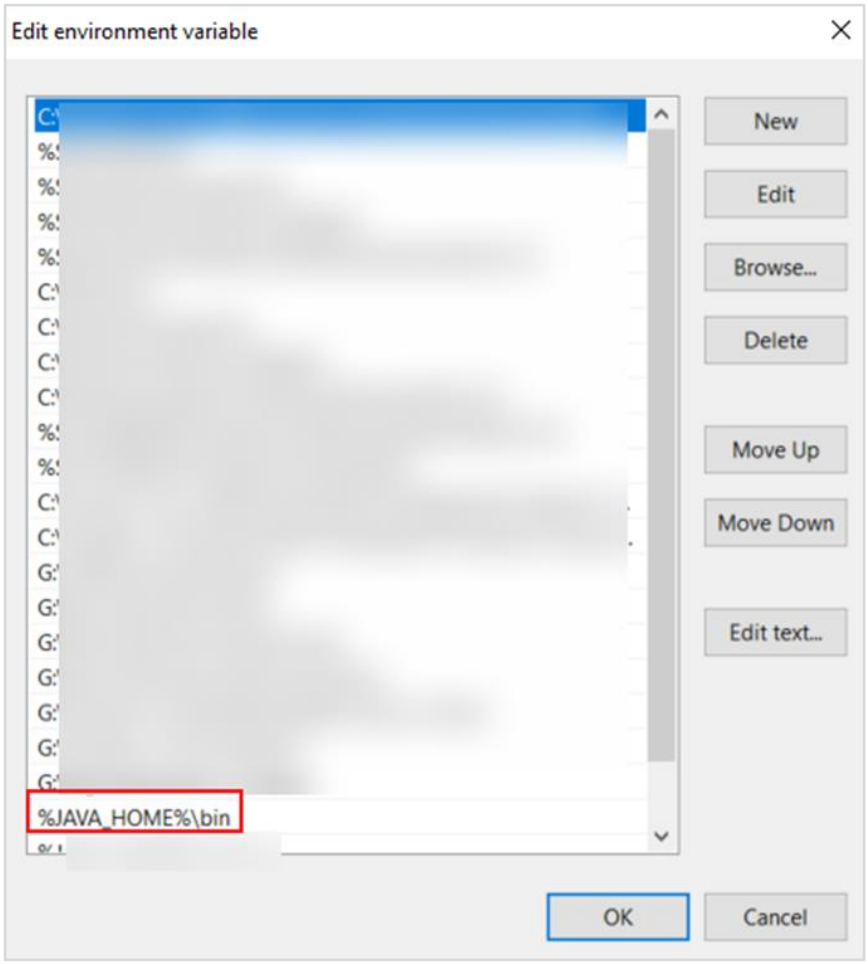


Figure 11-7

Step 4 Check that the JDK is installed successfully.

Choose **Start** > **Run**, enter **cmd**, and press **Enter**. In the displayed dialog box, enter the **java -version** command.

```
java -version
```

If the Java version information is displayed, the installation is successful.

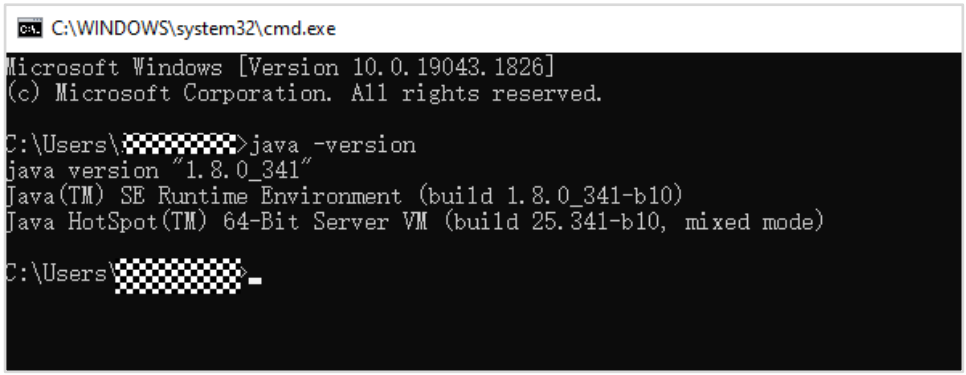


Figure 11-8

11.1.2 Installing Maven

Step 1 Install Maven.

Apache Maven is a software project management and comprehension tool. It can manage a project's build, reporting and documentation from a central piece of information. In short, Maven is one of the tools for managing Java projects.

After Maven is used, third-party JAR files such as **spring.jar** and **hibernate.jar** do not need to be copied to the **lib** directory of the project each time. The Maven configuration file can be used to automatically import JAR files to the project.

Download the latest version at **http://maven.apache.org/download.cgi** and decompress it to a proper directory on the PC, for example, **G:\Big Data\apache-maven\apache-maven-3.8.6**.

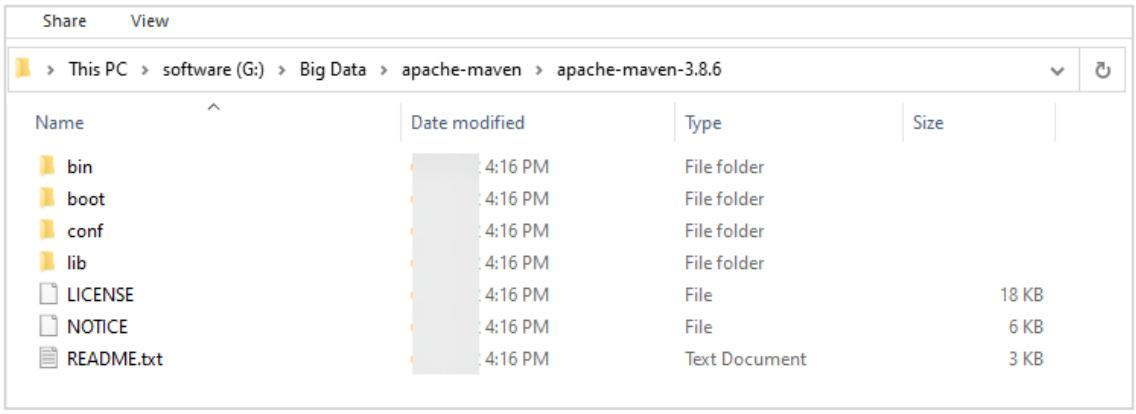


Figure 11-9

Add **MAVEN_HOME** or **M2_HOME** to system environment variables. Its value is the Maven installation directory.

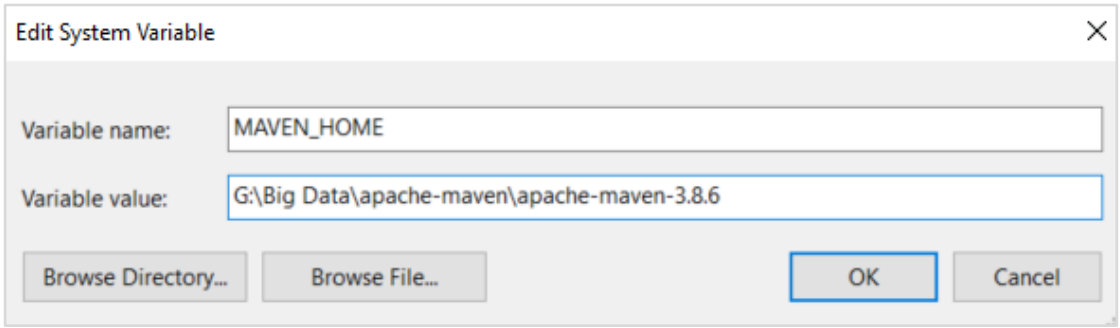


Figure 11-10

Find **Path** in the **System variables** area. Click **Path**, set the new environment variable to **%MAVEN_HOME %\bin**, and click **OK**.

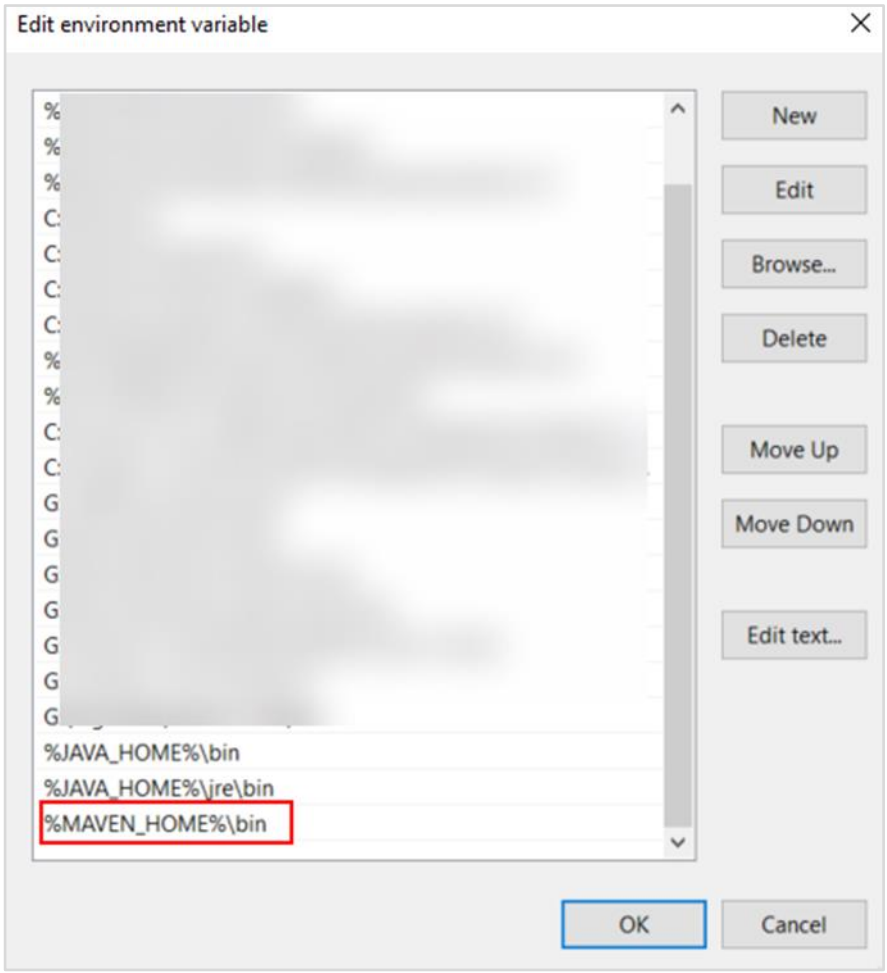


Figure 11-11

Step 2 Verify that Maven is installed.

Press **Win**+**R** to open the **Run** window, enter **cmd**, and run the following command:

```
mvn  -version
```

If the Maven information is displayed, the installation is successful.

Figure 11-12

11.1.3 Installing Eclipse

Step 1 Download Eclipse.

Visit https://www.eclipse.org/downloads/packages/, and select the version to download.

Select a 64-bit version to download. If your computer is 32-bit, click **Download Packages** and select a 32-bit version to download.

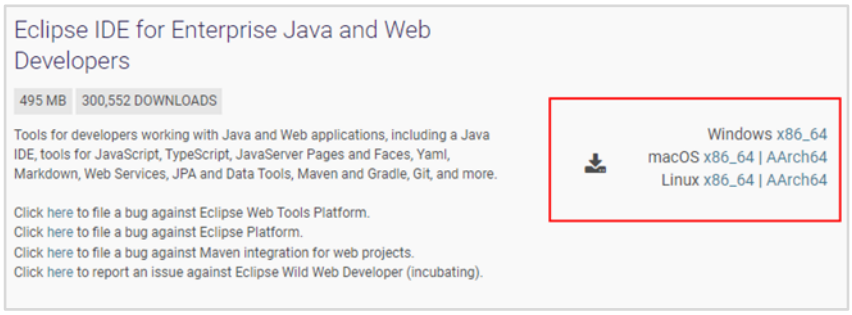


Figure 11-13

Step 2 Decompress the Eclipse package.

Decompress the downloaded package and go to the folder. The following figure shows the Eclipse startup program.

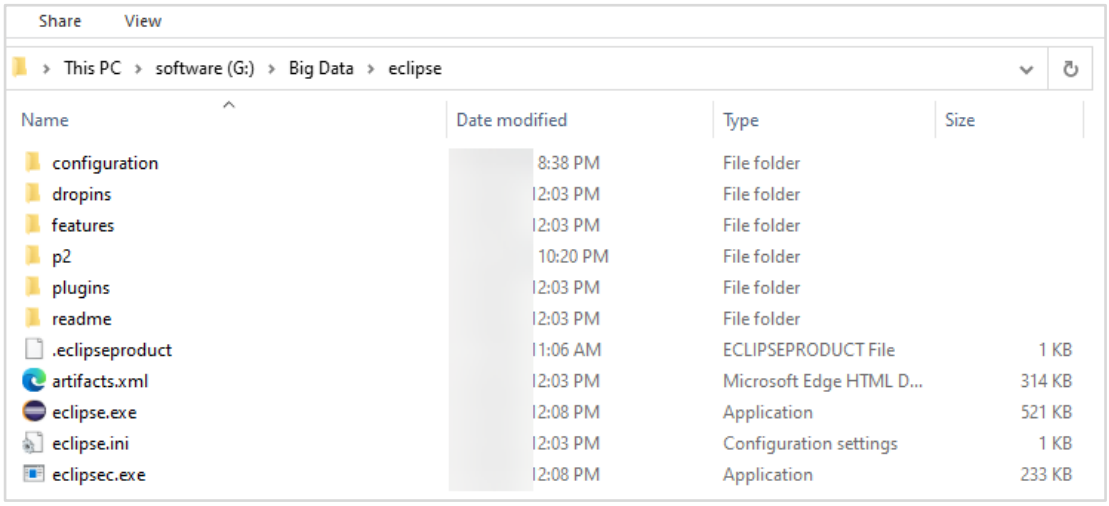


Figure 11-14

Double-click the program to start it. If you open the tool for the first time, you need to configure a workspace. You can select another location or use the default drive C. Then click **OK**.

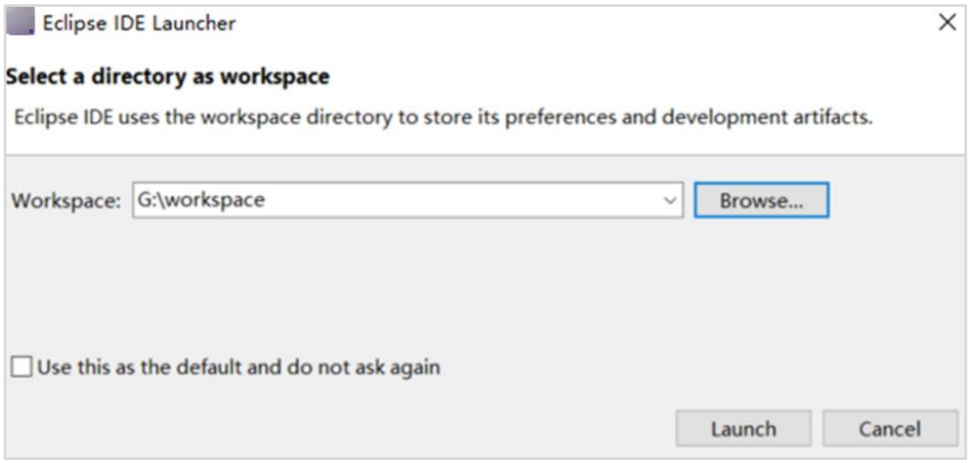


Figure 11-15

Step 3 Install the Maven plug-in in Eclipse.

In Eclipse, choose **Help** > **Eclipse Marketplace**, search for **Maven**, and click **Install**.

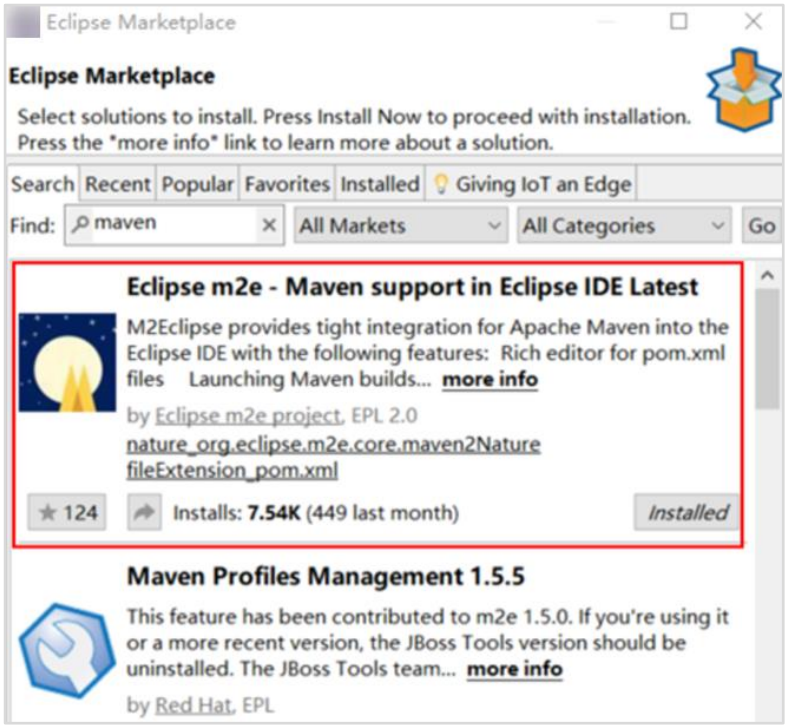


Figure 11-16

Step 4 Configure Maven.

Choose **Window** > **Preferences** > **Maven** > **User Settings** and configure Maven in the installation directory.

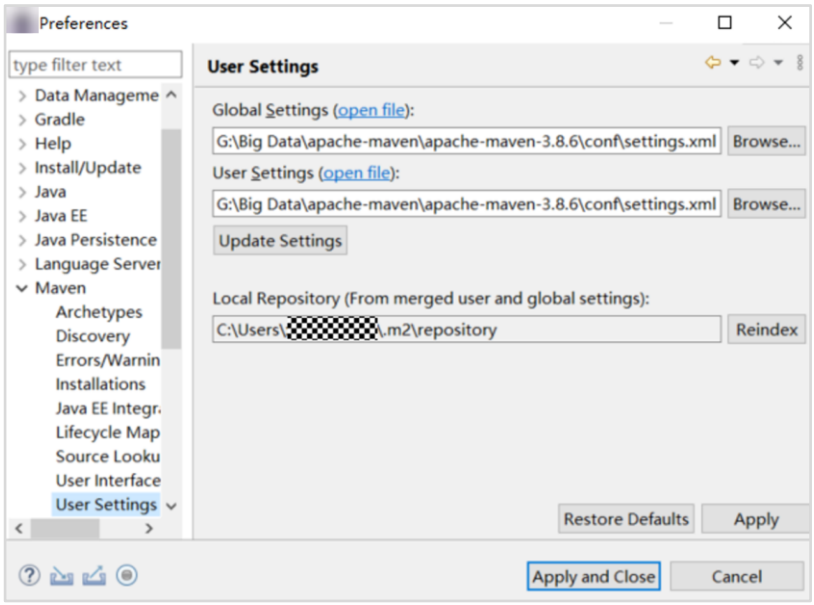


Figure 11-17

Step 5 Download the MRS 3.1.0 sample code.

The address for downloading the sample project of MRS on Huawei Cloud is https://github.com/huaweicloud/huaweicloud-mrs-example/tree/mrs-3.1.0.

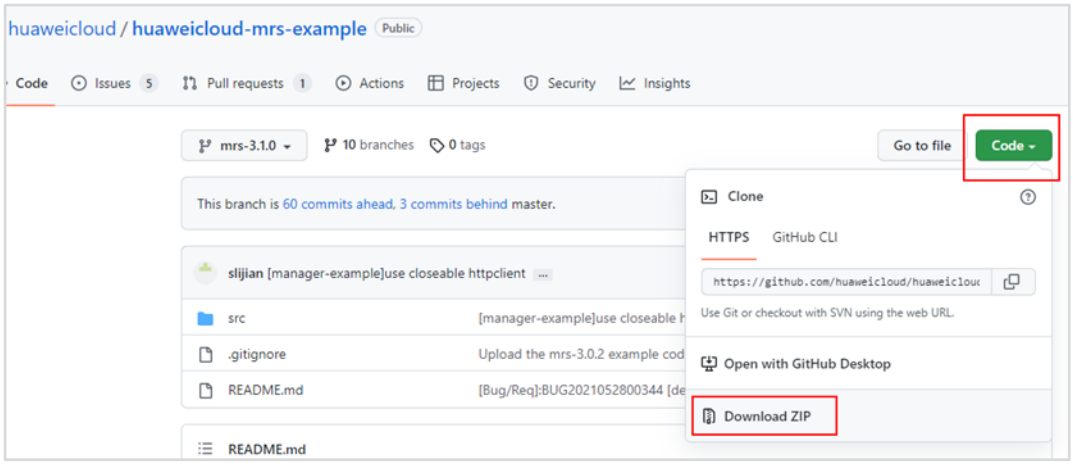


Figure 11-18

Download the ZIP package and decompress it.

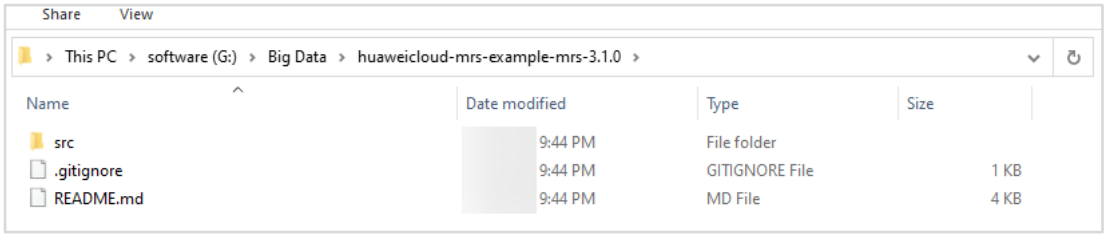


Figure 11-19

11.1.4 Importing the MRS 3.1.0 Sample Project to Eclipse

Step 1 Create a working set in Eclipse.

Start Eclipse, choose **File** > **New** > **Java Working Set**, enter the name, for example, **MRS2.0Demo**, and click **Finish**.

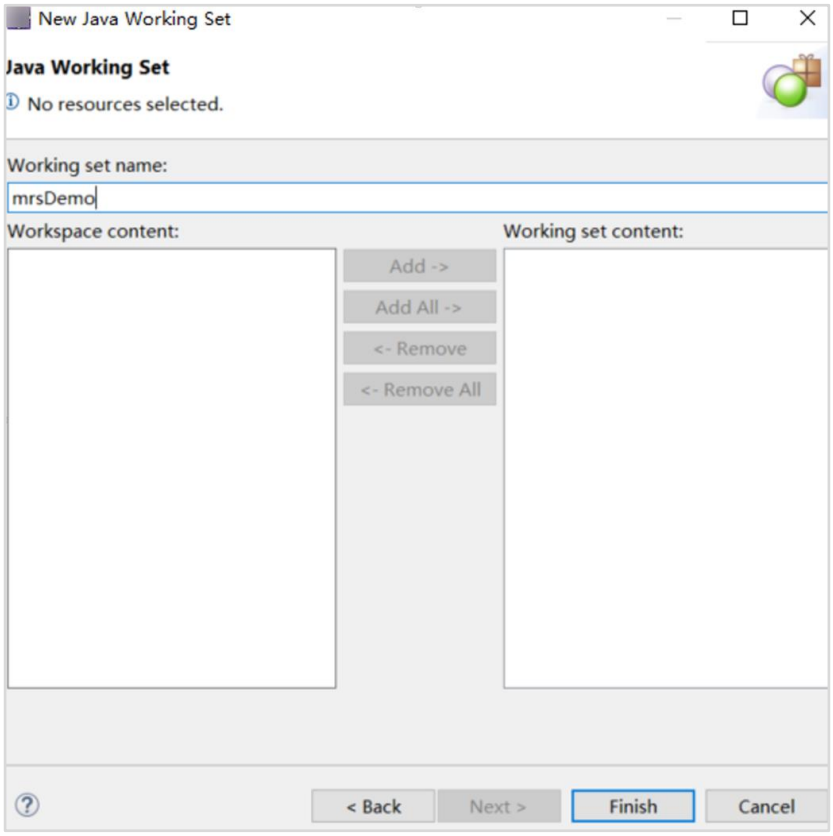


Figure 11-20

Step 2 Import a sample project to Eclipse.

Decompress the package and open **Eclipse**. Then choose **File** > **Import**.

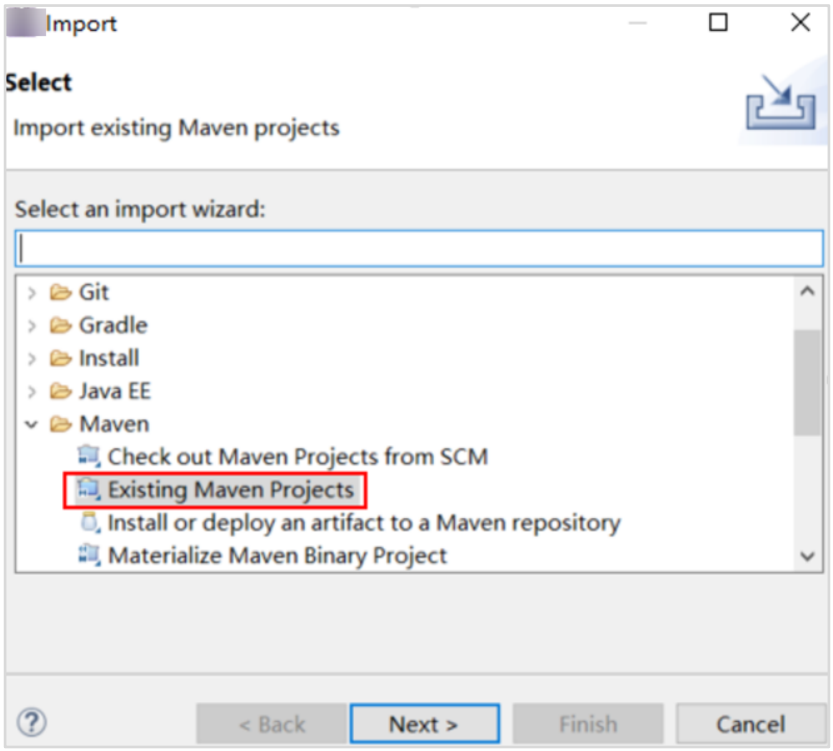


Figure 11-21

Click **Browse**, select the **huaweicloud-mrs-example-mrs-3.0** sample project folder in the decompressed package, select **Add project to working set**, select the **Java Working Set** created in the previous step, and click **Finish**.

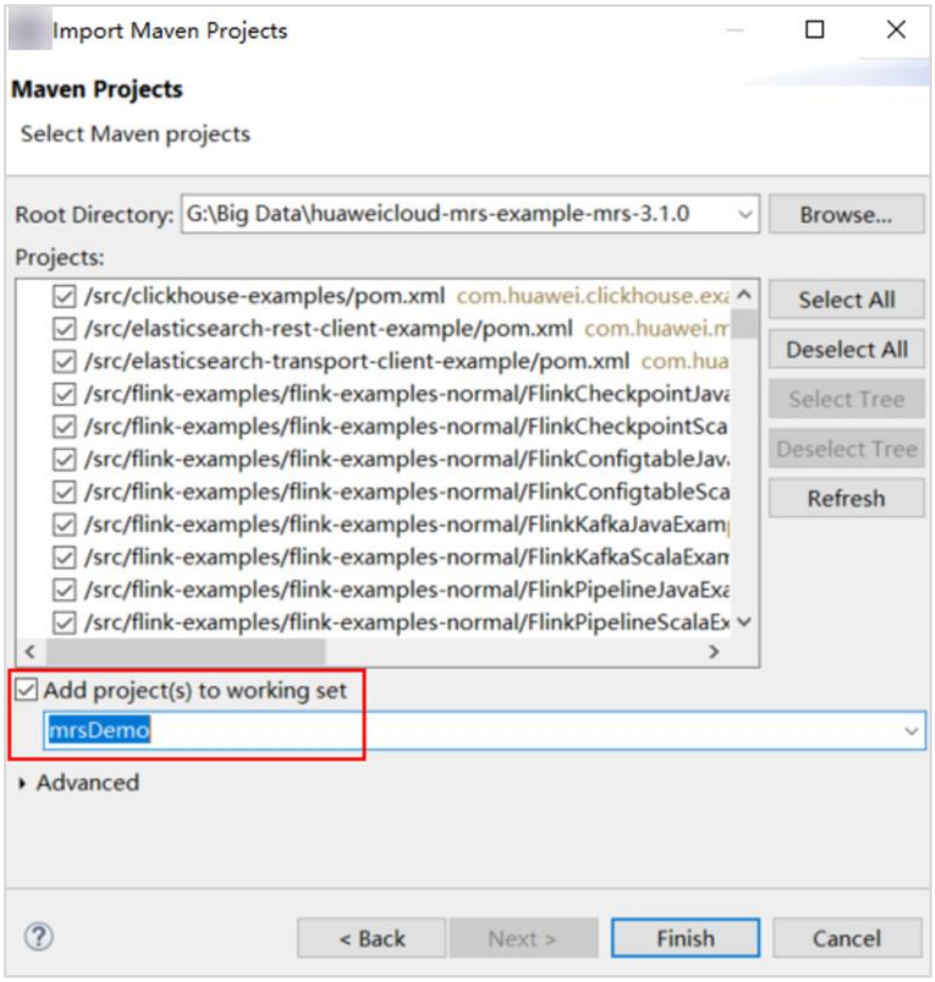


Figure 11-22

Wait until the Maven dependency package is loaded.

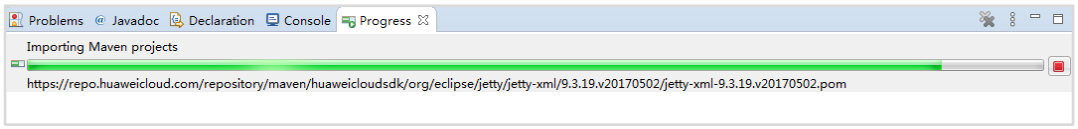


Figure 11-23

If an error is reported, ignore it and click **OK**.

Step 3 Modify the **pom** file.

Double-click the **pom** file in the **MRTest** project.

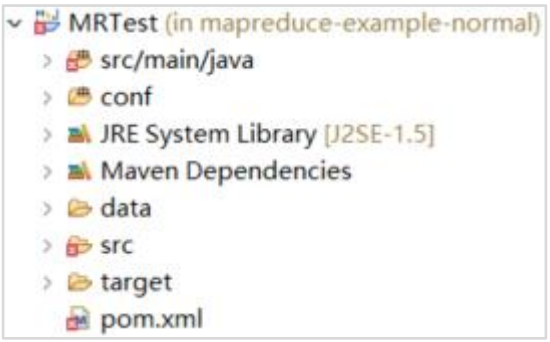


Figure 11-24

Switch to the **pom.xml** page, add the following code, and put the **repositories** code after the **dependencies** tag.

```
<repositories>
        <repository>
            <id>huaweicloudsdk</id>
            <url>https://mirrors.huaweicloud.com/repository/maven/huaweicloudsdk/</url>
            <releases><enabled>true</enabled></releases>
            <snapshots><enabled>true</enabled></snapshots>
        </repository>
        <repository>
            <id>central</id>
            <name>Mavn Centreal</name>
            <url>https://repo1.maven.org/maven2/</url>
        </repository>
</repositories>
```

For details about the code, see https://support.huaweicloud.com/devg3-mrs/mrs_07_010002.html.

The modifications are as follows.

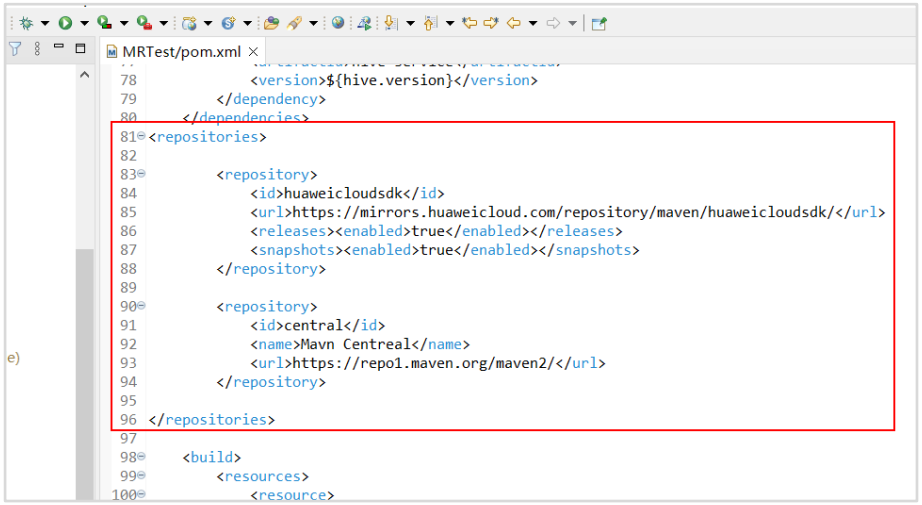


Figure 11-25

After saving the file, wait for the Eclipse JAR file to be downloaded. Keep the network connection and the Maven JAR file will be downloaded from the Huawei image repository.

Step 4 Modify the Java dependency.

Right-click the project name and choose **Build Path** > **Configure Build Path** from the shortcut menu.

Select **JRE System Library[J@SE-1.5]** and click **Remove**. Click **Add Library**, select **JRE System Library**, and click **Apply and Close**.

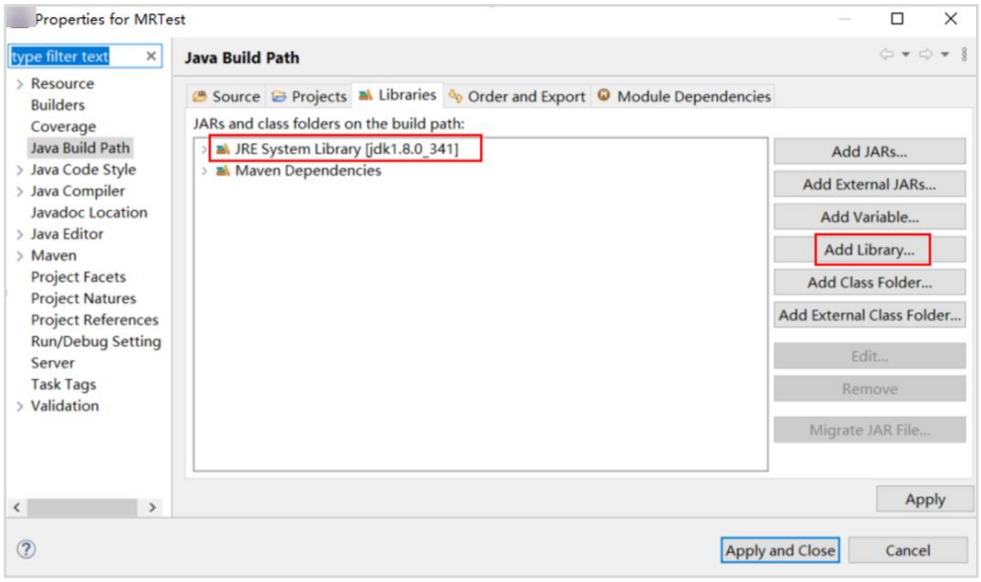


Figure 11-26

Select **Java Compiler**, set **Compiler compliance level** to **1.8**, and click **Apply and Close**.

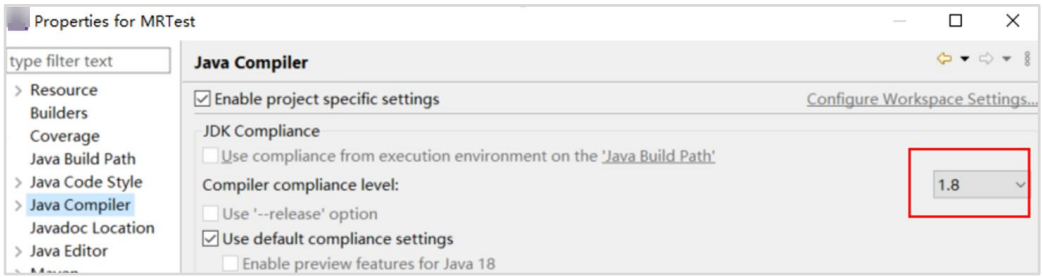


Figure 11-27

The project architecture is as follows.

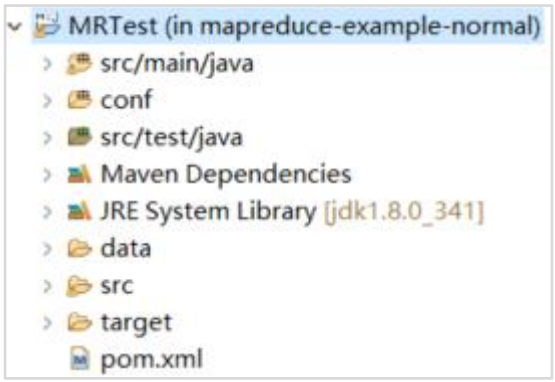


Figure 11-28

The configuration is complete.

11.2 How Can I Bind an EIP to an ECS?

Step 1 Access the cluster node management page.

Click the cluster name in the cluster list and click **Nodes**.

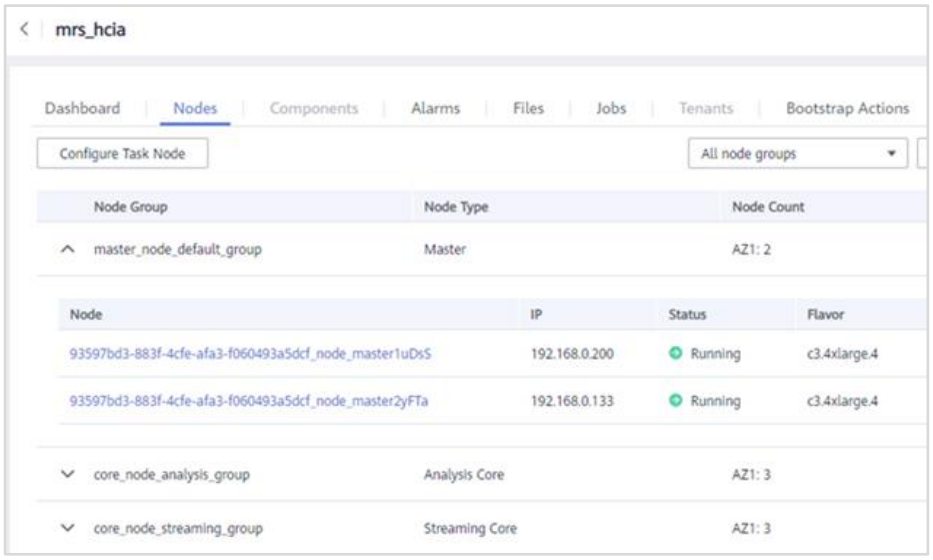


Figure 11-29

Step 2 Access the server.

Click a node name under **master_node_default_group** , as shown in the following figure.

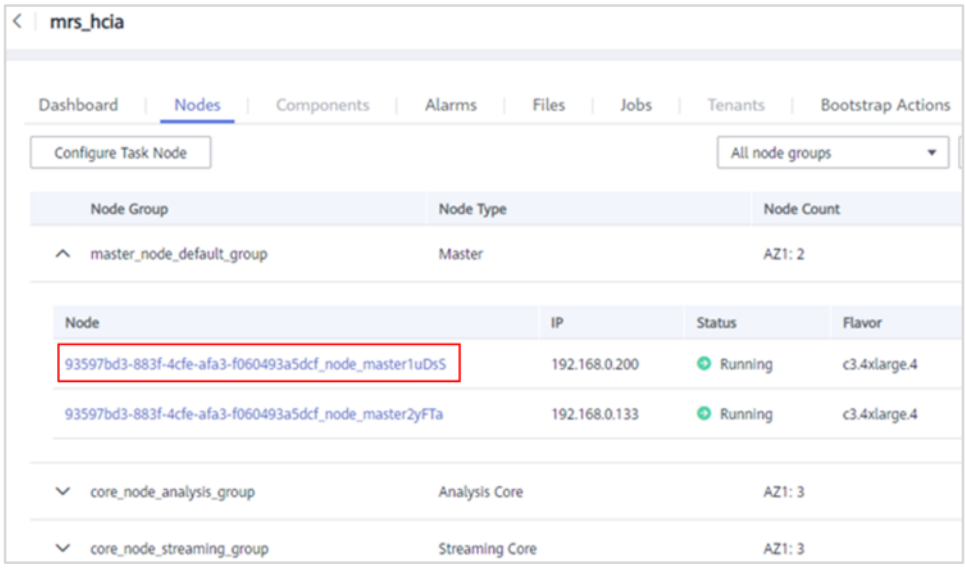


Figure 11-30

Select EIP and click **EIPs** to purchase an IP address. If you have purchased sufficient IP addresses when creating a cluster, click **Bind EIP**. Select **Pay-per-use**. After the purchase is complete, the Elastic Cloud Server page is displayed.

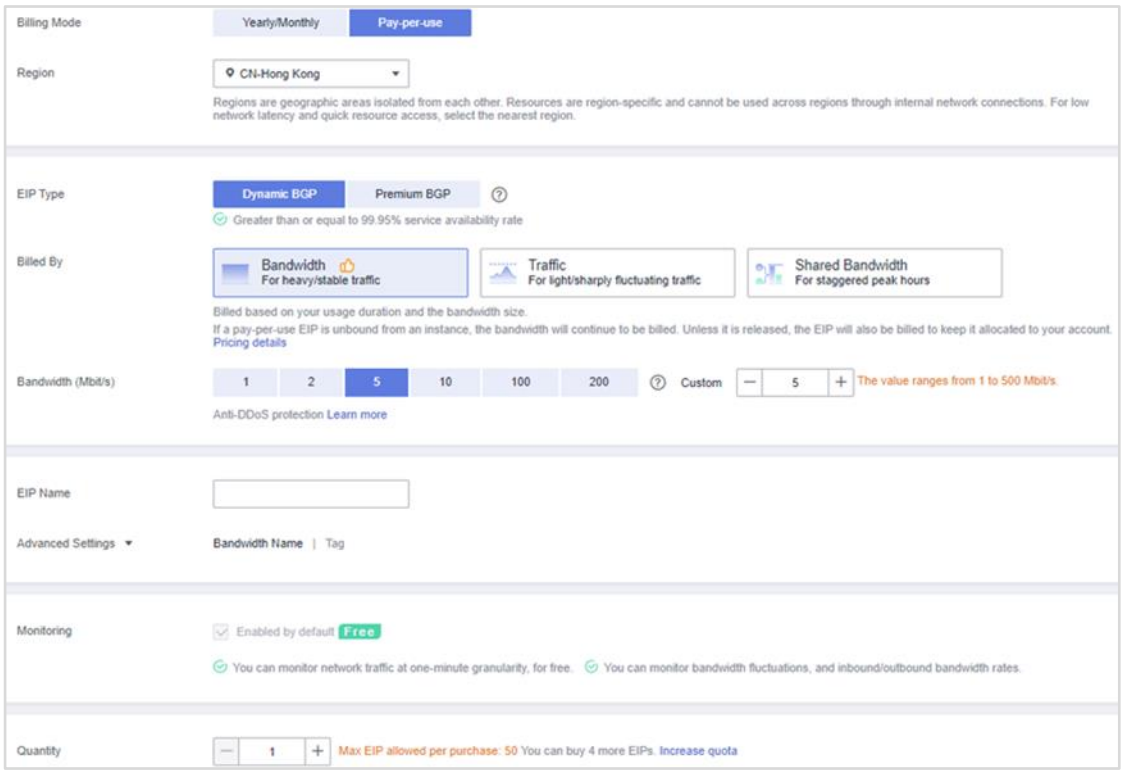


Figure 11-31

Step 3 Bind an EIP.

Click **Bind EIP**.

Figure 11-32

Select an IP address and click **OK**.

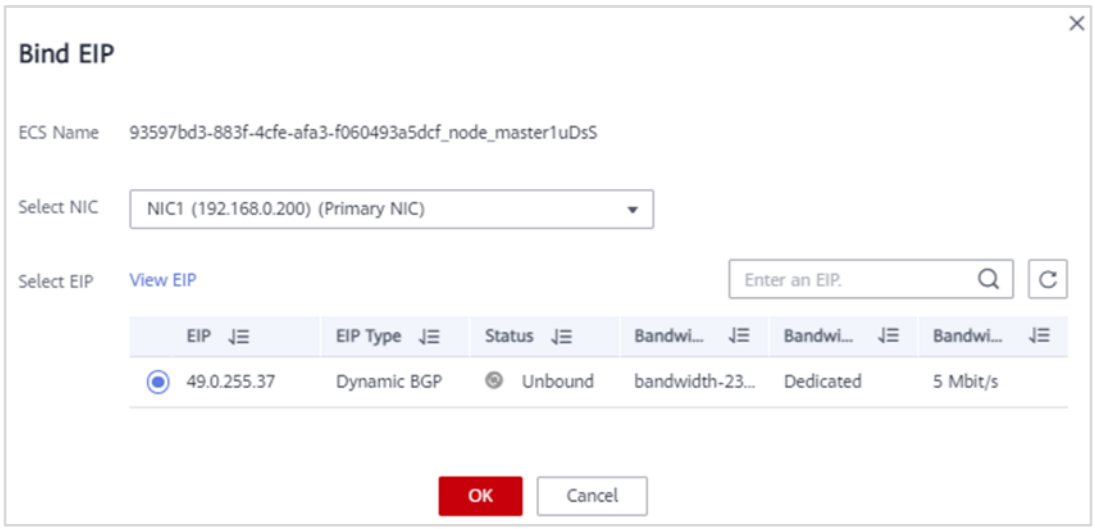


Figure 11-33

Refresh the page. You can see that the EIP is bound successfully.

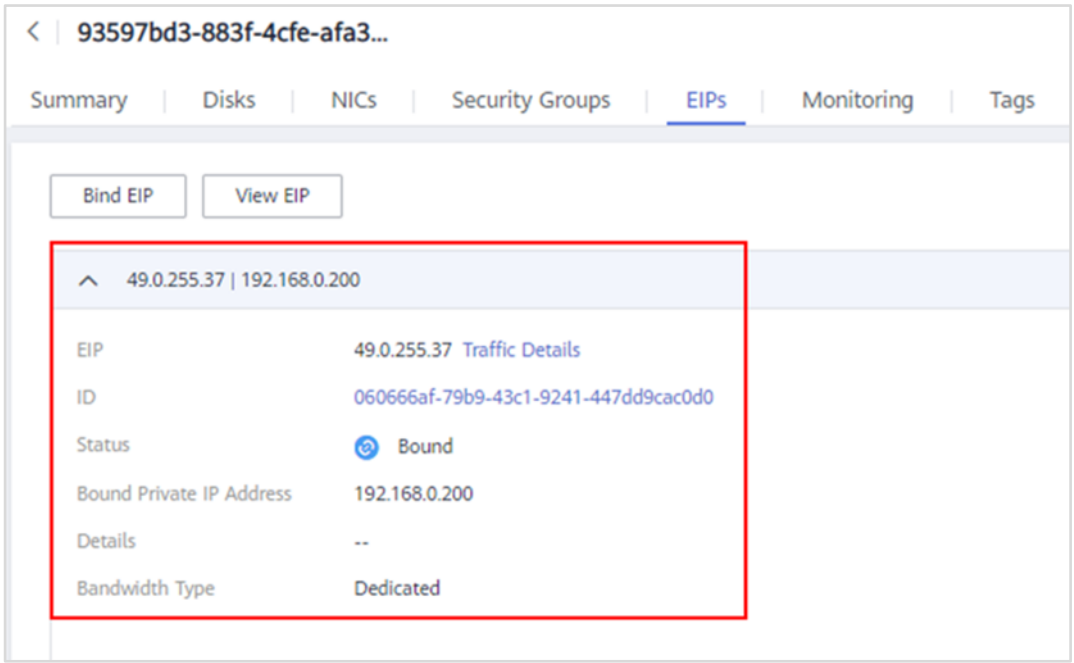


Figure 11-34

11.3 How Do I View the IP Address of ZooKeeper?

Step 1 Log in to the MRS cluster management page.

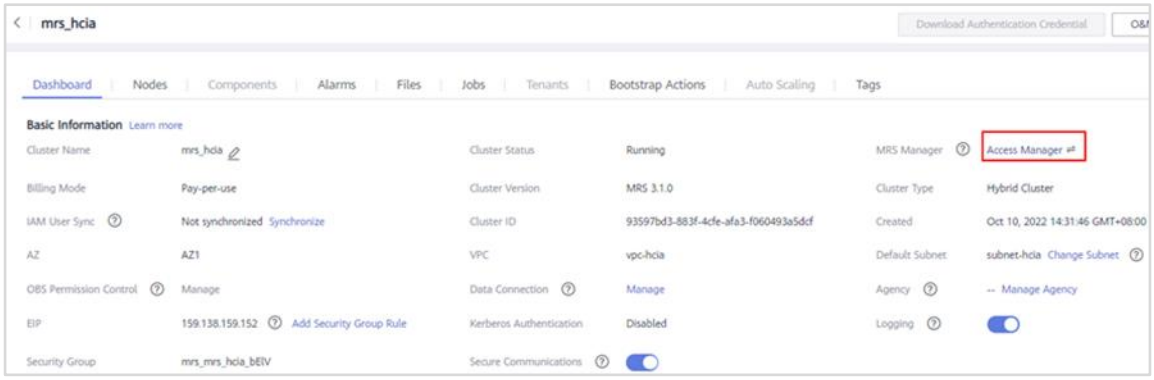


Figure 11-35

Step 2 Log in as user **admin**.

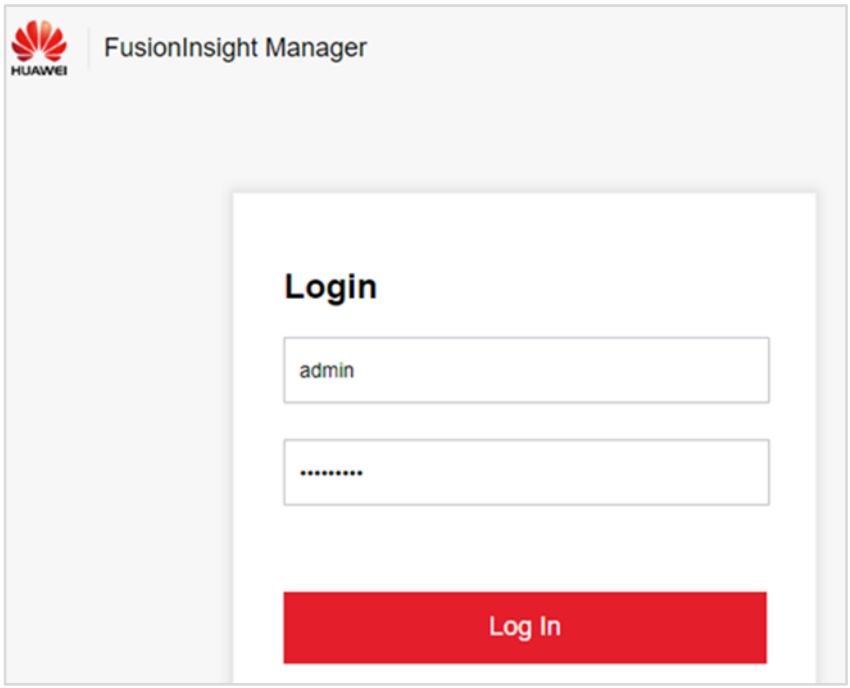


Figure 11-36

Step 3 Check the status of the ZooKeeper service.

On the **Homepage** page, click the ZooKeeper service in the **mrs_hcia** cluster. On the displayed ZooKeeper service page, click **Instance**. The service IP address of the ZooKeeper is displayed.

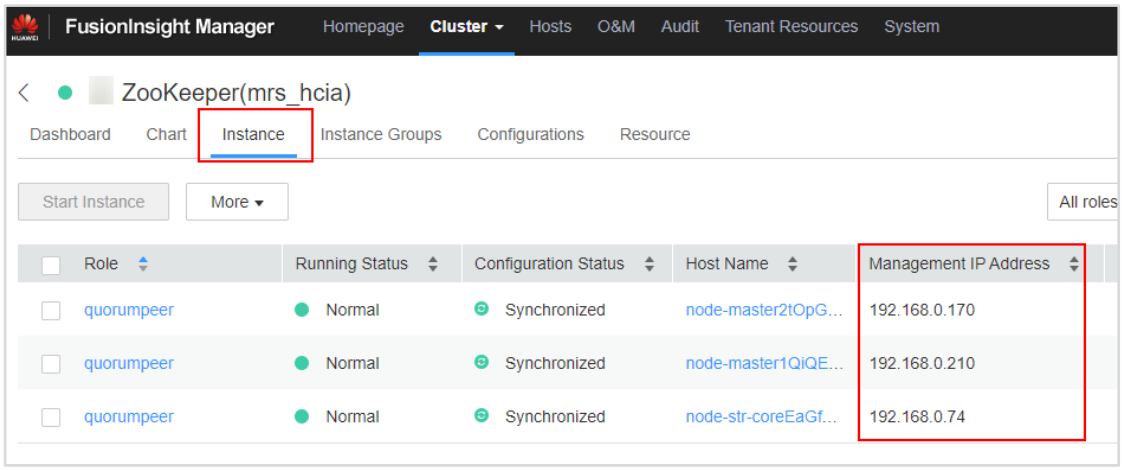


Figure 11-37

11.4 How Do I Check the Broker IP Address of a Kafka Instance?

Step 1 Log in to the MRS cluster management page.

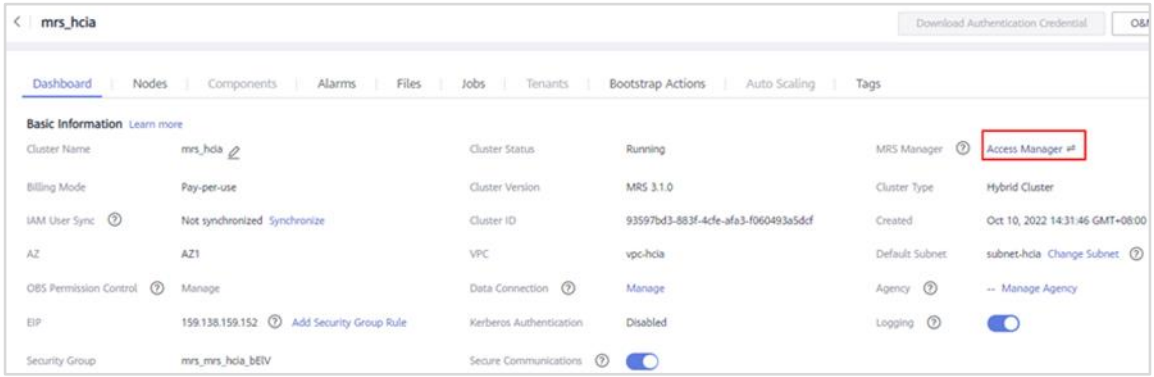


Figure 11-38

Step 2 Log in as user **admin**.

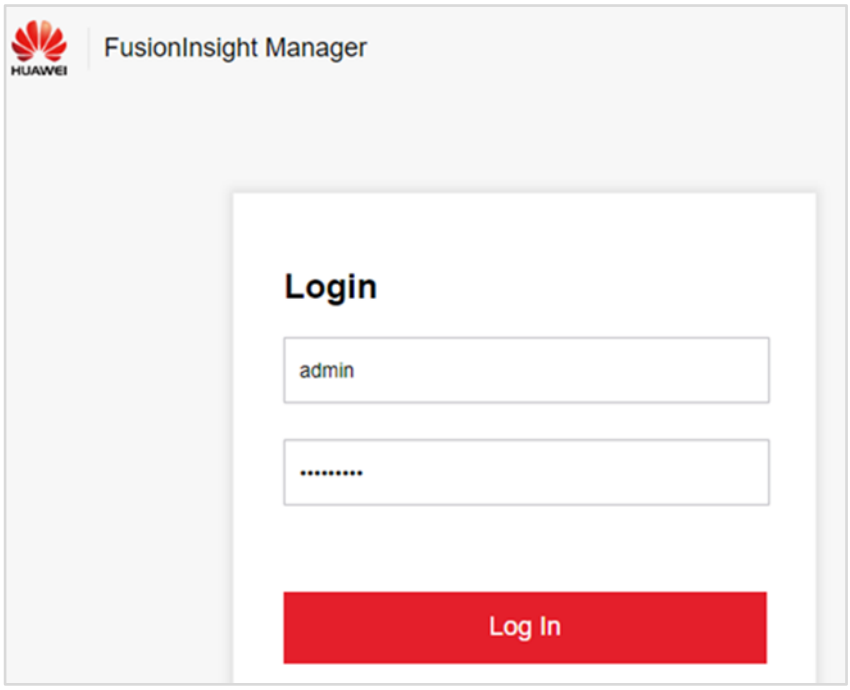


Figure 11-39

Step 3 View broker instances.

On the **Homepage** page, click the Kafka service in the **mrs_hcia** cluster. On the displayed Kafka service page, click **Instances**. The service IP address of the Kafka is displayed.

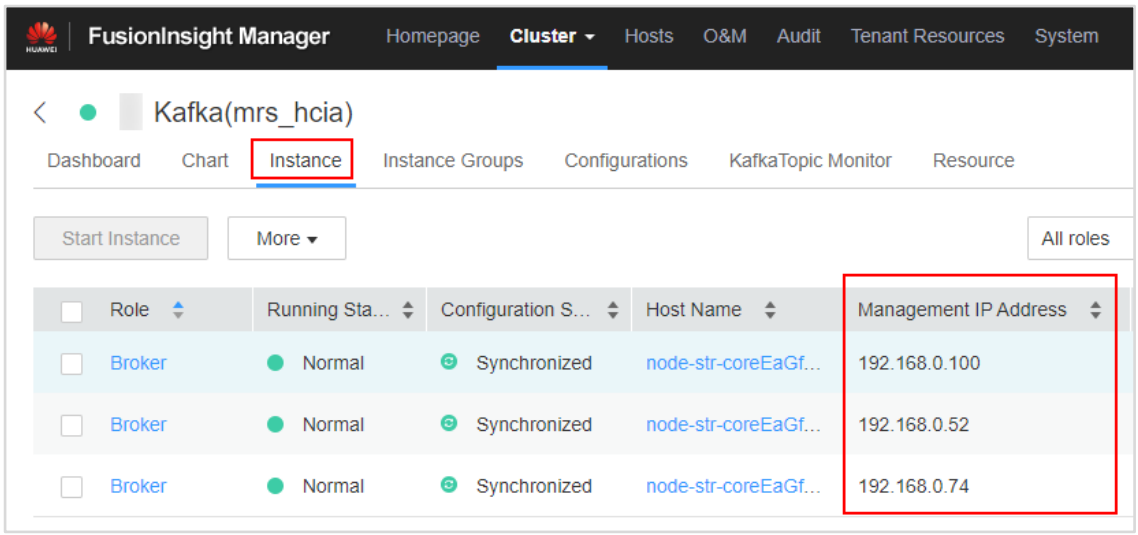


Figure 11-40

11.5 Common Linux Commands

cd /home: Go to the **/ home** directory.

cd ..: Go to the directory one level up from the current directory.

cd ../..: Go to the directory two levels up from the current directory.

cd: Go to your home directory.

cd ~user1: Go to another user's home directory.

cd -: Go to the the previous directory from the current working directory.

pwd: Display the working directory.

ls: Display the full list or content of your directory.

ls -F: Display files in the directory.

ls -l: Display details of files and directories.

ls -a: Display the whole list of the current directory including the hidden files.

ls *[0-9]*: Display file and directory names that contain digits.

tree: Display the tree structure of files and directories starting from the root directory.

lstree: Display the tree structure of files and directories starting from the root directory.

mkdir dir1: Create the **dir1** directory.

mkdir dir1 dir2: Create two directories concurrently.

mkdir -p /tmp/dir1/dir2: Create a directory with sub-directories.

rm -f file1: Delete the **file1** file.

rmdir dir1: Delete the **dir1** directory.

rm -rf dir1: Delete the **dir1** directory and its content.

rm -rf dir1 dir2: Delete two directories and their content concurrently.

mv dir1 new_dir: Rename or move a directory.

cp file1 file2: Copy a file.

cp dir/*: Copy all files in a directory to the current working directory.

cp -a /tmp/dir1: Copy a directory to the current working directory.

cp -a dir1 dir2: Copy a directory.

ln -s file1 lnk1: Create a soft link to a file or directory.

11.6 Yarn Application Operation Commands

Step 1 Run the following command to check all applications on Yarn:

```
yarn application -list
```

In the Flink exercise, the **yarn-session.sh** script is used to start a Flink cluster. This is a Yarn application. Run the following command to view the Yarn application:

```
[root@node-masterlnxXi ~]# yarn application -list
WARNING: YARN_CONF_DIR has been replaced by HADOOP_CONF_DIR. Using value of YARN_CONF_DIR.
WARNING: YARN_ROOT_LOGGER has been replaced by HADOOP_ROOT_LOGGER. Using value of YARN_ROOT_LOGGER.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.
30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/common/lib/slf4j-log4
j12-1.7.30.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]
Total number of applications (application-types: [], states: [SUBMITTED, ACCEPTED, RUNNING] and tags
: []):1
                Application-Id      Application-Name        Application-Type          User       Queu
e-User       Queue                   State             Final-State             Progress
           Tracking-URL
application_1663836345431_0008  Flink session cluster          Apache Flink          root
  root     default                 RUNNING               UNDEFINED                100%           ht
tp://192.168.0.89:32261
[root@node-masterlnxXi ~]#
```

Figure 11-41

Step 2 Run the following command to kill the Yarn application:

```
yarn application -kill   application id
```

For example, to kill a Flink cluster application, run the **-list** command to view the ID, and then run the **kill** command.

```
[root@node-masterlnxXi ~]# yarn application -kill application_1663836345431_0008
WARNING: YARN_CONF_DIR has been replaced by HADOOP_CONF_DIR. Using value of YARN_CONF_DIR.
WARNING: YARN_ROOT_LOGGER has been replaced by HADOOP_ROOT_LOGGER. Using value of YARN_ROOT_LOGGER.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/client/HDFS/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.
30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/Bigdata/client/HDFS/hadoop/share/hadoop/common/lib/slf4j-log4
j12-1.7.30.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]
Killing application application_1663836345431_0008
2[redacted],519 INFO impl.YarnClientImpl: Killed application application_1663836345431_0008
[root@node-masterlnxXi ~]#
```

Figure 11-42